axiom™

The 30 Year Horizon

Manuel Bronstein *William Burge* *Timothy Daly*
James Davenport *Michael Dewar* *Martin Dunstan*
Albrecht Fortenbacher *Patrizia Gianni* *Johannes Grabmeier*
Jocelyn Guidry *Richard Jenks* *Larry Lambe*
Michael Monagan *Scott Morrison* *William Sit*
Jonathan Steinbach *Robert Sutor* *Barry Trager*
Stephen Watt *Jim Wen* *Clifton Williamson*

Volume 10: Axiom Algebra: Numerics

Inclusion of names in the list of credits is based on historical information and is as accurate as possible. Inclusion of names does not in any way imply an endorsement but represents historical influence on Axiom development.

Cyril Alberga
George Andrews
Yurij Baransky
Gilbert Baumslag
Mark Botch
Martin Brock
William Burge
Bruce Char
Gregory V. Chudnovsky
Don Coppersmith
Gary Cornell
Timothy Daly Sr.
Jean Della Dora
Claire DiCrescendo
Martin Dunstan
Robert Edwards
Richard Fateman
Brian Ford
Constantine Frangos
Marc Gaetano
Patricia Gianni
Laureano Gonzalez-Vega
Matt Grayson
Oswald Gschnitzer
Vilya Harvey
Ralf Hemmecke
Pietro Iglio
Grant Keady
Klaus Kusche
Frederic Lehobey
Rudiger Loos
Camm Maguire
Ian Meikle
Gerard Milmeister
Michael Monagan
Mark Murray
John Nelder
Jinzhong Niu
Julian A. Padget
Susan Pelzel
Claude Quitte
Renaud Rioboo
Simon Robinson
Philip Santas
Gerhard Schneider
Fritz Schwarz
Elena Smirnova
Robert Sutor
James Thatcher
Dylan Thurston
Gregory Vanuxem
Juergen Weiss
James Wen
John M. Wiley
Stephen Wilson
Sandra Wityak

Roy Adler
Henry Baker
David R. Barton
Fred Blair
Alexandre Bouyer
Manuel Bronstein
Quentin Carpent
Cheekai Chin
Josh Cohen
George Corliss
Meino Cramer
Timothy Daly Jr.
Gabriel Dos Reis
Sam Dooley
Brian Dupee
Heow Eide-Goodman
Bertfried Fauser
Albrecht Fortenbacher
Timothy Freeman
Rudiger Gebauer
Holger Gollan
Stephen Gortler
James Griesmer
Jocelyn Guidry
Satoshi Hamaguchi
Henderson
Richard Jenks
Tony Kennedy
Bernhard Kutzler
Michel Levaud
Michael Lucks
Bob McElrath
David Mentre
Mohammed Mobarak
Marc Moreno-Maza
William Naylor
Godfrey Nolan
Michael O'Connor
Bill Page
Michel Petitot
Norman Ramsey
Jean Rivlin
Michael Rothstein
Alfred Scheerhorn
Martin Schoenert
Nick Simicich
Jonathan Steinbach
Moss E. Sweedler
Baldir Thomas
Barry Trager
Bernhard Wall
M. Weller
Thorsten Werther
Berhard Will
Shmuel Winograd
Waldemar Wiwianka

Richard Anderson
Stephen Balzac
Gerald Baumgartner
Vladimir Bondarenko
Peter A. Broadbery
Florian Bundschuh
Bob Caviness
David V. Chudnovsky
Christophe Conil
Robert Corless
Claire Di Crescenzo
James H. Davenport
Michael Dewar
Lionel Ducos
Dominique Duval
Lars Erickson
Stuart Feldman
George Frances
Korrinn Fu
Kathy Gerber
Teresa Gomez-Diaz
Johannes Grabmeier
Vladimir Grinberg
Steve Hague
Martin Hassner
Antoine Hersen
Kai Kaminski
Paul Kosinski
Larry Lambe
Howard Levy
Richard Luczak
Michael McGettrick
Victor S. Miller
H. Michael Moeller
Scott Morrison
C. Andrew Neff
Arthur Norman
Kostas Oikonomou
Jaap Weel
Didier Pinchon
Michael Richardson
Nicolas Robidoux
Martin Rubey
William Schelter
Marshall Schor
William Sit
Christine Sundaresan
Eugene Surowitz
Mike Thomas
Themos T. Tsikas
Stephen Watt
Mark Wegman
Michael Wester
Clifton J. Williamson
Robert Wisbauer
Knut Wolf

Contents

1 Numerical Analysis [4] 1

2 Chapter Overview 3

3 Algebra Cover Code 5
- 3.1 package BLAS1 BlasLevelOne . . . 5
  - 3.1.1 BlasLevelOne (BLAS1) . . . 9
- 3.2 dcabs1 BLAS . . . 11
- 3.3 lsame BLAS . . . 14
- 3.4 xerbla BLAS . . . 14

4 BLAS Level 1 15
- 4.1 dasum BLAS . . . 15
- 4.2 daxpy BLAS . . . 26
- 4.3 dcopy BLAS . . . 36
- 4.4 ddot BLAS . . . 43
- 4.5 dnrm2 BLAS . . . 48
- 4.6 drotg BLAS . . . 52
- 4.7 drot BLAS . . . 56
- 4.8 dscal BLAS . . . 60
- 4.9 dswap BLAS . . . 64
- 4.10 dzasum BLAS . . . 69
- 4.11 dznrm2 BLAS . . . 73
- 4.12 icamax BLAS . . . 77
- 4.13 idamax BLAS . . . 81
- 4.14 isamax BLAS . . . 85
- 4.15 izamax BLAS . . . 89
- 4.16 zaxpy BLAS . . . 93
- 4.17 zcopy BLAS . . . 97
- 4.18 zdotc BLAS . . . 101
- 4.19 zdotu BLAS . . . 105
- 4.20 zdscal BLAS . . . 109
- 4.21 zrotg BLAS . . . 112
- 4.22 zscal BLAS . . . 116

4.23 zswap BLAS 119

5 BLAS Level 2 **123**
5.1 dgbmv BLAS 123
5.2 dgemv BLAS 133
5.3 dger BLAS 142
5.4 dsbmv BLAS 147
5.5 dspmv BLAS 158
5.6 dspr2 BLAS 168
5.7 dspr BLAS 177
5.8 dsymv BLAS 184
5.9 dsyr2 BLAS 194
5.10 dsyr BLAS 203
5.11 dtbmv BLAS 210
5.12 dtbsv BLAS 223
5.13 dtpmv BLAS 237
5.14 dtpsv BLAS 251
5.15 dtrmv BLAS 265
5.16 dtrsv BLAS 277
5.17 zgbmv BLAS 289
5.18 zgemv BLAS 300
5.19 zgerc BLAS 310
5.20 zgeru BLAS 315
5.21 zhbmv BLAS 320
5.22 zhemv BLAS 331
5.23 zher2 BLAS 341
5.24 zher BLAS 354
5.25 zhpmv BLAS 364
5.26 zhpr2 BLAS 375
5.27 zhpr BLAS 392
5.28 ztbmv BLAS 402
5.29 ztbsv BLAS 419
5.30 ztpmv BLAS 436
5.31 ztpsv BLAS 452
5.32 ztrmv BLAS 469
5.33 ztrsv BLAS 484

6 BLAS Level 3 **501**
6.1 dgemm BLAS 501
6.2 dsymm BLAS 511
6.3 dsyr2k BLAS 522
6.4 dsyrk BLAS 534
6.5 dtrmm BLAS 545
6.6 dtrsm BLAS 559
6.7 zgemm BLAS 575
6.8 zhemm BLAS 590

6.9 zher2k BLAS . . . 602
6.10 zherk BLAS . . . 620
6.11 zsymm BLAS . . . 635
6.12 zsyr2k BLAS . . . 646
6.13 zsyrk BLAS . . . 658
6.14 ztrmm BLAS . . . 669
6.15 ztrsm BLAS . . . 686

7 LAPACK 705
7.1 dbdsdc LAPACK . . . 705
7.2 dbdsqr LAPACK . . . 720
7.3 ddisna LAPACK . . . 749
7.4 dgebak LAPACK . . . 755
7.5 dgebal LAPACK . . . 761
7.6 dgebd2 LAPACK . . . 769
7.7 dgebrd LAPACK . . . 778
7.8 dgeev LAPACK . . . 786
7.9 dgeevx LAPACK . . . 801
7.10 dgehd2 LAPACK . . . 821
7.11 dgehrd LAPACK . . . 826
7.12 dgelq2 LAPACK . . . 834
7.13 dgelqf LAPACK . . . 838
7.14 dgeqr2 LAPACK . . . 843
7.15 dgeqrf LAPACK . . . 847
7.16 dgesdd LAPACK . . . 852
7.17 dgesvd LAPACK . . . 899
7.18 dgesv LAPACK . . . 1042
7.19 dgetf2 LAPACK . . . 1046
7.20 dgetrf LAPACK . . . 1051
7.21 dgetrs LAPACK . . . 1056
7.22 dhseqr LAPACK . . . 1060
7.23 dlabad LAPACK . . . 1075
7.24 dlabrd LAPACK . . . 1077
7.25 dlacon LAPACK . . . 1092
7.26 dlacpy LAPACK . . . 1098
7.27 dladiv LAPACK . . . 1102
7.28 dlaed6 LAPACK . . . 1104
7.29 dlaexc LAPACK . . . 1114
7.30 dlahqr LAPACK . . . 1127
7.31 dlahrd LAPACK . . . 1145
7.32 dlaln2 LAPACK . . . 1152
7.33 dlamch LAPACK . . . 1171
7.34 dlamc1 LAPACK . . . 1175
7.35 dlamc2 LAPACK . . . 1181
7.36 dlamc3 LAPACK . . . 1189
7.37 dlamc4 LAPACK . . . 1191

7.38 dlamc5 LAPACK 1194
7.39 dlamrg LAPACK 1198
7.40 dlange LAPACK 1202
7.41 dlanhs LAPACK 1207
7.42 dlanst LAPACK 1212
7.43 dlanv2 LAPACK 1217
7.44 dlapy2 LAPACK 1222
7.45 dlaqtr LAPACK 1224
7.46 dlarfb LAPACK 1253
7.47 dlarfg LAPACK 1269
7.48 dlarf LAPACK 1273
7.49 dlarft LAPACK 1276
7.50 dlarfx LAPACK 1285
7.51 dlartg LAPACK 1332
7.52 dlas2 LAPACK 1337
7.53 dlascl LAPACK 1341
7.54 dlasd0 LAPACK 1349
7.55 dlasd1 LAPACK 1357
7.56 dlasd2 LAPACK 1364
7.57 dlasd3 LAPACK 1379
7.58 dlasd4 LAPACK 1394
7.59 dlasd5 LAPACK 1430
7.60 dlasd6 LAPACK 1437
7.61 dlasd7 LAPACK 1446
7.62 dlasd8 LAPACK 1459
7.63 dlasda LAPACK 1469
7.64 dlasdq LAPACK 1485
7.65 dlasdt LAPACK 1495
7.66 dlaset LAPACK 1500
7.67 dlasq1 LAPACK 1504
7.68 dlasq2 LAPACK 1509
7.69 dlasq3 LAPACK 1531
7.70 dlasq4 LAPACK 1547
7.71 dlasq5 LAPACK 1561
7.72 dlasq6 LAPACK 1573
7.73 dlasr LAPACK 1584
7.74 dlasrt LAPACK 1600
7.75 dlassq LAPACK 1608
7.76 dlasv2 LAPACK 1612
7.77 dlaswp LAPACK 1618
7.78 dlasy2 LAPACK 1623
7.79 dorg2r LAPACK 1641
7.80 dorgbr LAPACK 1645
7.81 dorghr LAPACK 1653
7.82 dorgl2 LAPACK 1658
7.83 dorglq LAPACK 1663

7.84 dorgqr LAPACK 1669
7.85 dorm2r LAPACK 1675
7.86 dormbr LAPACK 1680
7.87 dorml2 LAPACK 1688
7.88 dormlq LAPACK 1693
7.89 dormqr LAPACK 1700
7.90 dtrevc LAPACK 1707
7.91 dtrexc LAPACK 1753
7.92 dtrsna LAPACK 1763
7.93 ieeeck LAPACK 1781
7.94 ilaenv LAPACK 1786
7.95 zlange LAPACK 1799
7.96 zlassq LAPACK 1804

8 Chunk collections **1809**

9 Index **1817**

New Foreword

On October 1, 2001 Axiom was withdrawn from the market and ended life as a commercial product. On September 3, 2002 Axiom was released under the Modified BSD license, including this document. On August 27, 2003 Axiom was released as free and open source software available for download from the Free Software Foundation's website, Savannah.

Work on Axiom has had the generous support of the Center for Algorithms and Interactive Scientific Computation (CAISS) at City College of New York. Special thanks go to Dr. Gilbert Baumslag for his support of the long term goal.

The online version of this documentation is roughly 1000 pages. In order to make printed versions we've broken it up into three volumes. The first volume is tutorial in nature. The second volume is for programmers. The third volume is reference material. We've also added a fourth volume for developers. All of these changes represent an experiment in print-on-demand delivery of documentation. Time will tell whether the experiment succeeded.

Axiom has been in existence for over thirty years. It is estimated to contain about three hundred man-years of research and has, as of September 3, 2003, 143 people listed in the credits. All of these people have contributed directly or indirectly to making Axiom available. Axiom is being passed to the next generation. I'm looking forward to future milestones.

With that in mind I've introduced the theme of the "30 year horizon". We must invent the tools that support the Computational Mathematician working 30 years from now. How will research be done when every bit of mathematical knowledge is online and instantly available? What happens when we scale Axiom by a factor of 100, giving us 1.1 million domains? How can we integrate theory with code? How will we integrate theorems and proofs of the mathematics with space-time complexity proofs and running code? What visualization tools are needed? How do we support the conceptual structures and semantics of mathematics in effective ways? How do we support results from the sciences? How do we teach the next generation to be effective Computational Mathematicians?

The "30 year horizon" is much nearer than it appears.

Tim Daly
CAISS, City College of New York
November 10, 2003 ((iHy))

Chapter 1

Numerical Analysis [4]

We can describe each number as x^* which has a machine-representable form which differs from the number x it is intended to represent. Quoting Householder we get:

$$x^* = \pm(x_1\beta^{-1} + x_2\beta^{-2} + \cdots + x_\lambda\beta^\lambda)\beta^\sigma$$

where β is the base, usually 2 or 10, λ is a positive integer, and σ is any integer, possibly zero. It may be that λ is fixed throughout the course of the computation, or it may vary, but in any case it is limited by practical considerations. Such a number will be called a representation. It may be that x^* is obtained by "rounding off" a number whose true value is x (for example, $x = 1/3$, $x^* = 0.33$), or that x^* is the result of measuring physically a quantity whose true value is x, or that x^* is the result of a computation intended to give an approximation to the quantity x.

Suppose one is interested in performing an operations ω upon a pair of numbers x and y. That is to say, $x\omega y$ may represent a product of x and y, a quotient of x by y, the yth power of x, ... In the numerical computation, however, one has only x^* and y^* upon which to operate, not x and y (or at least these are the quantitites upon which one does, in fact operate). Not only this, but often one does not even perform the strict operation ω, but rather a pseudo operation ω^*, which yields a rounded-off product, quotient, power, etc. Hence, instead of obtaining the desired result $x\omega y$, one obtains a result $x^*\omega^* y^*$.

The error in the result is therefore

$$x\omega y - x^*\omega^* y^* = (x\omega y - x^*\omega y^*) + (x^*\omega y^* - x^*\omega^* y^*)$$

Since x^* and y^* are numbers, the operation ω can be applied to them, and $x^*\omega y^*$ is well defined, except for special cases as when ω represents division and $y^* = 0$. But the expression in the first parenthesees on the right represents propagated error, and that in the second parentheses represents generated error, or round-off. Hence the total error in the result is the sum of the error propagated by the operation and that generated by the operation.

Householder notes that, given two operations ω and ϕ, it may be true that the operations are associative, e.g:

$$(x^*\omega y^*)\phi z^* = x^*\omega(y^*\phi z^*)$$

but if we expand these in terms of the above definitions of propogation and generation error we get two different expressions:

$$\begin{array}{rcccc} (x^*\omega y^*)\phi z^* - (x^*\omega^* y^*)\phi^* z^* & = & [(x^*\omega y^*)\phi z^* & - & (x^*\omega^* y^*)\phi z^*] \\ & + & [(x^*\omega^* y^*)\phi z^* & - & (x^*\omega^* y^*)\phi^* z^*] \end{array}$$

$$\begin{array}{rcccc} x^*\omega(y^*\phi z^*) - x^*\omega^*(y^*\phi^* z^*) & = & [x^*\omega(y^*\phi z^*) & - & x^*\omega(y^*\phi^* z^*)] \\ & + & [x^*\omega(y^*\phi^* z^*) & - & x^*\omega^*(y^*\phi^* z^*)] \end{array}$$

These are not always equal which implies that the strictly machine operations are not necessarily commutative.

Householder distinguishes a third class of error (besides propagation and generative) called residual errors. This occurs because some functions are approximated by infinite series. The finite computation of the series forces the truncation of the remaining terms causing these residual errors.

We will try to perform an analysis of each of the routines in this library for the given inputs to try to see the propagation errors and generation errors they introduce. Every effort will be made to minimize these errors. In particular, we will appeal to the machine generated code to see what approximations actually occur.

Chapter 2

Chapter Overview

Each routine in the Basic Linear Algebra Subroutine set (BLAS) has a prefix where:

- C - Complex
- D - Double Precision
- S - Real
- Z - Complex*16

Routines in level 2 and level 3 of BLAS use the prefix for type:

- GE - general
- GB - general band
- SY - symmetric
- HE - hermitian
- TR - triangular
- SB - symmetric band
- HB - hermetian band
- TB - triangular band
- SP - Sum packed
- HP - hermitian packed
- TP - triangular packed

For level 2 and level 3 BLAS options the options argument is CHARACTER*1 and may be passed as character strings. They mean:

- TRANx
  - **N**o transpose
  - **T**ranspose
  - **C**onjugate transpose (X, X^T, X^H)
- UPLO
  - **U**pper triangular
  - **L**ower triangular
- DIAG
  - **N**on-unit triangular
  - **U**nit triangular
- SIDE
  - **L**eft - A or op(A) on the left
  - **R**ight - A or op(A) on the right

For real matrices, TRANSx=T and TRANSx=C have the same meaning. For Hermitian matrices, TRANSx=T is not allowed. For complex symmetric matrices, TRANSx=H is not allowed.

Chapter 3

Algebra Cover Code

3.1 package BLAS1 BlasLevelOne

⟨*BlasLevelOne.input*⟩≡

```
)set break resume
)sys rm -f BlasLevelOne.output
)spool BlasLevelOne.output
)set message test on
)set message auto off
)clear all

--S 1 of 10
t1:Complex DoubleFloat := complex(1.0,0)
--R
--R
--R   (1)  1.
--R                                                    Type: Complex DoubleFloat
--E 1

--S 2 of 10
dcabs1(t1)
--R
--R
--R   (2)  1.
--R                                                           Type: DoubleFloat
--E 2

--S 3 of 10
t2:Complex DoubleFloat := complex(1.0,1.0)
--R
```

```
--R
--R   (3)  1. + %i
--R                                                    Type: Complex DoubleFloat
--E 3

--S 4 of 10
dcabs1(t2)
--R
--R
--R   (4)  2.
--R                                                            Type: DoubleFloat
--E 4

--S 5 of 10
t3:Complex DoubleFloat := complex(1.0,-1.0)
--R
--R
--R   (5)  1. - %i
--R                                                    Type: Complex DoubleFloat
--E 5

--S 6 of 10
dcabs1(t3)
--R
--R
--R   (6)  2.
--R                                                            Type: DoubleFloat
--E 6

--S 7 of 10
t4:Complex DoubleFloat := complex(-1.0,-1.0)
--R
--R
--R   (7)  - 1. - %i
--R                                                    Type: Complex DoubleFloat
--E 7

--S 8 of 10
dcabs1(t4)
--R
--R
--R   (8)  2.
--R                                                            Type: DoubleFloat
--E 8

--S 9 of 10
```

```
t5:Complex DoubleFloat := complex(-2.0,-2.0)
--R
--R
--R   (9)  - 2. - 2. %i
--R                                                    Type: Complex DoubleFloat
--E 9

--S 10 of 10
dcabs1(t5)
--R
--R
--R   (10)  4.
--R                                                          Type: DoubleFloat
--E 10

a:PRIMARR(DFLOAT):=[ [1.0,2.0,3.0,4,0,5,0,6,0] ]
dasum(3,a,-1) -- 0.0   neg incx
dasum(3,a,0) --  0.0   zero incx
dasum(-1,a,1) -- 0.0   neg elements
dasum(0,a,1) --  0.0   no elements
dasum(1,a,1) --  1.0   1.0
dasum(2,a,1) --  3.0   1.0+2.0
dasum(3,a,1) --  6.0   1.0+2.0+3.0
dasum(4,a,1) -- 10.0   1.0+2.0+3.0+4.0
dasum(5,a,1) -- 15.0   1.0+2.0+3.0+4.0+5.0
dasum(6,a,1) -- 21.0   1.0+2.0+3.0+4.0+5.0+6.0
dasum(7,a,1) -- 21.0   1.0+2.0+3.0+4.0+5.0+6.0
dasum(1,a,2) --  1.0   1.0
dasum(2,a,2) --  4.0   1.0+3.0
dasum(3,a,2) --  9.0   1.0+3.0+5.0
dasum(4,a,2) --  9.0   1.0+3.0+5.0
dasum(1,a,3) --  1.0   1.0
dasum(2,a,3) --  5.0   1.0+4.0
dasum(3,a,3) --  5.0   1.0+4.0
dasum(1,a,4) --  1.0   1.0
dasum(2,a,4) --  6.0   1.0+5.0
dasum(3,a,4) --  6.0   1.0+5.0
dasum(1,a,5) --  1.0   1.0
dasum(2,a,5) --  7.0   1.0+6.0
dasum(3,a,5) --  7.0   1.0+6.0
dasum(1,a,6) --  1.0   1.0
dasum(2,a,6) --  1.0   1.0
dasum(1,a,7) --  1.0   1.0

)spool
)lisp (bye)
```

⟨*BlasLevelOne.help*⟩≡

```
==========================================================================
BlasLevelOne examples
==========================================================================

The dcabs1 routine computes the sum of the absolute value of the
real and imaginary parts of a complex number.

t1:Complex DoubleFloat := complex(1.0,0)
   1.

dcabs1(t1)
   1.

t2:Complex DoubleFloat := complex(1.0,1.0)
   1. + %i

dcabs1(t2)
   2.

t3:Complex DoubleFloat := complex(1.0,-1.0)
   1. - %i

dcabs1(t3)
   2.

t4:Complex DoubleFloat := complex(-1.0,-1.0)
   - 1. - %i

dcabs1(t4)
   2.

t5:Complex DoubleFloat := complex(-2.0,-2.0)
   - 2. - 2. %i

dcabs1(t5)
   4.

See Also:
o )show BlasLevelOne
```

3.1.1 BlasLevelOne (BLAS1)

⟨*package BLAS1 BlasLevelOne*⟩≡

```
)abbrev package BLAS1 BlasLevelOne
++ Author: Timothy Daly
++ Date Created: 2010
++ Date March 24, 2010
++ Description:
++ This package provides an interface to the Blas library (level 1)
BlasLevelOne() : Exports == Implementation where

  SI  ==> SingleInteger
  DF  ==> DoubleFloat
  DX  ==> PrimitiveArray(DoubleFloat)
  CDF ==> Complex DoubleFloat

  Exports == with

    dcabs1: CDF -> DF
      ++ dcabs1(z) computes (+ (abs (realpart z)) (abs (imagpart z)))
      ++
      ++X t1:Complex DoubleFloat := complex(1.0,0)
      ++X dcabs(t1)

    dasum: (SI, DX, SI) -> DF
      ++ dasum(n,array,incx) computes the sum of n elements in array
      ++ using a stride of incx
      ++
      ++X dx:PRIMARR(DFLOAT):=[ [1.0,2.0,3.0,4.0,5.0,6.0] ]
      ++X dasum(6,dx,1)
      ++X dasum(3,dx,2)

    daxpy: (SI, DF, DX, SI, DX,SI) -> DX
      ++ daxpy(n,da,x,incx,y,incy) computes a y = a*x + y
      ++ for each of the chosen elements of the vectors x and y
      ++ and a constant multiplier a
      ++ Note that the vector y is modified with the results.
      ++
      ++X x:PRIMARR(DFLOAT):=[ [1.0,2.0,3.0,4.0,5.0,6.0] ]
      ++X y:PRIMARR(DFLOAT):=[ [1.0,2.0,3.0,4.0,5.0,6.0] ]
      ++X daxpy(6,2.0,x,1,y,1)
      ++X y
      ++X m:PRIMARR(DFLOAT):=[ [1.0,2.0,3.0] ]
      ++X n:PRIMARR(DFLOAT):=[ [1.0,2.0,3.0,4.0,5.0,6.0] ]
      ++X daxpy(3,-2.0,m,1,n,2)
      ++X n
```

```
    dcopy: (SI, DX, SI, DX,SI) -> DX
      ++ dcopy(n,x,incx,y,incy) copies y from x
      ++ for each of the chosen elements of the vectors x and y
      ++ Note that the vector y is modified with the results.
      ++
      ++X x:PRIMARR(DFLOAT):=[ [1.0,2.0,3.0,4.0,5.0,6.0] ]
      ++X y:PRIMARR(DFLOAT):=[ [0.0,0.0,0.0,0.0,0.0,0.0] ]
      ++X dcopy(6,x,1,y,1)
      ++X y
      ++X m:PRIMARR(DFLOAT):=[ [1.0,2.0,3.0] ]
      ++X n:PRIMARR(DFLOAT):=[ [0.0,0.0,0.0,0.0,0.0,0.0] ]
      ++X dcopy(3,m,1,n,2)
      ++X n

  Implementation  == add

      dcabs1(z:CDF):DF ==
        DCABS1(z)$Lisp
      dasum(n:SI,dx:DX,incx:SI):DF ==
        DASUM(n,dx,incx)$Lisp
      daxpy(n:SI,da:DF,dx:DX,incx:SI,dy:DX,incy:SI):DX ==
        DAXPY(n,da,dx,incx,dy,incy)$Lisp
      dcopy(n:SI,dx:DX,incx:SI,dy:DX,incy:SI):DX ==
        DCOPY(n,dx,incx,dy,incy)$Lisp
```

⟨*BLAS1.dotabb*⟩≡

```
"BLAS1" [color="#FF8844",href="bookvol10.5.pdf#nameddest=BLAS1"]
"FIELD" [color="#4488FF",href="bookvol10.2.pdf#nameddest=FIELD"]
"RADCAT" [color="#4488FF",href="bookvol10.2.pdf#nameddest=RADCAT"]
"BLAS1" -> "FIELD"
"BLAS1" -> "RADCAT"
```

3.2 dcabs1 BLAS

⟨*dcabs1.input*⟩≡

```
)set break resume
)sys rm -f dcabs1.output
)spool dcabs1.output
)set message test on
)set message auto off
)clear all

--S 1 of 10
t1:Complex DoubleFloat := complex(1.0,0)
--R
--R
--R   (1)  1.
--R                                                  Type: Complex DoubleFloat
--E 1

--S 2 of 10
dcabs1(t1)
--R
--R
--R   (2)  1.
--R                                                       Type: DoubleFloat
--E 2

--S 3 of 10
t2:Complex DoubleFloat := complex(1.0,1.0)
--R
--R
--R   (3)  1. + %i
--R                                                  Type: Complex DoubleFloat
--E 3

--S 4 of 10
dcabs1(t2)
--R
--R
--R   (4)  2.
--R                                                       Type: DoubleFloat
--E 4

--S 5 of 10
t3:Complex DoubleFloat := complex(1.0,-1.0)
--R
--R
```

```
--R   (5)  1. - %i
--R                                                    Type: Complex DoubleFloat
--E 5

--S 6 of 10
dcabs1(t3)
--R
--R
--R   (6)  2.
--R                                                           Type: DoubleFloat
--E 6

--S 7 of 10
t4:Complex DoubleFloat := complex(-1.0,-1.0)
--R
--R
--R   (7)  - 1. - %i
--R                                                    Type: Complex DoubleFloat
--E 7

--S 8 of 10
dcabs1(t4)
--R
--R
--R   (8)  2.
--R                                                           Type: DoubleFloat
--E 8

--S 9 of 10
t5:Complex DoubleFloat := complex(-2.0,-2.0)
--R
--R
--R   (9)  - 2. - 2. %i
--R                                                    Type: Complex DoubleFloat
--E 9

--S 10 of 10
dcabs1(t5)
--R
--R
--R   (10)  4.
--R                                                           Type: DoubleFloat
--E 10

)spool
)lisp (bye)
```

⟨*dcabs1.help*⟩≡

```
==========================================================================
dcabs1 examples
==========================================================================

The dcabs1 routine computes the sum of the absolute value of the
real and imaginary parts of a complex number.

t1:Complex DoubleFloat := complex(1.0,0)
   1.

dcabs1(t1)
   1.

t2:Complex DoubleFloat := complex(1.0,1.0)
   1. + %i

dcabs1(t2)
   2.

t3:Complex DoubleFloat := complex(1.0,-1.0)
   1. - %i

dcabs1(t3)
   2.

t4:Complex DoubleFloat := complex(-1.0,-1.0)
   - 1. - %i

dcabs1(t4)
   2.

t5:Complex DoubleFloat := complex(-2.0,-2.0)
   - 2. - 2. %i

dcabs1(t5)
   4.

==========================================================================
Man Page Details
==========================================================================

The argument is:
\begin{itemize}
\item z - Complex DoubleFloat
\end{itemize}
```

```
The result is
\begin{itemize}
\item (+ (abs (realpart z)) (abs (imagpart z)))
\end{itemize}

See Also:
o )show BlasLevelOne
o )display operations dcabs1
o )help dcabs1
```

Axiom represents the type Complex(DoubleFloat) as a pair whose car is the real part and whose cdr is the imaginary part. This fact is used in this implementation.

This should really be a macro.

⟨*BLAS dcabs1*⟩≡

```
(defun dcabs1 (z)
 "Complex(DoubleFloat) z is a pair where (realpart . imaginarypart).
  The result is a DoubleFloat (+ (abs (realpart z)) (abs (imagpart z)))"
 (the double-float
  (+
   (the double-float (abs (the double-float (car z))))
   (the double-float (abs (the double-float (cdr z)))))))
```

3.3 lsame BLAS

The `lsame` function returns t if `ca` and
cb represent the same letter regardless of case.

This has been replaced everywhere with common lisp's char-equal function which compares characters ignoring case. The type (simple-array character (*)) has been replaced everywhere which character.

3.4 xerbla BLAS

The `xerbla` routine is an error handler. It is called if an input parameter has an invalid value.

This function has been rewritten everywhere to use the common lisp error function.

Chapter 4

BLAS Level 1

4.1 dasum BLAS

⟨*dasum.input*⟩≡

```
)set break resume
)sys rm -f dasum.output
)spool dasum.output
)set message test on
)set message auto off
)clear all

--S 1 of 28
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0] ]
--R
--R
--R   (1)  [1.,2.,3.,4.,5.,6.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 1

--S 2 of 28
dasum(3,a,-1) -- 0.0   neg incx
--R
--R
--R   (2)  0.
--R                                                                Type: DoubleFloat
--E 2

--S 3 of 28
dasum(3,a,0) --  0.0   zero incx
--R
```

```
--R
--R   (3)  0.
--R                                                    Type: DoubleFloat
--E 3

--S 4 of 28
dasum(-1,a,1) -- 0.0   neg elements
--R
--R
--R   (4)  0.
--R                                                    Type: DoubleFloat
--E 4

--S 5 of 28
dasum(0,a,1) --  0.0   no elements
--R
--R
--R   (5)  0.
--R                                                    Type: DoubleFloat
--E 5

--S 6 of 28
dasum(1,a,1) --  1.0   1.0
--R
--R
--R   (6)  1.
--R                                                    Type: DoubleFloat
--E 6

--S 7 of 28
dasum(2,a,1) --  3.0   1.0+2.0
--R
--R
--R   (7)  3.
--R                                                    Type: DoubleFloat
--E 7

--S 8 of 28
dasum(3,a,1) --  6.0   1.0+2.0+3.0
--R
--R
--R   (8)  6.
--R                                                    Type: DoubleFloat
--E 8

--S 9 of 28
```

```
dasum(4,a,1) -- 10.0   1.0+2.0+3.0+4.0
--R
--R
--R   (9)  10.
--R                                                    Type: DoubleFloat
--E 9

--S 10 of 28
dasum(5,a,1) -- 15.0   1.0+2.0+3.0+4.0+5.0
--R
--R
--R   (10)  15.
--R                                                    Type: DoubleFloat
--E 10

--S 11 of 28
dasum(6,a,1) -- 21.0   1.0+2.0+3.0+4.0+5.0+6.0
--R
--R
--R   (11)  21.
--R                                                    Type: DoubleFloat
--E 11

--S 12 of 28
dasum(7,a,1) -- 21.0   1.0+2.0+3.0+4.0+5.0+6.0
--R
--R
--R   (12)  21.
--R                                                    Type: DoubleFloat
--E 12

--S 13 of 28
dasum(1,a,2) --  1.0   1.0
--R
--R
--R   (13)  1.
--R                                                    Type: DoubleFloat
--E 13

--S 14 of 28
dasum(2,a,2) --  4.0   1.0+3.0
--R
--R
--R   (14)  4.
--R                                                    Type: DoubleFloat
--E 14
```

```
--S 15 of 28
dasum(3,a,2) --  9.0   1.0+3.0+5.0
--R
--R
--R   (15)  9.
--R                                                    Type: DoubleFloat
--E 15

--S 16 of 28
dasum(4,a,2) --  9.0   1.0+3.0+5.0
--R
--R
--R   (16)  9.
--R                                                    Type: DoubleFloat
--E 16

--S 17 of 28
dasum(1,a,3) --  1.0   1.0
--R
--R
--R   (17)  1.
--R                                                    Type: DoubleFloat
--E 17

--S 18 of 28
dasum(2,a,3) --  5.0   1.0+4.0
--R
--R
--R   (18)  5.
--R                                                    Type: DoubleFloat
--E 18

--S 19 of 28
dasum(3,a,3) --  5.0   1.0+4.0
--R
--R
--R   (19)  5.
--R                                                    Type: DoubleFloat
--E 19

--S 20 of 28
dasum(1,a,4) --  1.0   1.0
--R
--R
--R   (20)  1.
```

```
--R                                                    Type: DoubleFloat
--E 20

--S 21 of 28
dasum(2,a,4) --  6.0   1.0+5.0
--R
--R
--R   (21)  6.
--R                                                    Type: DoubleFloat
--E 21

--S 22 of 28
dasum(3,a,4) --  6.0   1.0+5.0
--R
--R
--R   (22)  6.
--R                                                    Type: DoubleFloat
--E 22

--S 23 of 28
dasum(1,a,5) --  1.0   1.0
--R
--R
--R   (23)  1.
--R                                                    Type: DoubleFloat
--E 23

--S 24 of 28
dasum(2,a,5) --  7.0   1.0+6.0
--R
--R
--R   (24)  7.
--R                                                    Type: DoubleFloat
--E 24

--S 25 of 28
dasum(3,a,5) --  7.0   1.0+6.0
--R
--R
--R   (25)  7.
--R                                                    Type: DoubleFloat
--E 25

--S 26 of 28
dasum(1,a,6) --  1.0   1.0
--R
```

```
--R
--R   (26)  1.
--R                                                    Type: DoubleFloat
--E 26

--S 27 of 28
dasum(2,a,6) --  1.0   1.0
--R
--R
--R   (27)  1.
--R                                                    Type: DoubleFloat
--E 27

--S 28 of 28
dasum(1,a,7) --  1.0   1.0
--R
--R
--R   (28)  1.
--R                                                    Type: DoubleFloat
--E 28

)spool
)lisp (bye)
```

⟨*dasum.help*⟩≡

```
  ==========================================================================
  dasum examples
  ==========================================================================

  a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0] ]
      [1.,2.,3.,4.,5.,6.]

  dasum(3,a,-1) -- 0.0   neg incx
      0.

  dasum(3,a,0) --  0.0   zero incx
      0.

  dasum(-1,a,1) -- 0.0   neg elements
      0.

  dasum(0,a,1) --  0.0   no elements
      0.

  dasum(1,a,1) --  1.0   1.0
      1.

  dasum(2,a,1) --  3.0   1.0+2.0
      3.

  dasum(3,a,1) --  6.0   1.0+2.0+3.0
      6.

  dasum(4,a,1) -- 10.0   1.0+2.0+3.0+4.0
     10.

  dasum(5,a,1) -- 15.0   1.0+2.0+3.0+4.0+5.0
     15.

  dasum(6,a,1) -- 21.0   1.0+2.0+3.0+4.0+5.0+6.0
     21.

  dasum(7,a,1) -- 21.0   1.0+2.0+3.0+4.0+5.0+6.0
     21.

  dasum(1,a,2) --  1.0   1.0
      1.

  dasum(2,a,2) --  4.0   1.0+3.0
      4.
```

```
 dasum(3,a,2) --  9.0   1.0+3.0+5.0
     9.

 dasum(4,a,2) --  9.0   1.0+3.0+5.0
     9.

 dasum(1,a,3) --  1.0   1.0
     1.

 dasum(2,a,3) --  5.0   1.0+4.0
     5.

 dasum(3,a,3) --  5.0   1.0+4.0
     5.

 dasum(1,a,4) --  1.0   1.0
     1.

 dasum(2,a,4) --  6.0   1.0+5.0
     6.

 dasum(3,a,4) --  6.0   1.0+5.0
     6.

 dasum(1,a,5) --  1.0   1.0
     1.

 dasum(2,a,5) --  7.0   1.0+6.0
     7.

 dasum(3,a,5) --  7.0   1.0+6.0
     7.

 dasum(1,a,6) --  1.0   1.0
     1.

 dasum(2,a,6) --  1.0   1.0
     1.

 dasum(1,a,7) --  1.0   1.0
     1.

 ========================================================================
 Man Page Details
 ========================================================================
```

```
Computes doublefloat $asum \leftarrow ||re(x)||_1 + ||im(x)||_1$

Arguments are:
\begin{itemize}
\item n - fixnum
\item dx - array doublefloat
\item incx - fixnum
\end{itemize}

Return values are:
\begin{itemize}
\item 1 nil
\item 2 nil
\item 3 nil
\end{itemize}

NAME
       DASUM  -  BLAS level one, sums the absolute values of the elements of a
       double precision vector

SYNOPSIS
       DOUBLE PRECISION FUNCTION DASUM ( n, x, incx )
           INTEGER                      n, incx
           DOUBLE PRECISION             x

AXIOM SIGNATURE:
       SI  ==> SingleInteger
       DF  ==> DoubleFloat
       DX  ==> PrimitiveArray(DoubleFloat)

       dasum: (SI, DX, SI) -> DF

DESCRIPTION
       This routine performs the following vector operation:

                          n
                 DASUM <-- Sum abs(x(i))
                          i=1

ARGUMENTS
       n       INTEGER. (input)
               Number of vector elements to be summed.
               if n <= 0, the result will be 0.0
               if n > length(x) then the whole array is summed.
```

```
        x       DOUBLE PRECISION. (input)
                Array of dimension (n-1) * abs(incx)+ 1.
                Vector that contains elements to be summed.

        incx    INTEGER. (input)
                Increment between elements of x.
                If incx <= 0, the results will be 0.0

RESULT:
        DASUM   DOUBLE PRECISION. (output)
                Sum of the absolute values of the elements of the vector x.
                If n <= 0, DASUM is set to 0.0

NOTES:
        Axiom uses 0-based arrays. Fortran uses 1-based arrays.

        if the index of the array exceeds the length of x
        then no additional elements are added. Thus,
        if x = #(1.0 2.0 3.0 4.0 5.0 6.0)
        then
         (dasum 3 a -1) =  0.0 ; neg incx
         (dasum 3 a 0)  =  0.0 ; zero incx
         (dasum -1 a 1) =  0.0 ; neg elements
         (dasum 0 a 1)  =  0.0 ; no elements
         (dasum 1 a 1)  =  1.0 ; 1.0
         (dasum 2 a 1)  =  3.0 ; 1.0+2.0
         (dasum 3 a 1)  =  6.0 ; 1.0+2.0+3.0
         (dasum 4 a 1)  = 10.0 ; 1.0+2.0+3.0+4.0
         (dasum 5 a 1)  = 15.0 ; 1.0+2.0+3.0+4.0+5.0
         (dasum 6 a 1)  = 21.0 ; 1.0+2.0+3.0+4.0+5.0+6.0
         (dasum 7 a 1)  = 21.0 ; 1.0+2.0+3.0+4.0+5.0+6.0
         (dasum 1 a 2)  =  1.0 ; 1.0
         (dasum 2 a 2)  =  4.0 ; 1.0+3.0
         (dasum 3 a 2)  =  9.0 ; 1.0+3.0+5.0
         (dasum 4 a 2)  =  9.0 ; 1.0+3.0+5.0
         (dasum 1 a 3)  =  1.0 ; 1.0
         (dasum 2 a 3)  =  5.0 ; 1.0+4.0
         (dasum 3 a 3)  =  5.0 ; 1.0+4.0
         (dasum 1 a 4)  =  1.0 ; 1.0
         (dasum 2 a 4)  =  6.0 ; 1.0+5.0
         (dasum 3 a 4)  =  6.0 ; 1.0+5.0
         (dasum 1 a 5)  =  1.0 ; 1.0
         (dasum 2 a 5)  =  7.0 ; 1.0+6.0
         (dasum 3 a 5)  =  7.0 ; 1.0+6.0
         (dasum 1 a 6)  =  1.0 ; 1.0
         (dasum 2 a 6)  =  1.0 ; 1.0
```

```
(dasum 1 a 7)  =  1.0 ; 1.0
```

⟨*BLAS 1 dasum*⟩≡

```
(defun dasum (n dx incx)
 (declare (type (simple-array double-float (*)) dx) (type fixnum incx n))
 (let ((dasum 0.0) (maxlen (length dx)))
  (declare (type (double-float) dasum) (type fixnum maxlen))
  (when (> incx 0)
   (when (> n maxlen) (setq n maxlen))
   (unless (<= n 0)
     (do ((i 0 (1+ i)) (j 0 (+ j incx)))
         ((or (>= i n) (>= j maxlen)))
      (setq dasum
       (the double-float
        (+ (the double-float dasum)
         (the double-float (abs (the double-float (svref dx j))))))))))
    dasum))
```

4.2 daxpy BLAS

⟨*daxpy.input*⟩≡

```
)set break resume
)sys rm -f daxpy.output
)spool daxpy.output
)set message test on
)set message auto off
)clear all

--S 1 of 22
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R   (1)  [1.,2.,3.,4.,5.,6.,7.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 1

--S 2 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R   (2)  [1.,2.,3.,4.,5.,6.,7.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 2

--S 3 of 22
daxpy(3,2.0,a,1,b,1)
--R
--R
--R   (3)  [3.,6.,9.,4.,5.,6.,7.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 3

--S 4 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R   (4)  [1.,2.,3.,4.,5.,6.,7.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 4

--S 5 of 22
daxpy(7,2.0,a,1,b,1)
--R
--R
```

```
--R   (5)  [3.,6.,9.,12.,15.,18.,21.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 5

--S 6 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R   (6)  [1.,2.,3.,4.,5.,6.,7.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 6

--S 7 of 22
daxpy(8,2.0,a,1,b,1)
--R
--R
--R   (7)  [1.,2.,3.,4.,5.,6.,7.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 7

--S 8 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R   (8)  [1.,2.,3.,4.,5.,6.,7.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 8

--S 9 of 22
daxpy(3,2.0,a,3,b,3)
--R
--R
--R   (9)  [3.,2.,3.,12.,5.,6.,21.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 9

--S 10 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R   (10)  [1.,2.,3.,4.,5.,6.,7.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 10

--S 11 of 22
daxpy(4,2.0,a,2,b,2)
```

```
--R
--R
--R   (11)  [3.,2.,9.,4.,15.,6.,21.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 11

--S 12 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R   (12)  [1.,2.,3.,4.,5.,6.,7.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 12

--S 13 of 22
daxpy(5,2.0,a,2,b,2)
--R
--R
--R   (13)  [1.,2.,3.,4.,5.,6.,7.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 13

--S 14 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R   (14)  [1.,2.,3.,4.,5.,6.,7.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 14

--S 15 of 22
daxpy(3,2.0,a,2,b,2)
--R
--R
--R   (15)  [3.,2.,9.,4.,15.,6.,7.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 15

--S 16 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R   (16)  [1.,2.,3.,4.,5.,6.,7.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 16
```

```
--S 17 of 22
daxpy(3,-2.0,a,2,b,2)
--R
--R
--R   (17)  [- 1.,2.,- 3.,4.,- 5.,6.,7.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 17

--S 18 of 22
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0] ]
--R
--R
--R   (18)  [1.,2.,3.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 18

--S 19 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
--R
--R
--R   (19)  [1.,2.,3.,4.,5.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 19

--S 20 of 22
daxpy(3,-2.0,a,1,b,2)
--R
--R
--R   (20)  [- 1.,2.,- 1.,4.,- 1.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 20

--S 21 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R   (21)  [1.,2.,3.,4.,5.,6.,7.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 21

--S 22 of 22
daxpy(3,0.0,a,1,b,2)
--R
--R
--R   (22)  [1.,2.,3.,4.,5.,6.,7.]
--R                                              Type: PrimitiveArray DoubleFloat
```

```
--E 22

)spool
)lisp (bye)
```

⟨*daxpy.help*⟩≡

```
  ========================================================================
  daxpy examples
  ========================================================================

  For each of the following examples we assume that we have preset
  the variables to the following values. Note that the daxpy function
  will modify the second array. Each example assumes we have reset
  the variables to these values.

  a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
  b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]

  then we compute the sum of the first 3 elements of each vector
  and we show the steps of the computation with trailing comments.
  The comments show which elements are changed and what the computation is.

  daxpy(3,2.0,a,1,b,1) ==> [3.,6.,9.,4.,5.,6.,7.]
    dy(0)[3.0] = dy(0)[1.0] + ( da[2.0] * dx(0)[1.0] )
    dy(1)[6.0] = dy(1)[2.0] + ( da[2.0] * dx(1)[2.0] )
    dy(2)[9.0] = dy(2)[3.0] + ( da[2.0] * dx(2)[3.0] )

  or we compute the first 7 elements of each vector

  daxpy(7,2.0,a,1,b,1) ==> [3.,6.,9.,12.,15.,18.,21.]
    dy(0) [3.0] = dy(0)[1.0] + ( da[2.0] * dx(0)[1.0] )
    dy(1) [6.0] = dy(1)[2.0] + ( da[2.0] * dx(1)[2.0] )
    dy(2) [9.0] = dy(2)[3.0] + ( da[2.0] * dx(2)[3.0] )
    dy(3)[12.0] = dy(3)[4.0] + ( da[2.0] * dx(3)[4.0] )
    dy(4)[15.0] = dy(4)[5.0] + ( da[2.0] * dx(4)[5.0] )
    dy(5)[18.0] = dy(5)[6.0] + ( da[2.0] * dx(5)[6.0] )
    dy(6)[21.0] = dy(6)[7.0] + ( da[2.0] * dx(6)[7.0] )

  or we compute the first 8 elements, which fails due to the fact that
  the vectors are not long enough.

  daxpy(8,2.0,a,1,b,1) ==> [1.,2.,3.,4.,5.,6.,7.]

  or we compute the 3 elements, taking every 3rd element, giving the
  index of 0, 3, 6

  daxpy(3,2.0,a,3,b,3) ==> [3.,2.,3.,12.,5.,6.,21.]
    dy(0) [3.0] = dy(0)[1.0] + ( da[2.0] * dx(0)[1.0] )
    dy(3)[12.0] = dy(3)[4.0] + ( da[2.0] * dx(3)[4.0] )
    dy(6)[21.0] = dy(6)[7.0] + ( da[2.0] * dx(6)[7.0] )
```

```
or we compute 4 elements, taking every 2nd element, giving the
index of 0, 2, 4, 6

daxpy(4,2.0,a,2,b,2) ==> [3.,2.,9.,4.,15.,6.,21.]
  dy(0) [3.0] = dy(0)[1.0] + ( da[2.0] * dx(0)[1.0] )
  dy(2) [9.0] = dy(2)[3.0] + ( da[2.0] * dx(2)[3.0] )
  dy(4)[15.0] = dy(4)[5.0] + ( da[2.0] * dx(4)[5.0] )
  dy(6)[21.0] = dy(6)[7.0] + ( da[2.0] * dx(6)[7.0] )

or we compute 5 elements, taking every 2nd element, which fails
due to vector length

daxpy(5,2.0,a,2,b,2) ==> [1.,2.,3.,4.,5.,6.,7.]

or we compute 3 elements, taking every 2nd value, giving the
index of 0, 2, 4

daxpy(3,2.0,a,2,b,2) ==> [3.,2.,9.,4.,15.,6.,7.]
  dy(0) [3.0] = dy(0)[1.0] + ( da[2.0] * dx(0)[1.0] )
  dy(2) [9.0] = dy(2)[3.0] + ( da[2.0] * dx(2)[3.0] )
  dy(4)[15.0] = dy(4)[5.0] + ( da[2.0] * dx(4)[5.0] )

or we compute 3 elements, taking every 2nd value, giving the
index of 0, 2, 4 but with a negative multiplier

daxpy(3,-2.0,a,2,b,2) ==> [- 1.,2.,- 3.,4.,- 5.,6.,7.]
  dy(0)[-1.0] = dy(0)[1.0] + ( da[-2.0] * dx(0)[1.0] )
  dy(2)[-3.0] = dy(2)[3.0] + ( da[-2.0] * dx(2)[3.0] )
  dy(4)[-5.0] = dy(4)[5.0] + ( da[-2.0] * dx(4)[5.0] )

or we change the lengths of the input vectors, making them unequal.
So for the next two examples we assume the arrays look like:

a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0] ]
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]

We compute 3 elements, with a negative multiplier, and different increments
using an index for the 'a' array having values 0, 1, 2
using an index for the 'b' array having values 0, 2, 3

daxpy(3,-2.0,a,1,b,2) ==> [- 1.,2.,- 1.,4.,- 1.]
  dy(0)[-1.0] = dy(0)[1.0] + ( da[-2.0] * dx(0)[1.0] )
  dy(2)[-1.0] = dy(2)[3.0] + ( da[-2.0] * dx(1)[2.0] )
  dy(4)[-1.0] = dy(4)[5.0] + ( da[-2.0] * dx(2)[3.0] )

or we compute 3 elements with a multiplier of 0.0 which fails
```

```
daxpy(3,0.0,a,1,b,2) ==> [1.,2.,3.,4.,5.,6.,7.]

=========================================================================
Man Page Details
=========================================================================

NAME
       DAXPY - BLAS level one axpy subroutine

SYNOPSIS
       SUBROUTINE DAXPY    ( n, alpha, x, incx, y, incy )

           INTEGER           n, incx, incy

           DOUBLE PRECISION  alpha, x, y


DESCRIPTION
       DAXPY  adds  a  scalar multiple of a double precision vector to another
       double precision vector.

       DAXPY computes a constant alpha times a vector x plus a vector y.   The
       result overwrites the initial values of vector y.

       This routine performs the following vector operation:

            y  <--  alpha*x + y

       incx and incy specify the increment between two consecutive
       elements of respectively vector x and y.

ARGUMENTS
       n       INTEGER. (input)
               Number  of  elements in the vectors.  If n <= 0, these routines
               return without any computation.

       alpha   DOUBLE PRECISION. (input)
               If alpha = 0 this routine returns without any computation.

       x       DOUBLE PRECISION, (input)
               Array of dimension (n-1) * |incx| + 1.  Contains the vector  to
               be scaled before summation.

       incx    INTEGER. (input)
               Increment between elements of x.
```

```
                If incx = 0, the results will be unpredictable.

        y       DOUBLE PRECISION, (input and output)
                array of dimension (n-1) * |incy| + 1.
                Before calling the routine, y contains the vector to be summed.
                After the routine ends, y contains the result of the summation.

        incy    INTEGER. (input)
                Increment between elements of y.
                If incy = 0, the results will be unpredictable.

RETURN VALUES
       When  n <= 0, double precision alpha = 0., this routine returns immedi-
       ately with no change in its arguments.
```

⟨*BLAS 1 daxpy*⟩≡

```
(defun daxpy (n da dx incx dy incy)
 (declare (type (simple-array double-float) dx dy)
          (type double-float da) (type fixnum incy incx n))
  (let ((maxx (length dx)) (maxy (length dy)))
   (declare (type fixnum maxx maxy))
   (when (and (> n 0) (/= da 0.0)
              (> incx 0) (< (* (1- n) incx) maxx)
              (> incy 0) (< (* (1- n) incy) maxy))
    (if (and (= incx 1) (= incy 1))
     ; unit increments
     (dotimes (i n)
         ; (format t "dy(~s)[~s] = dy(~s)[~s] + ( da[~s] * dx(~s)[~s] )~%"
         ;  i (+ (svref dy i) (* da (svref dx i)))
         ;  i (svref dy i)
         ;  da i (svref dx i))
       (setf (the double-float (svref dy i))
        (+ (the double-float (svref dy i))
           (* (the double-float da)
              (the double-float (svref dx i))))))
     ; non-unit increments
     (let ((ix 0) (iy 0))
      (declare (type fixnum ix iy))
      (when (< incx 0) (setq ix (* (1+ (- n)) incx)))
      (when (< incy 0) (setq ix (* (1+ (- n)) incy)))
      (dotimes (i n)
          ; (format t "dy(~s)[~s] = dy(~s)[~s] + ( da[~s] * dx(~s)[~s] )~%"
          ;  iy (+ (svref dy iy) (* da (svref dx ix)))
          ;  iy (svref dy iy)
          ;  da ix (svref dx ix))
        (setf (the double-float (svref dy iy))
         (+ (the double-float (svref dy iy))
            (* (the double-float da)
               (the double-float (svref dx ix)))))
        (setq ix (+ ix incx))
        (setq iy (+ iy incy)))))))
    dy)
```

4.3 dcopy BLAS

⟨*dcopy.input*⟩≡

```
)set break resume
)sys rm -f dcopy.output
)spool dcopy.output
)set message test on
)set message auto off
)clear all

--S 1 of 23
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R   (1)  [1.,2.,3.,4.,5.,6.,7.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 1

--S 2 of 23
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R   (2)  [0.,0.,0.,0.,0.,0.,0.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 2

--S 3 of 23
dcopy(3,a,1,b,1)
--R
--R
--R   (3)  [1.,2.,3.,0.,0.,0.,0.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 3

--S 4 of 23
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R   (4)  [0.,0.,0.,0.,0.,0.,0.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 4

--S 5 of 23
dcopy(7,a,1,b,1)
--R
--R
```

```
--R   (5)  [1.,2.,3.,4.,5.,6.,7.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 5

--S 6 of 23
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R   (6)  [0.,0.,0.,0.,0.,0.,0.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 6

--S 7 of 23
dcopy(8,a,1,b,1)
--R
--R
--R   (7)  [0.,0.,0.,0.,0.,0.,0.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 7

--S 8 of 23
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R   (8)  [0.,0.,0.,0.,0.,0.,0.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 8

--S 9 of 23
dcopy(3,a,3,b,3)
--R
--R
--R   (9)  [1.,0.,0.,4.,0.,0.,7.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 9

--S 10 of 23
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R   (10)  [0.,0.,0.,0.,0.,0.,0.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 10

--S 11 of 23
dcopy(4,a,2,b,2)
```

```
--R
--R
--R   (11)  [1.,0.,3.,0.,5.,0.,7.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 11

--S 12 of 23
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R   (12)  [0.,0.,0.,0.,0.,0.,0.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 12

--S 13 of 23
dcopy(5,a,2,b,2)
--R
--R
--R   (13)  [0.,0.,0.,0.,0.,0.,0.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 13

--S 14 of 23
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R   (14)  [0.,0.,0.,0.,0.,0.,0.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 14

--S 15 of 23
dcopy(3,a,2,b,2)
--R
--R
--R   (15)  [1.,0.,3.,0.,5.,0.,0.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 15

--S 16 of 23
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0] ]
--R
--R
--R   (16)  [1.,2.,3.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 16
```

```
--S 17 of 23
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
--R
--R
--R   (17)  [1.,2.,3.,4.,5.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 17

--S 18 of 23
dcopy(3,a,1,b,1)
--R
--R
--R   (18)  [1.,2.,3.,4.,5.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 18

--S 19 of 23
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
--R
--R
--R   (19)  [1.,2.,3.,4.,5.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 19

--S 20 of 23
dcopy(3,a,1,b,2)
--R
--R
--R   (20)  [1.,2.,2.,4.,3.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 20

--S 21 of 23
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
--R
--R
--R   (21)  [1.,2.,3.,4.,5.]
--R                                              Type: PrimitiveArray DoubleFloat
--E 21

--S 22 of 23
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0] ]
--R
--R
--R   (22)  [1.,2.,3.]
--R                                              Type: PrimitiveArray DoubleFloat
```

```
--E 22

--S 23 of 23
dcopy(5,a,1,b,1)
--R
--R
--R   (23)  [1.,2.,3.]
--R                                                Type: PrimitiveArray DoubleFloat
--E 23

)spool
)lisp (bye)
```

⟨*dcopy.help*⟩≡

```
==========================================================================
dcopy examples
==========================================================================

Assume we have two arrays which we initialize to these values:
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]

b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]

Note that after each call to dcopy the b array is modified.
The changed value is shown. We reset it after each bcopy but we
do not show that here.

dcopy(3,a,1,b,1) ==> [1.,2.,3.,0.,0.,0.,0.]

dcopy(7,a,1,b,1) ==> [1.,2.,3.,4.,5.,6.,7.]

dcopy(8,a,1,b,1) ==> [0.,0.,0.,0.,0.,0.,0.]

dcopy(3,a,3,b,3) ==> [1.,0.,0.,4.,0.,0.,7.]

dcopy(4,a,2,b,2) ==> [1.,0.,3.,0.,5.,0.,7.]

dcopy(5,a,2,b,2) ==> [0.,0.,0.,0.,0.,0.,0.]

dcopy(3,a,2,b,2) ==> [1.,0.,3.,0.,5.,0.,0.]

The arrays can be of different lengths:

a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0] ]

b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]

dcopy(3,a,1,b,1) ==>  [1.,2.,3.,4.,5.]

dcopy(3,a,1,b,2) ==>  [1.,2.,2.,4.,3.]

a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]

b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0] ]

dcopy(5,a,1,b,1) ==> [1.,2.,3.]

==========================================================================
Man Page Details
```

```
=====================================================================

NAME
       DCOPY  -  BLAS level one, copies a double precision vector into another
       double precision vector

SYNOPSIS
       SUBROUTINE DCOPY    ( n, x, incx, y, incy )

           INTEGER           n, incx, incy

           DOUBLE PRECISION  x, y


DESCRIPTION
       DCOPY copies a double precision vector into  another  double  precision
       vector.  DCOPY copies a vector x, whose length is n to a vector y. incx
       and incy specify the increment  between  two  consecutive  elements  of
       respectively vector x and y.

       This routine performs the following vector operation:

            y  <--  x

       where x and y are double precision vectors.

ARGUMENTS
       n       INTEGER. (input)
               Number of vector elements to be copied.
               If n <= 0, this routine returns without computation.

       x       DOUBLE PRECISION, (input)
               Vector from which to copy.

       incx    INTEGER. (input)
               Increment between elements of x.
               If incx = 0, the y vector is unchanged

       y       DOUBLE PRECISION, (output)
               array of dimension (n-1) * |incy| + 1, result vector.

       incy    INTEGER. (input)
               Increment between elements of y.
               If incy = 0, the y vector is unchanged
```

⟨*BLAS 1 dcopy*⟩≡
```
(defun dcopy (n dx incx dy incy)
 (declare (type (simple-array double-float) dy dx)
          (type fixnum incy incx n))
  (let ((maxx (length dx)) (maxy (length dy)))
   (declare (type fixnum maxx maxy))
   (when (and (> n 0)
              (> incx 0) (< (* (1- n) incx) maxx)
              (> incy 0) (< (* (1- n) incy) maxy))
    (if (and (= incx 1) (= incy 1))
     ; unit increments
     (dotimes (i n)
       (setf (the double-float (svref dy i)) (the double-float (svref dx i))))
     ; non-unit increments
     (let ((ix 0) (iy 0))
      (declare (type fixnum ix iy))
      (when (< incx 0) (setq ix (* (1+ (- n)) incx)))
      (when (< incy 0) (setq ix (* (1+ (- n)) incy)))
      (dotimes (i n)
       (setf (the double-float (svref dy iy)) (the double-float (svref dx ix)))
       (setq ix (+ ix incx))
       (setq iy (+ iy incy)))))))
   dy)
```

4.4 ddot BLAS

⟨*ddot.input*⟩≡
```
)set break resume
)sys rm -f ddot.output
)spool ddot.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*ddot.help*⟩≡

```
======================================================================
ddot examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DDOT  -  BLAS  level one, computes a dot product (inner product) of two
       double precision vectors

SYNOPSIS
       DOUBLE PRECISION FUNCTION DDOT ( n, x, incx, y, incy )

           INTEGER                     n, incx, incy

           DOUBLE PRECISION            x, y


DESCRIPTION
       DDOT computes a dot product of two double precision vectors  (l  double
       precision inner product).
         2

       This routine performs the following vector operation:

                                                    n
               DDOT  <--  (transpose of x) * y  =  Sum  x(i)*y(i)
                                                   i=1
       where x and y are double precision vectors.

       If n <= 0, DDOT is set to 0.

ARGUMENTS
       n       INTEGER. (input)
               Number of elements in each vector.

       x       DOUBLE PRECISION. (input)
               Array of dimension (n-1) * |incx| + 1.
               Array x contains the first vector operand.

       incx    INTEGER. (input)
               Increment between elements of x.
               If incx = 0, the results will be unpredictable.
```

```
        y       DOUBLE PRECISION, (input)
                Array of dimension (n-1) * |incy| + 1.
                Array y contains the second vector operand.

        incy    INTEGER. (input)
                Increment between elements of y.  If incy = 0, the results will
                be unpredictable.

RETURN VALUES
       DDOT    DOUBLE PRECISION. Result (dot product). (output)

NOTES
       When  working  backward  (incx < 0 or incy < 0), each routine starts at
       the end of the vector and moves backward, as follows:

            x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

            y(1-incy * (n-1)), y(1-incy * (n-2)), ..., y(1)
```

⟨*BLAS 1 ddot*⟩≡

```
(defun ddot (n dx incx dy incy)
  (declare (type (simple-array double-float (*)) dy dx)
           (type fixnum incy incx n))
  (f2cl-lib:with-multi-array-data
      ((dx double-float dx-%data% dx-%offset%)
       (dy double-float dy-%data% dy-%offset%))
    (prog ((i 0) (ix 0) (iy 0) (m 0) (mp1 0) (dtemp 0.0) (ddot 0.0))
      (declare (type (double-float) ddot dtemp)
               (type fixnum mp1 m iy ix i))
      (setf ddot 0.0)
      (setf dtemp 0.0)
      (if (<= n 0) (go end_label))
      (if (and (= incx 1) (= incy 1)) (go label20))
      (setf ix 1)
      (setf iy 1)
      (if (< incx 0)
          (setf ix
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incx)
                   1)))
      (if (< incy 0)
          (setf iy
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incy)
                   1)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf dtemp
                  (+ dtemp
                     (* (f2cl-lib:fref dx-%data% (ix) ((1 *)) dx-%offset%)
                        (f2cl-lib:fref dy-%data% (iy) ((1 *)) dy-%offset%))))
          (setf ix (f2cl-lib:int-add ix incx))
          (setf iy (f2cl-lib:int-add iy incy))))
      (setf ddot dtemp)
      (go end_label)
 label20
      (setf m (mod n 5))
      (if (= m 0) (go label40))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i m) nil)
        (tagbody
          (setf dtemp
                  (+ dtemp
                     (* (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%)
```

```
                        (f2cl-lib:fref dy-%data% (i) ((1 *)) dy-%offset%))))))
      (if (< n 5) (go label60))
  label40
      (setf mp1 (f2cl-lib:int-add m 1))
      (f2cl-lib:fdo (i mp1 (f2cl-lib:int-add i 5))
                    ((> i n) nil)
        (tagbody
          (setf dtemp
                  (+ dtemp
                     (* (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%)
                        (f2cl-lib:fref dy-%data% (i) ((1 *)) dy-%offset%))
                     (*
                      (f2cl-lib:fref dx-%data%
                                     ((f2cl-lib:int-add i 1))
                                     ((1 *))
                                     dx-%offset%)
                      (f2cl-lib:fref dy-%data%
                                     ((f2cl-lib:int-add i 1))
                                     ((1 *))
                                     dy-%offset%))
                     (*
                      (f2cl-lib:fref dx-%data%
                                     ((f2cl-lib:int-add i 2))
                                     ((1 *))
                                     dx-%offset%)
                      (f2cl-lib:fref dy-%data%
                                     ((f2cl-lib:int-add i 2))
                                     ((1 *))
                                     dy-%offset%))
                     (*
                      (f2cl-lib:fref dx-%data%
                                     ((f2cl-lib:int-add i 3))
                                     ((1 *))
                                     dx-%offset%)
                      (f2cl-lib:fref dy-%data%
                                     ((f2cl-lib:int-add i 3))
                                     ((1 *))
                                     dy-%offset%))
                     (*
                      (f2cl-lib:fref dx-%data%
                                     ((f2cl-lib:int-add i 4))
                                     ((1 *))
                                     dx-%offset%)
                      (f2cl-lib:fref dy-%data%
                                     ((f2cl-lib:int-add i 4))
                                     ((1 *))
```

```
                                        dy-%offset%))))))
   label60
        (setf ddot dtemp)
   end_label
        (return (values ddot nil nil nil nil nil)))))
```

4.5 dnrm2 BLAS

⟨*dnrm2.input*⟩≡

```
  )set break resume
  )sys rm -f dnrm2.output
  )spool dnrm2.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dnrm2.help*⟩≡

```
======================================================================
dnrm2 examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DNRM2 - BLAS level one, computes the Euclidean norm of a vector

SYNOPSIS
       DOUBLE PRECISION FUNCTION DNRM2 ( n, x, incx )

           INTEGER                      n, incx

           DOUBLE PRECISION             x


DESCRIPTION
       DNRM2  computes the Euclidean (L2) norm of a double precision real vec-
       tor, as follows:

               DNRM2 <-- ||x||
                              2

       where x is a double precision real vector.

ARGUMENTS
       n       INTEGER. (input)
               Number of elements in the operand vector.

       x       DOUBLE PRECISION. (input)
               Array of dimension (n-1) * |incx| + 1.
               Array x contains the operand vector.

       incx    INTEGER. (input)
               Increment between elements of x.
               If incx = 0, the results will be unpredictable.

RETURN VALUES
       DNRM2   DOUBLE PRECISION. Result (Euclidean norm). (output)
               If n <= 0, DNRM2 is set to 0d0.

NOTES
```

When working backward (incx < 0), each routine starts at the end of the vector and moves backward, as follows:

```
x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)
```

⟨*BLAS 1 dnrm2*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dnrm2 (n x incx)
    (declare (type (simple-array double-float (*)) x)
             (type fixnum incx n))
    (f2cl-lib:with-multi-array-data
        ((x double-float x-%data% x-%offset%))
      (prog ((absxi 0.0) (norm 0.0) (scale 0.0) (ssq 0.0) (ix 0) (dnrm2 0.0))
        (declare (type fixnum ix)
                 (type (double-float) absxi norm scale ssq dnrm2))
        (cond
          ((or (< n 1) (< incx 1))
           (setf norm zero))
          ((= n 1)
           (setf norm (abs (f2cl-lib:fref x-%data% (1) ((1 *)) x-%offset%))))
          (t
           (setf scale zero)
           (setf ssq one)
           (f2cl-lib:fdo (ix 1 (f2cl-lib:int-add ix incx))
                         ((> ix
                             (f2cl-lib:int-add 1
                                               (f2cl-lib:int-mul
                                                (f2cl-lib:int-add n
                                                                  (f2cl-lib:int-sub
                                                                   1))
                                                incx)))
                          nil)
             (tagbody
               (cond
                 ((/= (f2cl-lib:fref x (ix) ((1 *))) zero)
                  (setf absxi
                          (abs
                           (f2cl-lib:fref x-%data% (ix) ((1 *)) x-%offset%)))
                  (cond
                    ((< scale absxi)
                     (setf ssq (+ one (* ssq (expt (/ scale absxi) 2))))
                     (setf scale absxi))
                    (t
                     (setf ssq (+ ssq (expt (/ absxi scale) 2)))))))))
             (setf norm (* scale (f2cl-lib:fsqrt ssq)))))
        (setf dnrm2 norm)
  end_label
        (return (values dnrm2 nil nil nil))))))
```

4.6 drotg BLAS

⟨*drotg.input*⟩≡

```
)set break resume
)sys rm -f drotg.output
)spool drotg.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*drotg.help*⟩≡

```
==========================================================================
drotg examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

NAME
       DGROTG - BLAS level one rotation subroutines

SYNOPSIS
       SUBROUTINE DROTG    ( a, b, c, s )

           DOUBLE PRECISION  a, b, c, s


DESCRIPTION
       DROTG  computes  the  elements  of  a Givens plane rotation matrix such
       that:

             _      _     _   _    _   _
             | c  s |     | a |    | r |
             |-s  c | *   | b | =  | 0 |
             -      -     -   -    -   -

       where  r = +- sqrt ( a**2 + b**2 )  and c**2 + s**2 =1.

       The Givens plane rotation can be used to introduce zero elements into
       a matrix selectively.



ARGUMENTS
       a       (input and output) DOUBLE PRECISION

               First vector component.  On input, the first component  of  the
               vector to be rotated.  On output, a is overwritten by by r, the
               first component of the vector in the rotated coordinate  system
               where:

               r = sign(sqrt(a**2 + b**2),a), if |a| > |b|

               r = sign(sqrt(a**2 + b**2),b), if |a| <= |b|
```

```
b       (input and output) DOUBLE PRECISION
        Second vector component.

        On input, the second component of the vector to be rotated.  On
        output, b contains z, where:

            z=s    if  |a| >  |b|
            z=1/c  if  |a| <= |b|  and  c != 0  and r != 0
            z=1    if  |a| <= |b|  and  c  = 0  and r != 0
            z=0    if  r = 0

c       (output) DOUBLE PRECISION
        Cosine of the angle of rotation:

             c = a/r  if  r != 0
             c = 1    if  r  = 0

s       (output) DOUBLE PRECISION
        Sine of the angle of rotation:

             s = b/r  if  r != 0
             s = 0    if  r  = 0


NOTE
        The value of z, returned in b by DROTG, gives a compact  representation
        of  the rotation matrix, which can be used later to reconstruct c and s
        as in the following example:

             IF (B .EQ. 1. ) THEN
                 C = 0.
                 S = 1.
             ELSEIF( ABS( B) .LT. 1) THEN
                 C = SQRT( 1. - B * B)
                 S = B
             ELSE
                 C = 1. / B
                 S = SQRT( 1 - C * C)
             ENDIF
```

Double precision. Computes plane rotation. Arguments are:

- da - double-float
- db - double-float
- c - double-float
- s - double-float

Returns multiple values where:

- 1 da - double-float
- 2 db - double-float
- 3 c - double-float
- 4 s - double-float

⟨*BLAS 1 drotg*⟩≡

```
(defun drotg (da db c s)
 (declare (type (double-float) s c db da))
 (prog ((roe 0.0) (scale 0.0) (r 0.0) (z 0.0))
  (declare (type (double-float) z r scale roe))
   (setf roe db)
   (when (> (the double-float (abs da)) (the double-float (abs db)))
    (setf roe da))
   (setf scale (+ (the double-float (abs da)) (the double-float (abs db))))
   (if (/= scale 0.0) (go label10))
   (setf c 1.0)
   (setf s 0.0)
   (setf r 0.0)
   (setf z 0.0)
   (go label20)
 label10
   (setf r
    (* scale (f2cl-lib:dsqrt (+ (expt (/ da scale) 2) (expt (/ db scale) 2)))))
   (setf r (* (f2cl-lib:dsign 1.0 roe) r))
   (setf c (/ da r))
   (setf s (/ db r))
   (setf z 1.0)
   (when (> (the double-float (abs da)) (the double-float (abs db)))
    (setf z s))
   (if (and (>= (the double-float (abs db)) (the double-float (abs da)))
            (/= c 0.0))
    (setf z (/ 1.0 c)))
 label20
   (setf da r)
```

```
    (setf db z)
    (return (values da db c s))))
```

4.7 drot BLAS

⟨*drot.input*⟩≡

```
)set break resume
)sys rm -f drot.output
)spool drot.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*drot.help*⟩≡

```
  ======================================================================
  drot examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DROT - BLAS level one, plane rotation subroutines

  SYNOPSIS
         SUBROUTINE DROT    ( n, x, incx, y, incy, c, s )

             INTEGER          n, incx, incy

             DOUBLE PRECISION x, y, c, s


  DESCRIPTION
         DROT   applies  a  plane  rotation matrix to a real sequence of ordered
         pairs:

              (x , y ), for all i = 1, 2, ..., n.
                i   i

  ARGUMENTS
         n       INTEGER. (input)
                 Number of ordered pairs (planar points in DROT) to be  rotated.
                 If n <= 0, this routine returns without computation.

         x       DOUBLE PRECISION, (input and output)
                 Array  of dimension (n-1) * |incx| + 1.  On input, array x con-
                 tains the x-coordinate of each planar point to be rotated.   On
                 output,  array x contains the x-coordinate of each rotated pla-
                 nar point.

         incx    INTEGER. (input)
                 Increment between elements of x.  If incx = 0, the results will
                 be unpredictable.

         y       DOUBLE PRECISION, (input and output)
                 array of dimension (n-1) * |incy| + 1.
                 On  input,  array  y  contains  the y-coordinate of each planar
                 point to be rotated.  On output, array y contains the y-coordi-
```

nate of each rotated planar point.

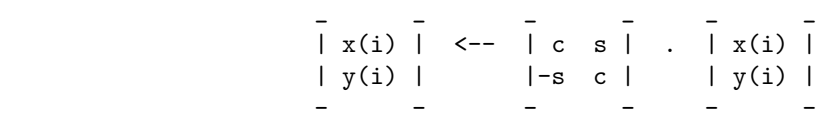
incy INTEGER. (input)
Increment between elements of y. If incy = 0, the results will be unpredictable.

c DOUBLE PRECISION, Cosine of the angle of rotation. (input)

s DOUBLE PRECISION, Sine of the angle of rotation. (input)

NOTES

This routine applies the following plane rotation to each pair of elements (x , y):



for i = 1,...,n

If coefficients c and s satisfy $c^2 + s^2 = 1.0$, the rotation matrix is orthogonal, and the transformation is called a Givens plane rotation. If c = 1 and s = 0, DROT returns without modifying any input parameters.

To calculate the Givens coefficients c and s from a two-element vector to determine the angle of rotation, use SROTG(3S).

When working backward (incx < 0 or incy < 0), each routine starts at the end of the vector and moves backward, as follows:

x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

y(1-incy * (n-1)), y(1-incy * (n-2)), ..., y(1)

⟨*BLAS 1 drot*⟩≡

```
(defun drot (n dx incx dy incy c s)
  (declare (type (double-float) s c)
           (type (simple-array double-float (*)) dy dx)
           (type fixnum incy incx n))
  (f2cl-lib:with-multi-array-data
      ((dx double-float dx-%data% dx-%offset%)
       (dy double-float dy-%data% dy-%offset%))
    (prog ((i 0) (ix 0) (iy 0) (dtemp 0.0))
      (declare (type (double-float) dtemp) (type fixnum iy ix i))
      (if (<= n 0) (go end_label))
      (if (and (= incx 1) (= incy 1)) (go label20))
      (setf ix 1)
      (setf iy 1)
      (if (< incx 0)
          (setf ix
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incx)
                   1)))
      (if (< incy 0)
          (setf iy
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incy)
                   1)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf dtemp
                  (+ (* c (f2cl-lib:fref dx-%data% (ix) ((1 *)) dx-%offset%))
                     (* s (f2cl-lib:fref dy-%data% (iy) ((1 *)) dy-%offset%))))
          (setf (f2cl-lib:fref dy-%data% (iy) ((1 *)) dy-%offset%)
                  (- (* c (f2cl-lib:fref dy-%data% (iy) ((1 *)) dy-%offset%))
                     (* s (f2cl-lib:fref dx-%data% (ix) ((1 *)) dx-%offset%))))
          (setf (f2cl-lib:fref dx-%data% (ix) ((1 *)) dx-%offset%) dtemp)
          (setf ix (f2cl-lib:int-add ix incx))
          (setf iy (f2cl-lib:int-add iy incy))))
      (go end_label)
  label20
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf dtemp
                  (+ (* c (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%))
                     (* s (f2cl-lib:fref dy-%data% (i) ((1 *)) dy-%offset%))))
          (setf (f2cl-lib:fref dy-%data% (i) ((1 *)) dy-%offset%)
                  (- (* c (f2cl-lib:fref dy-%data% (i) ((1 *)) dy-%offset%))
```

```
                        (* s (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%))))
            (setf (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%) dtemp)))
   end_label
        (return (values nil nil nil nil nil nil nil)))))
```

4.8 dscal BLAS

⟨*dscal.input*⟩≡

```
  )set break resume
  )sys rm -f dscal.output
  )spool dscal.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dscal.help*⟩≡

```
  ======================================================================
  dscal examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DSCAL - BLAS level one, scales a double precision vector

  SYNOPSIS
         SUBROUTINE DSCAL    ( n, alpha, x, incx )

             INTEGER           n, incx

             DOUBLE PRECISION  alpha, x


  DESCRIPTION
         DSCAL  scales a double precision vector with a double precision scalar.
         DSCAL scales the vector x of length n and increment incx  by  the  con-
         stant a.

         This routine performs the following vector operation:

                   x <-- alpha x

         where alpha is a double precisoin scalar, and x is a double precision
         vector.

  ARGUMENTS
         n       INTEGER. (input)
                 Number of elements in the vector.
                 If n <= 0, this routine returns without computation.

         alpha   DOUBLE PRECISION scalar alpha. (input)

         x       DOUBLE PRECISION, (input and output)
                 Array of dimension (n-1) * |incx| + 1. Vector to be scaled.

         incx    INTEGER. (input)
                 Increment between elements of x.
                 If incx = 0, the results will be unpredictable.
```

```
NOTES
       When working backward (incx < 0 or incy < 0), each  routine  starts  at
       the end of the vector and moves backward, as follows:

            x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)
```

⟨*BLAS 1 dscal*⟩≡

```
(defun dscal (n da dx incx)
  (declare (type (simple-array double-float (*)) dx)
           (type (double-float) da)
           (type fixnum incx n))
  (f2cl-lib:with-multi-array-data
      ((dx double-float dx-%data% dx-%offset%))
    (prog ((i 0) (m 0) (mp1 0) (nincx 0))
      (declare (type fixnum nincx mp1 m i))
      (if (or (<= n 0) (<= incx 0)) (go end_label))
      (if (= incx 1) (go label20))
      (setf nincx (f2cl-lib:int-mul n incx))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i incx))
                    ((> i nincx) nil)
        (tagbody
          (setf (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%)
                  (* da (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%)))))
      (go end_label)
 label20
      (setf m (mod n 5))
      (if (= m 0) (go label40))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i m) nil)
        (tagbody
          (setf (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%)
                  (* da (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%)))))
      (if (< n 5) (go end_label))
 label40
      (setf mp1 (f2cl-lib:int-add m 1))
      (f2cl-lib:fdo (i mp1 (f2cl-lib:int-add i 5))
                    ((> i n) nil)
        (tagbody
          (setf (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%)
                  (* da (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%)))
          (setf (f2cl-lib:fref dx-%data%
                               ((f2cl-lib:int-add i 1))
                               ((1 *))
                               dx-%offset%)
                  (* da
                     (f2cl-lib:fref dx-%data%
                                    ((f2cl-lib:int-add i 1))
                                    ((1 *))
                                    dx-%offset%)))
          (setf (f2cl-lib:fref dx-%data%
                               ((f2cl-lib:int-add i 2))
                               ((1 *))
```

```
                                dx-%offset%)
                        (* da
                           (f2cl-lib:fref dx-%data%
                                          ((f2cl-lib:int-add i 2))
                                          ((1 *))
                                          dx-%offset%)))
                (setf (f2cl-lib:fref dx-%data%
                                     ((f2cl-lib:int-add i 3))
                                     ((1 *))
                                     dx-%offset%)
                        (* da
                           (f2cl-lib:fref dx-%data%
                                          ((f2cl-lib:int-add i 3))
                                          ((1 *))
                                          dx-%offset%)))
                (setf (f2cl-lib:fref dx-%data%
                                     ((f2cl-lib:int-add i 4))
                                     ((1 *))
                                     dx-%offset%)
                        (* da
                           (f2cl-lib:fref dx-%data%
                                          ((f2cl-lib:int-add i 4))
                                          ((1 *))
                                          dx-%offset%)))))
   end_label
        (return (values nil nil nil nil)))))
```

4.9 dswap BLAS

⟨*dswap.input*⟩≡

```
  )set break resume
  )sys rm -f dswap.output
  )spool dswap.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dswap.help*⟩≡

```
======================================================================
dswap examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DSWAP - BLAS level one, Swaps two double precision vectors

SYNOPSIS
       SUBROUTINE DSWAP    ( n, x, incx, y, incy )

           INTEGER           n, incx, incy

           DOUBLE PRECISION  x, y


DESCRIPTION
       DSWAP  swaps  two double precision vectors, it interchanges n values of
       vector x and vector y.  incx and incy specify the increment between two
       consecutive elements of respectively vector x and y.


       This routine performs the following vector operation:

                 x <-> y

            where x and y are double precision vectors.

ARGUMENTS
       n       INTEGER. (input)
               Number of vector elements to be swapped.
               If n <= 0, this routine returns without computation.

       x       DOUBLE PRECISION, (input and output)
               Array of dimension (n-1) * |incx| + 1.

       incx    INTEGER. (input)
               Increment between elements of x.
               If incx = 0, the results will be unpredictable.

       y       DOUBLE PRECISION, (input and output)
               array of dimension (n-1) * |incy| + 1. Vector to be swapped.
```

```
        incy    INTEGER. (input)
                Increment between elements of y.  If incy = 0, the results will
                be unpredictable.

NOTES
        When  working  backward  (incx < 0 or incy < 0), each routine starts at
        the end of the vector and moves backward, as follows:

             x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

             y(1-incy * (n-1)), y(1-incy * (n-2)), ..., y(1)
```

⟨*BLAS 1 dswap*⟩≡

```
(defun dswap (n dx incx dy incy)
  (declare (type (simple-array double-float (*)) dy dx)
           (type fixnum incy incx n))
  (f2cl-lib:with-multi-array-data
      ((dx double-float dx-%data% dx-%offset%)
       (dy double-float dy-%data% dy-%offset%))
    (prog ((i 0) (ix 0) (iy 0) (m 0) (mp1 0) (dtemp 0.0))
      (declare (type (double-float) dtemp)
               (type fixnum mp1 m iy ix i))
      (if (<= n 0) (go end_label))
      (if (and (= incx 1) (= incy 1)) (go label20))
      (setf ix 1)
      (setf iy 1)
      (if (< incx 0)
          (setf ix
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incx)
                   1)))
      (if (< incy 0)
          (setf iy
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incy)
                   1)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf dtemp (f2cl-lib:fref dx-%data% (ix) ((1 *)) dx-%offset%))
          (setf (f2cl-lib:fref dx-%data% (ix) ((1 *)) dx-%offset%)
                  (f2cl-lib:fref dy-%data% (iy) ((1 *)) dy-%offset%))
          (setf (f2cl-lib:fref dy-%data% (iy) ((1 *)) dy-%offset%) dtemp)
          (setf ix (f2cl-lib:int-add ix incx))
          (setf iy (f2cl-lib:int-add iy incy))))
      (go end_label)
label20
      (setf m (mod n 3))
      (if (= m 0) (go label40))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i m) nil)
        (tagbody
          (setf dtemp (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%))
          (setf (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%)
                  (f2cl-lib:fref dy-%data% (i) ((1 *)) dy-%offset%))
          (setf (f2cl-lib:fref dy-%data% (i) ((1 *)) dy-%offset%) dtemp)))
      (if (< n 3) (go end_label))
label40
```

```
      (setf mp1 (f2cl-lib:int-add m 1))
      (f2cl-lib:fdo (i mp1 (f2cl-lib:int-add i 3))
                    ((> i n) nil)
        (tagbody
          (setf dtemp (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%))
          (setf (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%)
                  (f2cl-lib:fref dy-%data% (i) ((1 *)) dy-%offset%))
          (setf (f2cl-lib:fref dy-%data% (i) ((1 *)) dy-%offset%) dtemp)
          (setf dtemp
                  (f2cl-lib:fref dx-%data%
                                 ((f2cl-lib:int-add i 1))
                                 ((1 *))
                                 dx-%offset%))
          (setf (f2cl-lib:fref dx-%data%
                               ((f2cl-lib:int-add i 1))
                               ((1 *))
                               dx-%offset%)
                  (f2cl-lib:fref dy-%data%
                                 ((f2cl-lib:int-add i 1))
                                 ((1 *))
                                 dy-%offset%))
          (setf (f2cl-lib:fref dy-%data%
                               ((f2cl-lib:int-add i 1))
                               ((1 *))
                               dy-%offset%)
                  dtemp)
          (setf dtemp
                  (f2cl-lib:fref dx-%data%
                                 ((f2cl-lib:int-add i 2))
                                 ((1 *))
                                 dx-%offset%))
          (setf (f2cl-lib:fref dx-%data%
                               ((f2cl-lib:int-add i 2))
                               ((1 *))
                               dx-%offset%)
                  (f2cl-lib:fref dy-%data%
                                 ((f2cl-lib:int-add i 2))
                                 ((1 *))
                                 dy-%offset%))
          (setf (f2cl-lib:fref dy-%data%
                               ((f2cl-lib:int-add i 2))
                               ((1 *))
                               dy-%offset%)
                  dtemp)))
    end_label
      (return (values nil nil nil nil nil)))))
```

4.10 dzasum BLAS

⟨*dzasum.input*⟩≡

```
)set break resume
)sys rm -f dzasum.output
)spool dzasum.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dzasum.help*⟩≡

```
======================================================================
dzasum examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DZASUM - BLAS level one, sums the absolute values of the real and imag-
       inary parts of the elements of a double complex vector

SYNOPSIS
       DOUBLE PRECISION FUNCTION DZASUM ( n, x, incx )

           INTEGER                     n, incx

           DOUBLE COMPLEX              x


DESCRIPTION
       This routine performs the following vector operation:

                          n
                DZASUM <-- Sum abs( real(x(i)) ) + abs( aimag(x(i)) )
                           i=1

ARGUMENTS
       n       INTEGER. (input)
               Number of vector elements to be summed.

       x       DOUBLE COMPLEX. (input)
               Array of dimension (n-1) * abs(incx) + 1.
               Vector that contains elements to be summed.

       incx    INTEGER. (input)
               Increment between elements of x.
               If incx = 0, the results will be unpredictable.

RETURN VALUES
       DZASUM  DOUBLE PRECISION. (output)
               Sum of the absolute values of the real and imaginary  parts  of
               the elements of the vector x.
               If n <= 0, DZASUM is set to 0.
```

```
NOTES
       When working backward (incx < 0), each routine starts at the end of the
       vector and moves backward, as follows:

            x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)
```

Computes (complex double-float) $asum \leftarrow ||re(x)||_1 + ||im(x)||_1$

Arguments are:

- n - fixnum
- dx - array (complex double-float)
- incx - fixnum

Return values are:

- 1 nil
- 2 nil
- 3 nil

⟨*BLAS 1 dzasum*⟩≡

```
(defun dzasum (n zx incx)
  (declare (type (simple-array (complex double-float) (*)) zx)
           (type fixnum incx n))
  (f2cl-lib:with-multi-array-data
      ((zx (complex double-float) zx-%data% zx-%offset%))
    (prog ((i 0) (ix 0) (stemp 0.0) (dzasum 0.0))
      (declare (type (double-float) dzasum stemp)
               (type fixnum ix i))
      (setf dzasum 0.0)
      (setf stemp 0.0)
      (if (or (<= n 0) (<= incx 0)) (go end_label))
      (if (= incx 1) (go label20))
      (setf ix 1)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf stemp
                  (+ stemp
                     (dcabs1
                      (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%))))
          (setf ix (f2cl-lib:int-add ix incx))))
      (setf dzasum stemp)
      (go end_label)
 label20
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf stemp
                  (+ stemp
```

```
                      (dcabs1
                       (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%))))))
        (setf dzasum stemp)
   end_label
        (return (values dzasum nil nil nil)))))
```

4.11 dznrm2 BLAS

⟨*dznrm2.input*⟩≡

```
  )set break resume
  )sys rm -f dznrm2.output
  )spool dznrm2.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dznrm2.help*⟩≡

```
  ========================================================================
  dznrm2 examples
  ========================================================================

  ========================================================================
  Man Page Details
  ========================================================================

  NAME
         DZNRM2 - BLAS level one, computes the Euclidean norm of a vector

  SYNOPSIS
         DOUBLE PRECISION FUNCTION DZNRM2 ( n, x, incx )

             INTEGER                       n, incx

             DOUBLE COMPLEX                x


  DESCRIPTION
         DZNRM2  computes  the Euclidean (L2) norm of a double precision complex
         vector:

                 DZNRM2 <-- ||x||
                                 2

         where x is a double precision complex vector.


  ARGUMENTS
         n       INTEGER (input)
                 Number of elements in the operand vector.

         x       DOUBLE COMPLEX (input)
                 Array of dimension (n-1) * |incx| + 1.
                 Array x contains the operand vector.

         incx    INTEGER (input)
                 Increment between elements of x.
                 If incx = 0, the results will be unpredictable.

  RETURN VALUES
         DZNRM2  DOUBLE PRECISION Result (Euclidean norm). (output)
                 If n <= 0, DZNRM2 is set to 0d0.
```

```
NOTES
       When  working backward (incx < 0), DZNRM2 starts at the end of the vec-
       tor and moves backward, as follows:

               x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)
```

⟨*BLAS 1 dznrm2*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dznrm2 (n x incx)
    (declare (type (simple-array (complex double-float) (*)) x)
             (type fixnum incx n))
    (f2cl-lib:with-multi-array-data
        ((x (complex double-float) x-%data% x-%offset%))
      (prog ((norm 0.0) (scale 0.0) (ssq 0.0) (temp 0.0) (ix 0) (dznrm2 0.0))
        (declare (type fixnum ix)
                 (type (double-float) norm scale ssq temp dznrm2))
        (cond
          ((or (< n 1) (< incx 1))
           (setf norm zero))
          (t
           (setf scale zero)
           (setf ssq one)
           (f2cl-lib:fdo (ix 1 (f2cl-lib:int-add ix incx))
                         ((> ix
                             (f2cl-lib:int-add 1
                              (f2cl-lib:int-mul
                               (f2cl-lib:int-add n
                                (f2cl-lib:int-sub 1))
                               incx)))
                          nil)
             (tagbody
               (cond
                 ((/=
                  (coerce (realpart (f2cl-lib:fref x (ix) ((1 *)))) 'double-float)
                   zero)
                  (setf temp
                          (abs
                           (coerce (realpart
                            (f2cl-lib:fref x-%data% (ix) ((1 *)) x-%offset%))
                            'double-float)))
                  (cond
                    ((< scale temp)
                     (setf ssq (+ one (* ssq (expt (/ scale temp) 2))))
                     (setf scale temp))
                    (t
                     (setf ssq (+ ssq (expt (/ temp scale) 2)))))))
               (cond
                 ((/= (f2cl-lib:dimag (f2cl-lib:fref x (ix) ((1 *)))) zero)
                  (setf temp
                          (abs
```

```
                            (f2cl-lib:dimag
                             (f2cl-lib:fref x-%data% (ix) ((1 *)) x-%offset%))))
                    (cond
                      ((< scale temp)
                       (setf ssq (+ one (* ssq (expt (/ scale temp) 2))))
                       (setf scale temp))
                      (t
                       (setf ssq (+ ssq (expt (/ temp scale) 2)))))))))
              (setf norm (* scale (f2cl-lib:fsqrt ssq)))))
           (setf dznrm2 norm)
   end_label
           (return (values dznrm2 nil nil nil))))))
```

4.12 icamax BLAS

⟨*icamax.input*⟩≡

```
  )set break resume
  )sys rm -f icamax.output
  )spool icamax.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*icamax.help*⟩≡

```
  ========================================================================
  icamax examples
  ========================================================================

  ========================================================================
  Man Page Details
  ========================================================================

  NAME
         ICAMAX - BLAS level one, maximum index function

  SYNOPSIS
         INTEGER FUNCTION ICAMAX ( n, x, incx )

             INTEGER               n, incx

             COMPLEX               x


  DESCRIPTION
         ICAMAX  searches a complex vector for the first occurrence of the maxi-
         mum absolute value.

         ICAMAX determines the first index i such that

         |Real(x )|+ |Imag(x ) | = MAX(|Real(x )| + | Imag(x )|): j = 1, ..., n
                i           i                 j             j
                 where x  is an element of a complex vector.
                        j

  ARGUMENTS
         n       INTEGER. (input)
                 Number of elements to process in the vector to be searched.  If
                 n <= 0, these routines return 0.

         x       COMPLEX. (input)
                 Array of dimension (n-1) * |incx| + 1.
                 Array x contains the vector to be searched.

         incx    INTEGER. (input)
                 Increment between elements of x.

  RETURN VALUES
         ICAMAX  INTEGER. (output)
                 Return  the first index of the maximum absolute value of vector
```

```
               x.  The vector x has length n and increment incx.

NOTES
       When working backward (incx < 0), each routine starts at the end of the
       vector and moves backward, as follows:

            x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

       The largest absolute value is:

            ABS (x(1+(index-1) * incx)) when incx > 0

            ABS (x(1+(n-index) * |incx|)) when incx < 0
```

⟨*BLAS 1 icamax*⟩≡

```
(defun icamax (n cx incx)
  (declare (type (simple-array (complex single-float) (*)) cx)
           (type fixnum incx n))
  (f2cl-lib:with-multi-array-data
      ((cx (complex single-float) cx-%data% cx-%offset%))
    (labels ((cabs1 (zdum)
               (+ (abs (coerce (realpart zdum) 'single-float))
                  (abs (f2cl-lib:aimag zdum)))))
      (declare (ftype (function (complex single-float)
                                (values single-float &rest t))
                      cabs1))
      (prog ((zdum #C(0.0f0 0.0f0)) (i 0) (ix 0) (smax 0.0f0) (icamax 0))
        (declare (type (single-float) smax)
                 (type fixnum icamax ix i)
                 (type (complex single-float) zdum))
        (setf icamax 0)
        (if (or (< n 1) (<= incx 0)) (go end_label))
        (setf icamax 1)
        (if (= n 1) (go end_label))
        (if (= incx 1) (go label20))
        (setf ix 1)
        (setf smax (cabs1 (f2cl-lib:fref cx-%data% (1) ((1 *)) cx-%offset%)))
        (setf ix (f2cl-lib:int-add ix incx))
        (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                      ((> i n) nil)
          (tagbody
            (if
             (<= (cabs1 (f2cl-lib:fref cx-%data% (ix) ((1 *)) cx-%offset%))
                 smax)
             (go label5))
            (setf icamax i)
            (setf smax
                    (cabs1 (f2cl-lib:fref cx-%data% (ix) ((1 *)) cx-%offset%)))
  label5
            (setf ix (f2cl-lib:int-add ix incx))))
        (go end_label)
  label20
        (setf smax (cabs1 (f2cl-lib:fref cx-%data% (1) ((1 *)) cx-%offset%)))
        (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                      ((> i n) nil)
          (tagbody
            (if
             (<= (cabs1 (f2cl-lib:fref cx-%data% (i) ((1 *)) cx-%offset%))
                 smax)
             (go label30))
```

```
            (setf icamax i)
            (setf smax
                    (cabs1 (f2cl-lib:fref cx-%data% (i) ((1 *)) cx-%offset%)))
  label30))
  end_label
         (return (values icamax nil nil nil))))))
```

4.13 idamax BLAS

⟨*idamax.input*⟩≡

```
  )set break resume
  )sys rm -f idamax.output
  )spool idamax.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*idamax.help*⟩≡

```
======================================================================
idamax examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       IDAMAX - BLAS level one, maximum index function

SYNOPSIS
       INTEGER FUNCTION IDAMAX ( n, x, incx )

           INTEGER               n, incx

           DOUBLE PRECISION      x


DESCRIPTION
       IDAMAX  searches  a double precision vector for the first occurrence of
       the the maximum absolute value. The vector x has length n and increment
       incx.

ARGUMENTS
       n       INTEGER. (input)
               Number of elements to process in the vector to be searched.  If
               n <= 0, these routines return 0.

       x       DOUBLE PRECISION. (input)
               Array of dimension (n-1) * |incx| + 1.
               Array x contains the vector to be searched.

       incx    INTEGER. (input)
               Increment between elements of x.

RETURN VALUES
       IDAMAX  INTEGER. (output)
               Return the first index of the maximum absolute value of  vector
               x.  The vector x has length n and increment incx.

NOTES
       When working backward (incx < 0), each routine starts at the end of the
       vector and moves backward, as follows:
```

```
     x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

The largest absolute value is:

     ABS (x(1+(index-1) * incx)) when incx > 0

     ABS (x(1+(n-index) * |incx|)) when incx < 0
```

⟨*BLAS 1 idamax*⟩≡

```
(defun idamax (n dx incx)
  (declare (type (simple-array double-float (*)) dx)
           (type fixnum incx n))
  (f2cl-lib:with-multi-array-data
      ((dx double-float dx-%data% dx-%offset%))
    (prog ((i 0) (ix 0) (dmax 0.0) (idamax 0))
      (declare (type (double-float) dmax)
               (type fixnum idamax ix i))
      (setf idamax 0)
      (if (or (< n 1) (<= incx 0)) (go end_label))
      (setf idamax 1)
      (if (= n 1) (go end_label))
      (if (= incx 1) (go label20))
      (setf ix 1)
      (setf dmax
       (the double-float (abs
        (the double-float
         (f2cl-lib:fref dx-%data% (1) ((1 *)) dx-%offset%)))))
      (setf ix (f2cl-lib:int-add ix incx))
      (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (if
           (<=
            (the double-float (abs
             (the double-float
              (f2cl-lib:fref dx-%data% (ix) ((1 *)) dx-%offset%))))
            dmax)
           (go label5))
          (setf idamax i)
          (setf dmax
           (the double-float (abs
            (the double-float
             (f2cl-lib:fref dx-%data% (ix) ((1 *)) dx-%offset%)))))
 label5
          (setf ix (f2cl-lib:int-add ix incx))))
      (go end_label)
 label20
      (setf dmax
       (the double-float (abs
        (the double-float
         (f2cl-lib:fref dx-%data% (1) ((1 *)) dx-%offset%)))))
      (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
```

```
            (if
             (<=
              (the double-float (abs
               (the double-float
                (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%))))
              dmax)
             (go label30))
            (setf idamax i)
            (setf dmax
             (the double-float (abs
              (the double-float
               (f2cl-lib:fref dx-%data% (i) ((1 *)) dx-%offset%)))))
   label30))
   end_label
        (return (values idamax nil nil nil)))))
```

4.14 isamax BLAS

⟨*isamax.input*⟩≡

```
  )set break resume
  )sys rm -f isamax.output
  )spool isamax.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*isamax.help*⟩≡

```
==========================================================================
isamax examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

NAME
       ISAMAX - BLAS level one, maximum index function

SYNOPSIS
       INTEGER FUNCTION ISAMAX ( n, x, incx )

           INTEGER               n, incx

           REAL                  x


DESCRIPTION
       ISAMAX searches a real vector for the first occurrence of the the maxi-
       mum absolute value. The vector x has length n and increment incx.

       ISAMAX returns the first index i such that
               |x | = MAX |x | : j = 1, ..., n
                 i          j
       where x  is an element of a real vector.
              j

ARGUMENTS
       n       INTEGER. (input)
               Number of elements to process in the vector to be searched.  If
               n <= 0, these routines return 0.

       x       REAL. (input)
               Array of dimension (n-1) * |incx| + 1.
               Array x contains the vector to be searched.

       incx    INTEGER. (input)
               Increment between elements of x.

RETURN VALUES
       ISAMAX  INTEGER. (output)
               Return  the first index of the maximum absolute value of vector
               x.  The vector x has length n and increment incx.
```

```
NOTES
       When working backward (incx < 0), each routine starts at the end of the
       vector and moves backward, as follows:

            x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

       The largest absolute value is:

            ABS (x(1+(index-1) * incx)) when incx > 0

            ABS (x(1+(n-index) * |incx|)) when incx < 0
```

⟨*BLAS 1 isamax*⟩≡

```
(defun isamax (n sx incx)
  (declare (type (simple-array single-float (*)) sx)
           (type fixnum incx n))
  (f2cl-lib:with-multi-array-data
      ((sx single-float sx-%data% sx-%offset%))
    (prog ((i 0) (ix 0) (smax 0.0f0) (isamax 0))
      (declare (type (single-float) smax)
               (type fixnum isamax ix i))
      (setf isamax 0)
      (if (or (< n 1) (<= incx 0)) (go end_label))
      (setf isamax 1)
      (if (= n 1) (go end_label))
      (if (= incx 1) (go label20))
      (setf ix 1)
      (setf smax (abs (f2cl-lib:fref sx-%data% (1) ((1 *)) sx-%offset%)))
      (setf ix (f2cl-lib:int-add ix incx))
      (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (if
           (<= (abs (f2cl-lib:fref sx-%data% (ix) ((1 *)) sx-%offset%)) smax)
           (go label5))
          (setf isamax i)
          (setf smax (abs (f2cl-lib:fref sx-%data% (ix) ((1 *)) sx-%offset%)))
  label5
          (setf ix (f2cl-lib:int-add ix incx))))
      (go end_label)
  label20
      (setf smax (abs (f2cl-lib:fref sx-%data% (1) ((1 *)) sx-%offset%)))
      (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (if (<= (abs (f2cl-lib:fref sx-%data% (i) ((1 *)) sx-%offset%)) smax)
              (go label30))
          (setf isamax i)
          (setf smax (abs (f2cl-lib:fref sx-%data% (i) ((1 *)) sx-%offset%)))
  label30))
  end_label
      (return (values isamax nil nil nil)))))
```

4.15 izamax BLAS

⟨*izamax.input*⟩≡

```
)set break resume
)sys rm -f izamax.output
)spool izamax.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*izamax.help*⟩≡

```
======================================================================
izamax examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       IZAMAX - BLAS level one, maximum index function

SYNOPSIS
       INTEGER FUNCTION IZAMAX ( n, x, incx )

           INTEGER               n, incx

           DOUBLE COMPLEX        x


DESCRIPTION
       IZAMAX searches a double complex vector for the first occurrence of the
       maximum absolute value.

       IZAMAX determines the first index i such that

       |Real(x )|+ |Imag(x ) | = MAX(|Real(x )| + | Imag(x )|): j = 1, ..., n
              i           i                 j             j
               where x  is an element of a double complex vector.
                      j

ARGUMENTS
       n       INTEGER. (input)
               Number of elements to process in the vector to be searched.  If
               n <= 0, these routines return 0.

       x       DOUBLE COMPLEX. (input)
               Array of dimension (n-1) * |incx| + 1.
               Array x contains the vector to be searched.

       incx    INTEGER. (input)
               Increment between elements of x.

RETURN VALUES
       IZAMAX  INTEGER. (output)
               Return  the first index of the maximum absolute value of vector
```

```
               x.  The vector x has length n and increment incx.

NOTES
       When working backward (incx < 0), each routine starts at the end of the
       vector and moves backward, as follows:

            x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

       The largest absolute value is:

            ABS (x(1+(index-1) * incx)) when incx > 0

            ABS (x(1+(n-index) * |incx|)) when incx < 0
```

⟨*BLAS 1 izamax*⟩≡

```
(defun izamax (n zx incx)
  (declare (type (simple-array (complex double-float) (*)) zx)
           (type fixnum incx n))
  (f2cl-lib:with-multi-array-data
      ((zx (complex double-float) zx-%data% zx-%offset%))
    (prog ((i 0) (ix 0) (smax 0.0) (izamax 0))
      (declare (type (double-float) smax)
               (type fixnum izamax ix i))
      (setf izamax 0)
      (if (or (< n 1) (<= incx 0)) (go end_label))
      (setf izamax 1)
      (if (= n 1) (go end_label))
      (if (= incx 1) (go label20))
      (setf ix 1)
      (setf smax (dcabs1 (f2cl-lib:fref zx-%data% (1) ((1 *)) zx-%offset%)))
      (setf ix (f2cl-lib:int-add ix incx))
      (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (if
           (<= (dcabs1 (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%))
               smax)
           (go label5))
          (setf izamax i)
          (setf smax
                  (dcabs1 (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%)))
  label5
          (setf ix (f2cl-lib:int-add ix incx))))
      (go end_label)
  label20
      (setf smax (dcabs1 (f2cl-lib:fref zx-%data% (1) ((1 *)) zx-%offset%)))
      (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (if
           (<= (dcabs1 (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%)) smax)
           (go label30))
          (setf izamax i)
          (setf smax
                  (dcabs1 (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%)))
  label30))
  end_label
      (return (values izamax nil nil nil)))))
```

4.16 zaxpy BLAS

⟨*zaxpy.input*⟩≡

```
)set break resume
)sys rm -f zaxpy.output
)spool zaxpy.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zaxpy.help*⟩≡

```
======================================================================
zaxpy examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       ZAXPY - BLAS level one axpy subroutine

SYNOPSIS
       SUBROUTINE ZAXPY    ( n, alpha, x, incx, y, incy )

           INTEGER           n, incx, incy

           DOUBLE COMPLEX    alpha, x, y


DESCRIPTION
       ZAXPY adds a scalar multiple of a double complex vector to another dou-
       ble complex vector.

       ZAXPY computes a constant alpha times a vector x plus a vector y.   The
       result overwrites the initial values of vector y.

       This routine performs the following vector operation:

            y  <--  alpha*x + y

       incx and incy specify the increment between two consecutive
       elements of respectively vector x and y.

ARGUMENTS
       n       INTEGER. (input)
               Number  of  elements in the vectors.  If n <= 0, these routines
               return without any computation.

       alpha   DOUBLE COMPLEX. (input)
               If alpha = 0 this routine returns without any computation.

       x       DOUBLE COMPLEX. (input)
               Array of dimension (n-1) * |incx| + 1.  Contains the vector  to
               be scaled before summation.
```

```
        incx    INTEGER. (input)
                Increment between elements of x.
                If incx = 0, the results will be unpredictable.

        y       DOUBLE COMPLEX. (input and output)
                array of dimension (n-1) * |incy| + 1.
                Before calling the routine, y contains the vector to be summed.
                After the routine ends, y contains the result of the summation.

        incy    INTEGER. (input)
                Increment between elements of y.
                If incy = 0, the results will be unpredictable.

NOTES
       This  routine  is  Level  1  Basic  Linear Algebra Subprograms (Level 1
       BLAS).

       When working backward (incx < 0 or incy < 0), each  routine  starts  at
       the end of the vector and moves backward, as follows:

            x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

            y(1-incy * (n-1)), y(1-incy * (n-2)), ..., y(1)

RETURN VALUES
       When  n  <=  0, double complex alpha = 0 = 0.+0.i, this routine returns
       immediately with no change in its arguments.
```

Computes (complex double-float) $y \leftarrow \alpha x + y$

Arguments are:

- n - fixnum
- da - (complex double-float)
- dx - array (complex double-float)
- incx - fixnum
- dy - array (complex double-float)
- incy - fixnum

Return values are:

- 1 nil
- 2 nil
- 3 nil
- 4 nil
- 5 nil
- 6 nil

⟨*BLAS 1 zaxpy*⟩≡

```
(defun zaxpy (n za zx incx zy incy)
  (declare (type (simple-array (complex double-float) (*)) zy zx)
           (type (complex double-float) za)
           (type fixnum incy incx n))
  (f2cl-lib:with-multi-array-data
      ((zx (complex double-float) zx-%data% zx-%offset%)
       (zy (complex double-float) zy-%data% zy-%offset%))
    (prog ((i 0) (ix 0) (iy 0))
      (declare (type fixnum iy ix i))
      (if (<= n 0) (go end_label))
      (if (= (dcabs1 za) 0.0) (go end_label))
      (if (and (= incx 1) (= incy 1)) (go label20))
      (setf ix 1)
      (setf iy 1)
      (if (< incx 0)
          (setf ix
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incx)
                   1)))
```

```
        (if (< incy 0)
            (setf iy
                    (f2cl-lib:int-add
                     (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incy)
                     1)))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref zy-%data% (iy) ((1 *)) zy-%offset%)
                    (+ (f2cl-lib:fref zy-%data% (iy) ((1 *)) zy-%offset%)
                       (* za
                          (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%))))
            (setf ix (f2cl-lib:int-add ix incx))
            (setf iy (f2cl-lib:int-add iy incy))))
        (go end_label)
   label20
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref zy-%data% (i) ((1 *)) zy-%offset%)
               (+ (f2cl-lib:fref zy-%data% (i) ((1 *)) zy-%offset%)
                  (* za (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%))))))
   end_label
        (return (values nil nil nil nil nil nil)))))
```

4.17 zcopy BLAS

⟨*zcopy.input*⟩≡

```
)set break resume
)sys rm -f zcopy.output
)spool zcopy.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zcopy.help*⟩≡

```
======================================================================
zcopy examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       ZCOPY  -  BLAS  level  one, copies a double complex vector into another
       double complex vector

SYNOPSIS
       SUBROUTINE ZCOPY    ( n, x, incx, y, incy )

           INTEGER           n, incx, incy

           DOUBLE COMPLEX    x, y


DESCRIPTION
       ZCOPY copies a double complex vector into another double  complex  vec-
       tor.   ZCOPY  copies  a vector x, whose length is n to a vector y. incx
       and incy specify the increment  between  two  consecutive  elements  of
       respectively vector x and y.

       This routine performs the following vector operation:

            y  <--  x

       where x and y are double complex vectors.

ARGUMENTS
       n       INTEGER. (input)
               Number of vector elements to be copied.
               If n <= 0, this routine returns without computation.

       x       DOUBLE COMPLEX, (input)
               Vector from which to copy.

       incx    INTEGER. (input)
               Increment between elements of x.
               If incx = 0, the results will be unpredictable.

       y       DOUBLE COMPLEX, (output)
```

```
                array of dimension (n-1) * |incy| + 1, result vector.

        incy    INTEGER. (input)
                Increment between elements of y.  If incy = 0, the results will
                be unpredictable.

NOTES
        When  working  backward  (incx < 0 or incy < 0), each routine starts at
        the end of the vector and moves backward, as follows:

             x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

             y(1-incy * (n-1)), y(1-incy * (n-2)), ..., y(1)
```

⟨*BLAS 1 zcopy*⟩≡

```
(defun zcopy (n zx incx zy incy)
  (declare (type (simple-array (complex double-float) (*)) zy zx)
           (type fixnum incy incx n))
  (f2cl-lib:with-multi-array-data
      ((zx (complex double-float) zx-%data% zx-%offset%)
       (zy (complex double-float) zy-%data% zy-%offset%))
    (prog ((i 0) (ix 0) (iy 0))
      (declare (type fixnum iy ix i))
      (if (<= n 0) (go end_label))
      (if (and (= incx 1) (= incy 1)) (go label20))
      (setf ix 1)
      (setf iy 1)
      (if (< incx 0)
          (setf ix
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incx)
                   1)))
      (if (< incy 0)
          (setf iy
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incy)
                   1)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf (f2cl-lib:fref zy-%data% (iy) ((1 *)) zy-%offset%)
                  (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%))
          (setf ix (f2cl-lib:int-add ix incx))
          (setf iy (f2cl-lib:int-add iy incy))))
      (go end_label)
label20
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf (f2cl-lib:fref zy-%data% (i) ((1 *)) zy-%offset%)
                  (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%))))
end_label
      (return (values nil nil nil nil nil)))))
```

4.18 zdotc BLAS

⟨*zdotc.input*⟩≡

```
)set break resume
)sys rm -f zdotc.output
)spool zdotc.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zdotc.help*⟩≡

```
=====================================================================
zdotc examples
=====================================================================

=====================================================================
Man Page Details
=====================================================================

NAME
       ZDOTC  - BLAS level one, computes the hermitian dot product of vector x
       and vector y.

SYNOPSIS
       DOUBLE COMPLEX FUNCTION ZDOTC ( n, x, incx, y, incy )

           INTEGER                     n, incx, incy

           COMPLEX*16                  x, y


DESCRIPTION
       ZDOTC computes a dot product of the conjugate of a complex  vector  and
       another     complex     vector     (l     complex    inner    product).
       2

       ZDOTC computes a dot product of the conjugate of a complex  vector  and
       another     complex     vector     (l     complex    inner    product).
       2

       This routine performs the following vector operation:

            ZDOTC  <--  (conjugate transpose of x) * y
                            n
                        =  Sum  (complex conjugate of x(i))*y(i)
                           i=1
                                                 H
       where x and y are complex vectors, and x is the conjugate
       transpose of x.

       If n <= 0, ZDOTC is set to 0.

ARGUMENTS
       n       INTEGER. (input)
               Number of elements in each vector.
```

```
        x       COMPLEX*16. (input)
                Array of dimension (n-1) * |incx| + 1.
                Array x contains the first vector operand.

        incx    INTEGER. (input)
                Increment between elements of x.
                If incx = 0, the results will be unpredictable.

        y       COMPLEX*16. (input)
                Array of dimension (n-1) * |incy| + 1.
                Array y contains the second vector operand.

        incy    INTEGER. (input)
                Increment between elements of y.
                If incy = 0, the results will be unpredictable.

RETURN VALUES
       ZDOTC    DOUBLE COMPLEX. Result (dot product). (output)

NOTES
       When  working  backward  (incx < 0 or incy < 0), each routine starts at
       the end of the vector and moves backward, as follows:

            x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

            y(1-incy * (n-1)), y(1-incy * (n-2)), ..., y(1)
```

⟨*BLAS 1 zdotc*⟩≡

```
(defun zdotc (n zx incx zy incy)
  (declare (type (simple-array (complex double-float) (*)) zy zx)
           (type fixnum incy incx n))
  (f2cl-lib:with-multi-array-data
      ((zx (complex double-float) zx-%data% zx-%offset%)
       (zy (complex double-float) zy-%data% zy-%offset%))
    (prog ((i 0) (ix 0) (iy 0) (ztemp #C(0.0 0.0)) (zdotc #C(0.0 0.0)))
      (declare (type (complex double-float) zdotc ztemp)
               (type fixnum iy ix i))
      (setf ztemp (complex 0.0 0.0))
      (setf zdotc (complex 0.0 0.0))
      (if (<= n 0) (go end_label))
      (if (and (= incx 1) (= incy 1)) (go label20))
      (setf ix 1)
      (setf iy 1)
      (if (< incx 0)
          (setf ix
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incx)
                   1)))
      (if (< incy 0)
          (setf iy
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incy)
                   1)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf ztemp
                  (+ ztemp
                     (*
                      (f2cl-lib:dconjg
                       (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%))
                      (f2cl-lib:fref zy-%data% (iy) ((1 *)) zy-%offset%))))
          (setf ix (f2cl-lib:int-add ix incx))
          (setf iy (f2cl-lib:int-add iy incy))))
      (setf zdotc ztemp)
      (go end_label)
   label20
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf ztemp
                  (+ ztemp
                     (*
```

```
                        (f2cl-lib:dconjg
                         (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%))
                        (f2cl-lib:fref zy-%data% (i) ((1 *)) zy-%offset%))))))
        (setf zdotc ztemp)
   end_label
        (return (values zdotc nil nil nil nil nil)))))
```

4.19 zdotu BLAS

⟨*zdotu.input*⟩≡

```
  )set break resume
  )sys rm -f zdotu.output
  )spool zdotu.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*zdotu.help*⟩≡

```
=====================================================================
zdotu examples
=====================================================================

=====================================================================
Man Page Details
=====================================================================

NAME
       ZDOTU - BLAS level one, computes computes a dot product (inner product)
       of two complex vectors

SYNOPSIS
       DOUBLE COMPLEX FUNCTION ZDOTU ( n, x, incx, y, incy )

           INTEGER                     n, incx, incy

           COMPLEX*16                  x, y


DESCRIPTION
       ZDOTU computes a dot product of two complex vectors.

       This routine performs the following vector operation:

                                                    n
               ZDOTU  <--  (transpose of x) * y  =  Sum  x(i)*y(i)
                                                   i=1
                                        T
       where x and y are real vectors, and x is the transpose of
       x.

       If n <= 0, ZDOTU is set to 0.

ARGUMENTS
       n       INTEGER. (input)
               Number of elements in each vector.

       x       COMPLEX*16. (input)
               Array of dimension (n-1) * |incx| + 1.
               Array x contains the first vector operand.

       incx    INTEGER. (input)
               Increment between elements of x.
               If incx = 0, the results will be unpredictable.
```

```
        y       COMPLEX*16. (input)
                Array of dimension (n-1) * |incy| + 1.
                Array y contains the second vector operand.

        incy    INTEGER. (input)
                Increment between elements of y.
                If incy = 0, the results will be unpredictable.

RETURN VALUES
       ZDOTU   DOUBLE COMPLEX. Result (dot product). (output)

NOTES
       When  working  backward  (incx < 0 or incy < 0), each routine starts at
       the end of the vector and moves backward, as follows:

            x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

            y(1-incy * (n-1)), y(1-incy * (n-2)), ..., y(1)
```

⟨*BLAS 1 zdotu*⟩≡

```
(defun zdotu (n zx incx zy incy)
  (declare (type (simple-array (complex double-float) (*)) zy zx)
           (type fixnum incy incx n))
  (f2cl-lib:with-multi-array-data
      ((zx (complex double-float) zx-%data% zx-%offset%)
       (zy (complex double-float) zy-%data% zy-%offset%))
    (prog ((i 0) (ix 0) (iy 0) (ztemp #C(0.0 0.0)) (zdotu #C(0.0 0.0)))
      (declare (type (complex double-float) zdotu ztemp)
               (type fixnum iy ix i))
      (setf ztemp (complex 0.0 0.0))
      (setf zdotu (complex 0.0 0.0))
      (if (<= n 0) (go end_label))
      (if (and (= incx 1) (= incy 1)) (go label20))
      (setf ix 1)
      (setf iy 1)
      (if (< incx 0)
          (setf ix
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incx)
                   1)))
      (if (< incy 0)
          (setf iy
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incy)
                   1)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf ztemp
                  (+ ztemp
                     (* (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%)
                        (f2cl-lib:fref zy-%data% (iy) ((1 *)) zy-%offset%))))
          (setf ix (f2cl-lib:int-add ix incx))
          (setf iy (f2cl-lib:int-add iy incy))))
      (setf zdotu ztemp)
      (go end_label)
   label20
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf ztemp
                  (+ ztemp
                     (* (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%)
                        (f2cl-lib:fref zy-%data% (i) ((1 *)) zy-%offset%))))))
      (setf zdotu ztemp)
```

```
   end_label
        (return (values zdotu nil nil nil nil nil)))))
```

4.20 zdscal BLAS

⟨*zdscal.input*⟩≡

```
  )set break resume
  )sys rm -f zdscal.output
  )spool zdscal.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*zdscal.help*⟩≡

```
  ========================================================================
  zdscal examples
  ========================================================================

  ========================================================================
  Man Page Details
  ========================================================================

  NAME
         ZDSCAL - BLAS level one, Scales a double complex vector

  SYNOPSIS
         SUBROUTINE ZDSCAL    ( n, alpha, x, incx )

             INTEGER            n, incx

             DOUBLE COMPLEX     x

             DOUBLE PRECISION   alpha


  DESCRIPTION
         ZDSCAL  scales  a double complex vector with a double precision scalar.
         ZDSCAL scales the vector x of length n and increment incx by  the  con-
         stant alpha.

         This routine performs the following vector operation:

                   x <-- alpha x

         where alpha is a double precision scalar, and x is a double complex
         vector.

  ARGUMENTS
         n       INTEGER. (input)
                 Number of elements in the vector.
                 If n <= 0, this routine returns without computation.

         alpha   DOUBLE PRECISION. (input)
                 Value used to scale vector

         x       DOUBLE COMPLEX. (input and output)
                 Array  of dimension (n-1) * abs(incx) + 1. Vector to be scaled.

         incx    INTEGER. (input)
```

```
                Increment between elements of x.
                If incx = 0, the results will be unpredictable.

  NOTES
         When  working  backward  (incx < 0 or incy < 0), each routine starts at
         the end of the vector and moves backward, as follows:

              x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)
```

⟨*BLAS 1 zdscal*⟩≡

```
(defun zdscal (n da zx incx)
  (declare (type (simple-array (complex double-float) (*)) zx)
           (type (double-float) da)
           (type fixnum incx n))
  (f2cl-lib:with-multi-array-data
      ((zx (complex double-float) zx-%data% zx-%offset%))
    (prog ((i 0) (ix 0))
      (declare (type fixnum ix i))
      (if (or (<= n 0) (<= incx 0)) (go end_label))
      (if (= incx 1) (go label20))
      (setf ix 1)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%)
                  (* (coerce (complex da 0.0) '(complex doublefloat))
                     (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%)))
          (setf ix (f2cl-lib:int-add ix incx))))
      (go end_label)
 label20
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%)
                  (* (coerce (complex da 0.0) '(complex double-float))
                     (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%)))))
 end_label
      (return (values nil nil nil nil)))))
```

4.21 zrotg BLAS

⟨*zrotg.input*⟩≡

```
)set break resume
)sys rm -f zrotg.output
)spool zrotg.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zrotg.help*⟩≡

```
  ======================================================================
  zrotg examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         ZROTG - Extensions to BLAS level one rotation subroutines


  SYNOPSIS
              SUBROUTINE ZROTG ( a, b, c, s )

              DOUBLE COMPLEX
                           a, b, s

              DOUBLE PRECISION
                           c

  DESCRIPTION
         ZROTG  computes  the  elements  of  a Givens plane rotation matrix such
         that:

                _              _    _   _    _   _
               |      c     s  |    | a |    | r |
               |-congj(s)   c  | *  | b | =  | 0 |
               _              _    _   _    _   _

         where r = (a / sqrt(conjg(a)*a)) * sqrt ( conjg(a)*a + conjg(b)*b ) ,
         and the notation conjg(z) represents the complex conjugate of z.

         The Givens plane rotation can be used to introduce zero elements into
         a matrix selectively.



  ARGUMENTS
         a       (input and output) DOUBLE COMPLEX

                 First vector component.

                 On input, the first component of the vector to be rotated.   On
                 output,  a is overwritten by the unique complex number r, whose
```

size in the complex plane is the Euclidean norm of the complex vector (a,b), and whose direction in the complex plane is the same as that of the original complex element a.

```
if |a| != 0
r = a / |a| * sqrt( conjg(a)*a + conjg(b)*b )

if |a| = 0
r = b
```

b (input) DOUBLE COMPLEX

Second vector component.

The second component of the vector to be rotated.

c (output) DOUBLE PRECISION

Cosine of the angle of rotation.

```
if |a| != 0
c = |a| /  sqrt( conjg(a)*a + conjg(b)*b )

if |a| = 0
c = 0
```

s (output) DOUBLE COMPLEX
Sine of the angle of rotation.

```
if |a| != 0
c=a/|a|*conjg(b)/sqrt(conjg(a)*a+conjg(b)*b)

if |a| = 0
s = ( 1.0 , 0.0 )
```

(Complex Double-Float). Computes plane rotation. Arguments are:

- da - (complex double-float)
- db - (complex double-float)
- c - double-float
- s - (complex double-float)

Returns multiple values where:

- 1 da - ca
- 2 db - nil
- 3 c - c
- 4 s - s

⟨*BLAS 1 zrotg*⟩≡

```
(defun zrotg (ca cb c s)
  (declare (type (double-float) c) (type (complex double-float) s cb ca))
  (prog ((alpha #C(0.0 0.0)) (norm 0.0) (scale 0.0))
    (declare (type (double-float) scale norm)
             (type (complex double-float) alpha))
    (if (/= (f2cl-lib:cdabs ca) 0.0) (go label10))
    (setf c 0.0)
    (setf s (complex 1.0 0.0))
    (setf ca cb)
 (go label20)
 label10
    (setf scale
            (coerce (+ (f2cl-lib:cdabs ca) (f2cl-lib:cdabs cb)) 'double-float))
    (setf norm
            (* scale
               (f2cl-lib:dsqrt
                (+ (expt (f2cl-lib:cdabs (/ ca
                    (coerce (complex scale 0.0) '(complex double-float)))) 2)
                   (expt (f2cl-lib:cdabs (/ cb
                    (coerce (complex scale 0.0) '(complex double-float))))
                     2)))))
    (setf alpha (/ ca (f2cl-lib:cdabs ca)))
    (setf c (/ (f2cl-lib:cdabs ca) norm))
    (setf s (/ (* alpha (f2cl-lib:dconjg cb)) norm))
    (setf ca (* alpha norm))
 label20
    (return (values ca nil c s))))
```

4.22 zscal BLAS

⟨*zscal.input*⟩≡

```
)set break resume
)sys rm -f zscal.output
)spool zscal.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zscal.help*⟩≡

```
  ========================================================================
  zscal examples
  ========================================================================

  ========================================================================
  Man Page Details
  ========================================================================

  NAME
         ZSCAL - BLAS level one, Scales a double complex vector

  SYNOPSIS
         SUBROUTINE ZSCAL    ( n, alpha, x, incx )

             INTEGER            n, incx

             DOUBLE COMPLEX     x, alpha


  DESCRIPTION
         ZSCAL  scales  a  double  complex  vector with a double complex scalar.
         ZSCAL scales the vector x of length n and increment incx  by  the  con-
         stant alpha.

         This routine performs the following vector operation:

                   x <-- alpha x

         where alpha is a double complex scalar, and x is a double complex
         vector.

  ARGUMENTS
         n       INTEGER. (input)
                 Number of elements in the vector.
                 If n <= 0, this routine returns without computation.

         alpha   DOUBLE COMPLEX. (input)
                 Value used to scale vector

         x       DOUBLE COMPLEX. (input and output)
                 Array  of dimension (n-1) * abs(incx) + 1. Vector to be scaled.

         incx    INTEGER. (input)
                 Increment between elements of x.
                 If incx = 0, the results will be unpredictable.
```

```
  NOTES
         When  working  backward  (incx < 0 or incy < 0), each routine starts at
         the end of the vector and moves backward, as follows:

              x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)
```

⟨*BLAS 1 zscal*⟩≡

```
  (defun zscal (n za zx incx)
    (declare (type (simple-array (complex double-float) (*)) zx)
             (type (complex double-float) za)
             (type fixnum incx n))
    (f2cl-lib:with-multi-array-data
        ((zx (complex double-float) zx-%data% zx-%offset%))
      (prog ((i 0) (ix 0))
        (declare (type fixnum ix i))
        (if (or (<= n 0) (<= incx 0)) (go end_label))
        (if (= incx 1) (go label20))
        (setf ix 1)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%)
                    (* za (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%)))
            (setf ix (f2cl-lib:int-add ix incx))))
        (go end_label)
   label20
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%)
                    (* za (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%)))))
   end_label
        (return (values nil nil nil nil)))))
```

4.23 zswap BLAS

⟨*zswap.input*⟩≡

```
)set break resume
)sys rm -f zswap.output
)spool zswap.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zswap.help*⟩≡

```
  ========================================================================
  zswap examples
  ========================================================================

  ========================================================================
  Man Page Details
  ========================================================================

  NAME
         ZSWAP - BLAS level one, Swaps two double complex vectors

  SYNOPSIS
         SUBROUTINE ZSWAP    ( n, x, incx, y, incy )

             INTEGER           n, incx, incy

             DOUBLE COMPLEX    x, y


  DESCRIPTION
         ZSWAP  swaps  two  double  complex vectors, it interchanges n values of
         vector x and vector y.  incx and incy specify the increment between two
         consecutive elements of respectively vector x and y.


         This routine performs the following vector operation:

                   x <-> y

              where x and y are double complex vectors.

  ARGUMENTS
         n       INTEGER. (input)
                 Number of vector elements to be swapped.
                 If n <= 0, this routine returns without computation.

         x       DOUBLE COMPLEX, (input and output)
                 Array of dimension (n-1) * |incx| + 1.

         incx    INTEGER. (input)
                 Increment between elements of x.
                 If incx = 0, the results will be unpredictable.

         y       DOUBLE COMPLEX, (input and output)
                 array of dimension (n-1) * |incy| + 1. Vector to be swapped.
```

```
        incy    INTEGER. (input)
                Increment between elements of y.  If incy = 0, the results will
                be unpredictable.

NOTES
        When  working  backward  (incx < 0 or incy < 0), each routine starts at
        the end of the vector and moves backward, as follows:

             x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

             y(1-incy * (n-1)), y(1-incy * (n-2)), ..., y(1)
```

⟨*BLAS 1 zswap*⟩≡

```
(defun zswap (n zx incx zy incy)
  (declare (type (simple-array (complex double-float) (*)) zy zx)
           (type fixnum incy incx n))
  (f2cl-lib:with-multi-array-data
      ((zx (complex double-float) zx-%data% zx-%offset%)
       (zy (complex double-float) zy-%data% zy-%offset%))
    (prog ((i 0) (ix 0) (iy 0) (ztemp #C(0.0 0.0)))
      (declare (type (complex double-float) ztemp)
               (type fixnum iy ix i))
      (if (<= n 0) (go end_label))
      (if (and (= incx 1) (= incy 1)) (go label20))
      (setf ix 1)
      (setf iy 1)
      (if (< incx 0)
          (setf ix
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incx)
                   1)))
      (if (< incy 0)
          (setf iy
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incy)
                   1)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf ztemp (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%))
          (setf (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%)
                  (f2cl-lib:fref zy-%data% (iy) ((1 *)) zy-%offset%))
          (setf (f2cl-lib:fref zy-%data% (iy) ((1 *)) zy-%offset%) ztemp)
          (setf ix (f2cl-lib:int-add ix incx))
          (setf iy (f2cl-lib:int-add iy incy))))
      (go end_label)
label20
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf ztemp (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%))
          (setf (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%)
                  (f2cl-lib:fref zy-%data% (i) ((1 *)) zy-%offset%))
          (setf (f2cl-lib:fref zy-%data% (i) ((1 *)) zy-%offset%) ztemp)))
end_label
      (return (values nil nil nil nil nil)))))
```

Chapter 5

BLAS Level 2

5.1 dgbmv BLAS

⟨*dgbmv.input*⟩≡

```
)set break resume
)sys rm -f dgbmv.output
)spool dgbmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgbmv.help*⟩≡

```
==========================================================================
dgbmv examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

 NAME
      DGBMV - perform one of the matrix-vector operations   y :=
      alpha*A*x + beta*y, or y := alpha*A'*x + beta*y,

 SYNOPSIS
      SUBROUTINE DGBMV ( TRANS, M, N, KL, KU, ALPHA, A, LDA, X,
                         INCX, BETA, Y, INCY )

          DOUBLE       PRECISION ALPHA, BETA

          INTEGER      INCX, INCY, KL, KU, LDA, M, N

          CHARACTER*1  TRANS

          DOUBLE       PRECISION A( LDA, * ), X( * ), Y( * )

 PURPOSE
      DGBMV  performs one of the matrix-vector operations

      where alpha and beta are scalars, x and y are vectors and A
      is an m by n band matrix, with kl sub-diagonals and ku
      super-diagonals.

 PARAMETERS
      TRANS  - CHARACTER*1.
             On entry, TRANS specifies the operation to be per-
             formed as follows:

             TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.

             TRANS = 'T' or 't'   y := alpha*A'*x + beta*y.

             TRANS = 'C' or 'c'   y := alpha*A'*x + beta*y.

             Unchanged on exit.

      M      - INTEGER.
```

```
          On entry, M specifies the number of rows of the
          matrix A.  M must be at least zero.  Unchanged on
          exit.

N         - INTEGER.
          On entry, N specifies the number of columns of the
          matrix A.  N must be at least zero.  Unchanged on
          exit.

KL        - INTEGER.
          On entry, KL specifies the number of sub-diagonals of
          the matrix A. KL must satisfy  0 .le. KL.  Unchanged
          on exit.

KU        - INTEGER.
          On entry, KU specifies the number of super-diagonals
          of the matrix A. KU must satisfy  0 .le. KU.
          Unchanged on exit.

ALPHA     - DOUBLE PRECISION.
          On entry, ALPHA specifies the scalar alpha.
          Unchanged on exit.

A         - DOUBLE PRECISION array of DIMENSION ( LDA, n ).
          Before entry, the leading ( kl + ku + 1 ) by n part
          of the array A must contain the matrix of coeffi-
          cients, supplied column by column, with the leading
          diagonal of the matrix in row ( ku + 1 ) of the
          array, the first super-diagonal starting at position
          2 in row ku, the first sub-diagonal starting at posi-
          tion 1 in row ( ku + 2 ), and so on.  Elements in the
          array A that do not correspond to elements in the
          band matrix (such as the top left ku by ku triangle)
          are not referenced.  The following program segment
          will transfer a band matrix from conventional full
          matrix storage to band storage:

          DO 20, J = 1, N K = KU + 1 - J DO 10, I = MAX( 1, J -
          KU ), MIN( M, J + KL ) A( K + I, J ) = matrix( I, J )
          10    CONTINUE 20 CONTINUE

          Unchanged on exit.

LDA       - INTEGER.
          On entry, LDA specifies the first dimension of A as
          declared in the calling (sub) program. LDA must be at
```

```
         least ( kl + ku + 1 ).  Unchanged on exit.

X        - DOUBLE PRECISION array of DIMENSION at least
         ( 1 + ( n - 1 )*abs( INCX ) ) when TRANS = 'N' or 'n'
         and at least ( 1 + ( m - 1 )*abs( INCX ) ) otherwise.
         Before entry, the incremented array X must contain
         the vector x.  Unchanged on exit.

INCX     - INTEGER.
         On entry, INCX specifies the increment for the ele-
         ments of X. INCX must not be zero.  Unchanged on
         exit.

BETA     - DOUBLE PRECISION.
         On entry, BETA specifies the scalar beta. When BETA
         is supplied as zero then Y need not be set on input.
         Unchanged on exit.

Y        - DOUBLE PRECISION array of DIMENSION at least
         ( 1 + ( m - 1 )*abs( INCY ) ) when TRANS = 'N' or 'n'

         and at least ( 1 + ( n - 1 )*abs( INCY ) ) otherwise.
         Before entry, the incremented array Y must contain
         the vector y. On exit, Y is overwritten by the
         updated vector y.

INCY     - INTEGER.
         On entry, INCY specifies the increment for the ele-
         ments of Y. INCY must not be zero.  Unchanged on
         exit.
```

⟨*BLAS 2 dgbmv*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dgbmv (trans m n kl ku alpha a lda x incx beta y incy)
    (declare (type (simple-array double-float (*)) y x a)
             (type (double-float) beta alpha)
             (type fixnum incy incx lda ku kl n m)
             (type character trans))
    (f2cl-lib:with-multi-array-data
        ((trans character trans-%data% trans-%offset%)
         (a double-float a-%data% a-%offset%)
         (x double-float x-%data% x-%offset%)
         (y double-float y-%data% y-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (k 0) (kup1 0)
             (kx 0) (ky 0) (lenx 0) (leny 0) (temp 0.0))
        (declare (type fixnum i info ix iy j jx jy k kup1 kx ky
                                          lenx leny)
                 (type (double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 1))
          ((< m 0)
           (setf info 2))
          ((< n 0)
           (setf info 3))
          ((< kl 0)
           (setf info 4))
          ((< ku 0)
           (setf info 5))
          ((< lda (f2cl-lib:int-add kl ku 1))
           (setf info 8))
          ((= incx 0)
           (setf info 10))
          ((= incy 0)
           (setf info 13)))
        (cond
          ((/= info 0)
            (error
              " ** On entry to ~a parameter number ~a had an illegal value~%"
              "DGBMV" info)
           (go end_label)))
        (if (or (= m 0) (= n 0) (and (= alpha zero) (= beta one)))
```

```
    (go end_label))
(cond
  ((char-equal trans #\N)
   (setf lenx n)
   (setf leny m))
  (t
   (setf lenx m)
   (setf leny n)))
(cond
  ((> incx 0)
   (setf kx 1))
  (t
   (setf kx
           (f2cl-lib:int-sub 1
                             (f2cl-lib:int-mul
                              (f2cl-lib:int-sub lenx 1)
                              incx)))))
(cond
  ((> incy 0)
   (setf ky 1))
  (t
   (setf ky
           (f2cl-lib:int-sub 1
                             (f2cl-lib:int-mul
                              (f2cl-lib:int-sub leny 1)
                              incy)))))
(cond
  ((/= beta one)
   (cond
     ((= incy 1)
      (cond
        ((= beta zero)
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i leny) nil)
           (tagbody
             (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                     zero))))
        (t
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i leny) nil)
           (tagbody
             (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                     (* beta
                        (f2cl-lib:fref y-%data%
                                       (i)
                                       ((1 *))
```

```
                                          y-%offset%))))))))
          (t
           (setf iy ky)
           (cond
             ((= beta zero)
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                            ((> i leny) nil)
                (tagbody
                  (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                          zero)
                  (setf iy (f2cl-lib:int-add iy incy)))))
             (t
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                            ((> i leny) nil)
                (tagbody
                  (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                          (* beta
                             (f2cl-lib:fref y-%data%
                                            (iy)
                                            ((1 *))
                                            y-%offset%)))
                  (setf iy (f2cl-lib:int-add iy incy)))))))))
    (if (= alpha zero) (go end_label))
    (setf kup1 (f2cl-lib:int-add ku 1))
    (cond
      ((char-equal trans #\N)
       (setf jx kx)
       (cond
         ((= incy 1)
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                        ((> j n) nil)
            (tagbody
              (cond
                ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                 (setf temp
                         (* alpha
                            (f2cl-lib:fref x-%data%
                                           (jx)
                                           ((1 *))
                                           x-%offset%)))
                 (setf k (f2cl-lib:int-sub kup1 j))
                 (f2cl-lib:fdo (i
                                (max (the fixnum 1)
                                     (the fixnum
                                          (f2cl-lib:int-add j
                                                            (f2cl-lib:int-sub
```

```
                                                    ku))))
                                  (f2cl-lib:int-add i 1))
                                 ((> i
                                     (min (the fixnum m)
                                          (the fixnum
                                               (f2cl-lib:int-add j kl))))
                                  nil)
                            (tagbody
                              (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                                      (+
                                       (f2cl-lib:fref y-%data%
                                                      (i)
                                                      ((1 *))
                                                      y-%offset%)
                                       (* temp
                                          (f2cl-lib:fref a-%data%
                                                         ((f2cl-lib:int-add k i) j)
                                                         ((1 lda) (1 *))
                                                         a-%offset%))))))
                          (setf jx (f2cl-lib:int-add jx incx)))))
                     (t
                      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                    ((> j n) nil)
                        (tagbody
                          (cond
                            ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                             (setf temp
                                     (* alpha
                                        (f2cl-lib:fref x-%data%
                                                       (jx)
                                                       ((1 *))
                                                       x-%offset%)))
                             (setf iy ky)
                             (setf k (f2cl-lib:int-sub kup1 j))
                             (f2cl-lib:fdo (i
                                            (max (the fixnum 1)
                                                 (the fixnum
                                                      (f2cl-lib:int-add j
                                                                        (f2cl-lib:int-sub
                                                                         ku))))
                                            (f2cl-lib:int-add i 1))
                                           ((> i
                                               (min (the fixnum m)
                                                    (the fixnum
                                                         (f2cl-lib:int-add j kl))))
                                            nil)
```

```
                    (tagbody
                      (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                              (+
                               (f2cl-lib:fref y-%data%
                                              (iy)
                                              ((1 *))
                                              y-%offset%)
                               (* temp
                                  (f2cl-lib:fref a-%data%
                                                 ((f2cl-lib:int-add k i) j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))))
                      (setf iy (f2cl-lib:int-add iy incy))))))
              (setf jx (f2cl-lib:int-add jx incx))
              (if (> j ku) (setf ky (f2cl-lib:int-add ky incy))))))))
        (t
         (setf jy ky)
         (cond
           ((= incx 1)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                          ((> j n) nil)
              (tagbody
                (setf temp zero)
                (setf k (f2cl-lib:int-sub kup1 j))
                (f2cl-lib:fdo (i
                               (max (the fixnum 1)
                                    (the fixnum
                                         (f2cl-lib:int-add j
                                                           (f2cl-lib:int-sub
                                                            ku))))
                               (f2cl-lib:int-add i 1))
                              ((> i
                                  (min (the fixnum m)
                                       (the fixnum
                                            (f2cl-lib:int-add j kl))))
                               nil)
                  (tagbody
                    (setf temp
                            (+ temp
                               (*
                                (f2cl-lib:fref a-%data%
                                               ((f2cl-lib:int-add k i) j)
                                               ((1 lda) (1 *))
                                               a-%offset%)
                                (f2cl-lib:fref x-%data%
                                               (i)
```

```
                                              ((1 *))
                                              x-%offset%))))))
                    (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                            (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                               (* alpha temp)))
                    (setf jy (f2cl-lib:int-add jy incy)))))
                (t
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (setf temp zero)
                     (setf ix kx)
                     (setf k (f2cl-lib:int-sub kup1 j))
                     (f2cl-lib:fdo (i
                                    (max (the fixnum 1)
                                         (the fixnum
                                              (f2cl-lib:int-add j
                                                                (f2cl-lib:int-sub
                                                                 ku))))
                                    (f2cl-lib:int-add i 1))
                                   ((> i
                                       (min (the fixnum m)
                                            (the fixnum
                                                 (f2cl-lib:int-add j kl))))
                                    nil)
                       (tagbody
                         (setf temp
                                 (+ temp
                                    (*
                                     (f2cl-lib:fref a-%data%
                                                    ((f2cl-lib:int-add k i) j)
                                                    ((1 lda) (1 *))
                                                    a-%offset%)
                                     (f2cl-lib:fref x-%data%
                                                    (ix)
                                                    ((1 *))
                                                    x-%offset%))))
                         (setf ix (f2cl-lib:int-add ix incx))))
                     (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                             (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                (* alpha temp)))
                     (setf jy (f2cl-lib:int-add jy incy))
                     (if (> j ku) (setf kx (f2cl-lib:int-add kx incx)))))))))
  end_label
         (return
          (values nil nil nil nil nil nil nil nil nil nil nil nil nil))))))
```

5.2 dgemv BLAS

⟨*dgemv.input*⟩≡

```
)set break resume
)sys rm -f dgemv.output
)spool dgemv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgemv.help*⟩≡

```
==========================================================================
dgemv examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

 NAME
      DGEMV - perform one of the matrix-vector operations   y :=
      alpha*A*x + beta*y, or y := alpha*A'*x + beta*y,

 SYNOPSIS
      SUBROUTINE DGEMV ( TRANS, M, N, ALPHA, A, LDA, X, INCX,
                         BETA, Y, INCY )

          DOUBLE       PRECISION ALPHA, BETA

          INTEGER      INCX, INCY, LDA, M, N

          CHARACTER*1  TRANS

          DOUBLE       PRECISION A( LDA, * ), X( * ), Y( * )

 PURPOSE
      DGEMV  performs one of the matrix-vector operations

      where alpha and beta are scalars, x and y are vectors and A
      is an m by n matrix.

 PARAMETERS
      TRANS  - CHARACTER*1.
               On entry, TRANS specifies the operation to be per-
               formed as follows:

               TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.

               TRANS = 'T' or 't'   y := alpha*A'*x + beta*y.

               TRANS = 'C' or 'c'   y := alpha*A'*x + beta*y.

               Unchanged on exit.

      M      - INTEGER.
               On entry, M specifies the number of rows of the
```

```
          matrix A.  M must be at least zero.  Unchanged on
          exit.

N         - INTEGER.
          On entry, N specifies the number of columns of the
          matrix A.  N must be at least zero.  Unchanged on
          exit.

ALPHA     - DOUBLE PRECISION.
          On entry, ALPHA specifies the scalar alpha.
          Unchanged on exit.

A         - DOUBLE PRECISION array of DIMENSION ( LDA, n ).
          Before entry, the leading m by n part of the array A

          must contain the matrix of coefficients.  Unchanged
          on exit.

LDA       - INTEGER.
          On entry, LDA specifies the first dimension of A as
          declared in the calling (sub) program. LDA must be at
          least max( 1, m ).  Unchanged on exit.

X         - DOUBLE PRECISION array of DIMENSION at least
          ( 1 + ( n - 1 )*abs( INCX ) ) when TRANS = 'N' or 'n'
          and at least ( 1 + ( m - 1 )*abs( INCX ) ) otherwise.
          Before entry, the incremented array X must contain
          the vector x.  Unchanged on exit.

INCX      - INTEGER.
          On entry, INCX specifies the increment for the ele-
          ments of X. INCX must not be zero.  Unchanged on
          exit.

BETA      - DOUBLE PRECISION.
          On entry, BETA specifies the scalar beta. When BETA
          is supplied as zero then Y need not be set on input.
          Unchanged on exit.

Y         - DOUBLE PRECISION array of DIMENSION at least
          ( 1 + ( m - 1 )*abs( INCY ) ) when TRANS = 'N' or 'n'
          and at least ( 1 + ( n - 1 )*abs( INCY ) ) otherwise.
          Before entry with BETA non-zero, the incremented
          array Y must contain the vector y. On exit, Y is
          overwritten by the updated vector y.
```

```
INCY   - INTEGER.
         On entry, INCY specifies the increment for the ele-
         ments of Y. INCY must not be zero.  Unchanged on
         exit.
```

⟨*BLAS 2 dgemv*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dgemv (trans m n alpha a lda x incx beta y incy)
    (declare (type (simple-array double-float (*)) y x a)
             (type (double-float) beta alpha)
             (type fixnum incy incx lda n m)
             (type character trans))
    (f2cl-lib:with-multi-array-data
        ((trans character trans-%data% trans-%offset%)
         (a double-float a-%data% a-%offset%)
         (x double-float x-%data% x-%offset%)
         (y double-float y-%data% y-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (kx 0) (ky 0)
             (lenx 0) (leny 0) (temp 0.0))
        (declare (type fixnum i info ix iy j jx jy kx ky lenx
                                          leny)
                 (type (double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 1))
          ((< m 0)
           (setf info 2))
          ((< n 0)
           (setf info 3))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info 6))
          ((= incx 0)
           (setf info 8))
          ((= incy 0)
           (setf info 11)))
        (cond
          ((/= info 0)
           (error
             " ** On entry to ~a parameter number ~a had an illegal value~%"
             "DGEMV" info)
           (go end_label)))
        (if (or (= m 0) (= n 0) (and (= alpha zero) (= beta one)))
            (go end_label))
        (cond
          ((char-equal trans #\N)
           (setf lenx n)
```

```
   (setf leny m))
  (t
   (setf lenx m)
   (setf leny n)))
(cond
  ((> incx 0)
   (setf kx 1))
  (t
   (setf kx
           (f2cl-lib:int-sub 1
                             (f2cl-lib:int-mul
                              (f2cl-lib:int-sub lenx 1)
                              incx)))))
(cond
  ((> incy 0)
   (setf ky 1))
  (t
   (setf ky
           (f2cl-lib:int-sub 1
                             (f2cl-lib:int-mul
                              (f2cl-lib:int-sub leny 1)
                              incy)))))
(cond
  ((/= beta one)
   (cond
     ((= incy 1)
      (cond
        ((= beta zero)
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i leny) nil)
           (tagbody
             (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                     zero))))
        (t
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i leny) nil)
           (tagbody
             (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                     (* beta
                        (f2cl-lib:fref y-%data%
                                       (i)
                                       ((1 *))
                                       y-%offset%)))))))))
     (t
      (setf iy ky)
      (cond
```

```
            ((= beta zero)
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i leny) nil)
               (tagbody
                 (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                         zero)
                 (setf iy (f2cl-lib:int-add iy incy)))))
            (t
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i leny) nil)
               (tagbody
                 (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                         (* beta
                            (f2cl-lib:fref y-%data%
                                           (iy)
                                           ((1 *))
                                           y-%offset%)))
                 (setf iy (f2cl-lib:int-add iy incy))))))))
      (if (= alpha zero) (go end_label))
      (cond
        ((char-equal trans #\N)
         (setf jx kx)
         (cond
           ((= incy 1)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                          ((> j n) nil)
              (tagbody
                (cond
                  ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                   (setf temp
                           (* alpha
                              (f2cl-lib:fref x-%data%
                                             (jx)
                                             ((1 *))
                                             x-%offset%)))
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i m) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                               (+
                                (f2cl-lib:fref y-%data%
                                               (i)
                                               ((1 *))
                                               y-%offset%)
                                (* temp
                                   (f2cl-lib:fref a-%data%
```

```
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%))))))))
              (setf jx (f2cl-lib:int-add jx incx)))))
         (t
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                        ((> j n) nil)
            (tagbody
              (cond
                ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                 (setf temp
                         (* alpha
                            (f2cl-lib:fref x-%data%
                                           (jx)
                                           ((1 *))
                                           x-%offset%)))
                 (setf iy ky)
                 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                               ((> i m) nil)
                   (tagbody
                     (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                             (+
                              (f2cl-lib:fref y-%data%
                                             (iy)
                                             ((1 *))
                                             y-%offset%)
                              (* temp
                                 (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%))))
                     (setf iy (f2cl-lib:int-add iy incy))))))
              (setf jx (f2cl-lib:int-add jx incx)))))))
    (t
     (setf jy ky)
     (cond
       ((= incx 1)
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                      ((> j n) nil)
          (tagbody
            (setf temp zero)
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                          ((> i m) nil)
              (tagbody
                (setf temp
                        (+ temp
```

```
                                    (*
                                     (f2cl-lib:fref a-%data%
                                                    (i j)
                                                    ((1 lda) (1 *))
                                                    a-%offset%)
                                     (f2cl-lib:fref x-%data%
                                                    (i)
                                                    ((1 *))
                                                    x-%offset%))))))
                         (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                 (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                    (* alpha temp)))
                         (setf jy (f2cl-lib:int-add jy incy)))))
                    (t
                     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                   ((> j n) nil)
                       (tagbody
                         (setf temp zero)
                         (setf ix kx)
                         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                       ((> i m) nil)
                           (tagbody
                             (setf temp
                                     (+ temp
                                        (*
                                         (f2cl-lib:fref a-%data%
                                                        (i j)
                                                        ((1 lda) (1 *))
                                                        a-%offset%)
                                         (f2cl-lib:fref x-%data%
                                                        (ix)
                                                        ((1 *))
                                                        x-%offset%))))
                             (setf ix (f2cl-lib:int-add ix incx))))
                         (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                 (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                    (* alpha temp)))
                         (setf jy (f2cl-lib:int-add jy incy)))))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil nil nil nil))))))
```

5.3 dger BLAS

⟨*dger.input*⟩≡

```
)set break resume
)sys rm -f dger.output
)spool dger.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dger.help*⟩≡

```
  ======================================================================
  dger examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        DGER - perform the rank 1 operation   A := alpha*x*y' + A,

   SYNOPSIS
        SUBROUTINE DGER ( M, N, ALPHA, X, INCX, Y, INCY, A, LDA )

            DOUBLE      PRECISION ALPHA

            INTEGER     INCX, INCY, LDA, M, N

            DOUBLE      PRECISION A( LDA, * ), X( * ), Y( * )

   PURPOSE
        DGER   performs the rank 1 operation

        where alpha is a scalar, x is an m element vector, y is an n
        element vector and A is an m by n matrix.

   PARAMETERS
        M      - INTEGER.
                 On entry, M specifies the number of rows of the
                 matrix A.  M must be at least zero.  Unchanged on
                 exit.

        N      - INTEGER.
                 On entry, N specifies the number of columns of the
                 matrix A.  N must be at least zero.  Unchanged on
                 exit.

        ALPHA  - DOUBLE PRECISION.
                 On entry, ALPHA specifies the scalar alpha.
                 Unchanged on exit.

        X      - DOUBLE PRECISION array of dimension at least
                 ( 1 + ( m - 1 )*abs( INCX ) ).  Before entry, the
                 incremented array X must contain the m element vector
                 x.  Unchanged on exit.
```

```
INCX   - INTEGER.
       On entry, INCX specifies the increment for the ele-
       ments of X. INCX must not be zero.  Unchanged on
       exit.

Y      - DOUBLE PRECISION array of dimension at least
       ( 1 + ( n - 1 )*abs( INCY ) ).  Before entry, the
       incremented array Y must contain the n element vector
       y.  Unchanged on exit.

INCY   - INTEGER.
       On entry, INCY specifies the increment for the ele-
       ments of Y. INCY must not be zero.  Unchanged on

       exit.

A      - DOUBLE PRECISION array of DIMENSION ( LDA, n ).
       Before entry, the leading m by n part of the array A
       must contain the matrix of coefficients. On exit, A
       is overwritten by the updated matrix.

LDA    - INTEGER.
       On entry, LDA specifies the first dimension of A as
       declared in the calling (sub) program. LDA must be at
       least max( 1, m ).  Unchanged on exit.
```

⟨*BLAS 2 dger*⟩≡
```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dger (m n alpha x incx y incy a lda)
    (declare (type (simple-array double-float (*)) a y x)
             (type (double-float) alpha)
             (type fixnum lda incy incx n m))
    (f2cl-lib:with-multi-array-data
        ((x double-float x-%data% x-%offset%)
         (y double-float y-%data% y-%offset%)
         (a double-float a-%data% a-%offset%))
      (prog ((i 0) (info 0) (ix 0) (j 0) (jy 0) (kx 0) (temp 0.0))
        (declare (type fixnum i info ix j jy kx)
                 (type (double-float) temp))
        (setf info 0)
        (cond
          ((< m 0)
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((= incx 0)
           (setf info 5))
          ((= incy 0)
           (setf info 7))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info 9)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DYER" info)
           (go end_label)))
        (if (or (= m 0) (= n 0) (= alpha zero)) (go end_label))
        (cond
          ((> incy 0)
           (setf jy 1))
          (t
           (setf jy
                   (f2cl-lib:int-sub 1
                                     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                       incy)))))
        (cond
          ((= incx 1)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
```

```
      (cond
        ((/= (f2cl-lib:fref y (jy) ((1 *))) zero)
         (setf temp
                 (* alpha
                    (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)))
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i m) nil)
           (tagbody
             (setf (f2cl-lib:fref a-%data%
                                  (i j)
                                  ((1 lda) (1 *))
                                  a-%offset%)
                   (+
                    (f2cl-lib:fref a-%data%
                                   (i j)
                                   ((1 lda) (1 *))
                                   a-%offset%)
                    (*
                     (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%)
                     temp)))))))
      (setf jy (f2cl-lib:int-add jy incy)))))
(t
 (cond
   ((> incx 0)
    (setf kx 1))
   (t
    (setf kx
            (f2cl-lib:int-sub 1
                              (f2cl-lib:int-mul
                               (f2cl-lib:int-sub m 1)
                               incx)))))
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j n) nil)
   (tagbody
     (cond
       ((/= (f2cl-lib:fref y (jy) ((1 *))) zero)
        (setf temp
                (* alpha
                   (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)))
        (setf ix kx)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i m) nil)
          (tagbody
            (setf (f2cl-lib:fref a-%data%
                                 (i j)
                                 ((1 lda) (1 *))
```

```
                                        a-%offset%)
                                (+
                                 (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%)
                                 (*
                                  (f2cl-lib:fref x-%data%
                                                 (ix)
                                                 ((1 *))
                                                 x-%offset%)
                                  temp)))
                          (setf ix (f2cl-lib:int-add ix incx))))))
                 (setf jy (f2cl-lib:int-add jy incy))))))
   end_label
          (return (values nil nil nil nil nil nil nil nil nil))))))
```

5.4 dsbmv BLAS

⟨*dsbmv.input*⟩≡

```
  )set break resume
  )sys rm -f dsbmv.output
  )spool dsbmv.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dsbmv.help*⟩≡

```
==========================================================================
dsbmv examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

 NAME
      DSBMV - perform the matrix-vector operation   y := alpha*A*x
      + beta*y,

 SYNOPSIS
      SUBROUTINE DSBMV ( UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA,
                         Y, INCY )

          DOUBLE       PRECISION ALPHA, BETA

          INTEGER      INCX, INCY, K, LDA, N

          CHARACTER*1  UPLO

          DOUBLE       PRECISION A( LDA, * ), X( * ), Y( * )

 PURPOSE
      DSBMV  performs the matrix-vector  operation

      where alpha and beta are scalars, x and y are n element vec-
      tors and A is an n by n symmetric band matrix, with k
      super-diagonals.

 PARAMETERS
      UPLO   - CHARACTER*1.
               On entry, UPLO specifies whether the upper or lower
               triangular part of the band matrix A is being sup-
               plied as follows:

               UPLO = 'U' or 'u'   The upper triangular part of A is
               being supplied.

               UPLO = 'L' or 'l'   The lower triangular part of A is
               being supplied.

               Unchanged on exit.
```

```
N      - INTEGER.
       On entry, N specifies the order of the matrix A.  N
       must be at least zero.  Unchanged on exit.

K      - INTEGER.
       On entry, K specifies the number of super-diagonals
       of the matrix A. K must satisfy  0 .le. K.  Unchanged
       on exit.

ALPHA  - DOUBLE PRECISION.
       On entry, ALPHA specifies the scalar alpha.
       Unchanged on exit.

A      - DOUBLE PRECISION array of DIMENSION ( LDA, n ).

       Before entry with UPLO = 'U' or 'u', the leading ( k
       + 1 ) by n part of the array A must contain the upper
       triangular band part of the symmetric matrix, sup-
       plied column by column, with the leading diagonal of
       the matrix in row ( k + 1 ) of the array, the first
       super-diagonal starting at position 2 in row k, and
       so on. The top left k by k triangle of the array A is
       not referenced.  The following program segment will
       transfer the upper triangular part of a symmetric
       band matrix from conventional full matrix storage to
       band storage:

       DO 20, J = 1, N M = K + 1 - J DO 10, I = MAX( 1, J -
       K ), J A( M + I, J ) = matrix( I, J ) 10    CONTINUE
       20 CONTINUE

       Before entry with UPLO = 'L' or 'l', the leading ( k
       + 1 ) by n part of the array A must contain the lower
       triangular band part of the symmetric matrix, sup-
       plied column by column, with the leading diagonal of
       the matrix in row 1 of the array, the first sub-
       diagonal starting at position 1 in row 2, and so on.
       The bottom right k by k triangle of the array A is
       not referenced.  The following program segment will
       transfer the lower triangular part of a symmetric
       band matrix from conventional full matrix storage to
       band storage:

       DO 20, J = 1, N M = 1 - J DO 10, I = J, MIN( N, J + K
       ) A( M + I, J ) = matrix( I, J ) 10    CONTINUE 20
       CONTINUE
```

```
         Unchanged on exit.

LDA      - INTEGER.
         On entry, LDA specifies the first dimension of A as
         declared in the calling (sub) program. LDA must be at
         least ( k + 1 ).  Unchanged on exit.

X        - DOUBLE PRECISION array of DIMENSION at least
         ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
         incremented array X must contain the vector x.
         Unchanged on exit.

INCX     - INTEGER.
         On entry, INCX specifies the increment for the ele-
         ments of X. INCX must not be zero.  Unchanged on
         exit.

BETA     - DOUBLE PRECISION.
         On entry, BETA specifies the scalar beta.  Unchanged
         on exit.

Y        - DOUBLE PRECISION array of DIMENSION at least
         ( 1 + ( n - 1 )*abs( INCY ) ).  Before entry, the
         incremented array Y must contain the vector y. On
         exit, Y is overwritten by the updated vector y.

INCY     - INTEGER.
         On entry, INCY specifies the increment for the ele-
         ments of Y. INCY must not be zero.  Unchanged on
         exit.
```

⟨*BLAS 2 dsbmv*⟩≡
```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dsbmv (uplo n k alpha a lda x incx beta y incy)
    (declare (type (simple-array double-float (*)) y x a)
             (type (double-float) beta alpha)
             (type fixnum incy incx lda k n)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (a double-float a-%data% a-%offset%)
         (x double-float x-%data% x-%offset%)
         (y double-float y-%data% y-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (kplus1 0) (kx 0)
             (ky 0) (l 0) (temp1 0.0) (temp2 0.0))
        (declare (type fixnum i info ix iy j jx jy kplus1 kx ky l)
                 (type (double-float) temp1 temp2))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((< k 0)
           (setf info 3))
          ((< lda (f2cl-lib:int-add k 1))
           (setf info 6))
          ((= incx 0)
           (setf info 8))
          ((= incy 0)
           (setf info 11)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DSBMV" info)
           (go end_label)))
        (if (or (= n 0) (and (= alpha zero) (= beta one))) (go end_label))
        (cond
          ((> incx 0)
           (setf kx 1))
          (t
           (setf kx
                   (f2cl-lib:int-sub 1
                                     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
```

```
                                                    incx)))))
          (cond
            ((> incy 0)
             (setf ky 1))
            (t
             (setf ky
                     (f2cl-lib:int-sub 1
                                       (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                         incy)))))
          (cond
            ((/= beta one)
             (cond
               ((= incy 1)
                (cond
                  ((= beta zero)
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                               zero))))
                  (t
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                               (* beta
                                  (f2cl-lib:fref y-%data%
                                                 (i)
                                                 ((1 *))
                                                 y-%offset%)))))))))
               (t
                (setf iy ky)
                (cond
                  ((= beta zero)
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                               zero)
                       (setf iy (f2cl-lib:int-add iy incy)))))
                  (t
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                               (* beta
```

```
                          (f2cl-lib:fref y-%data%
                                         (iy)
                                         ((1 *))
                                         y-%offset%)))
              (setf iy (f2cl-lib:int-add iy incy))))))))))
(if (= alpha zero) (go end_label))
(cond
  ((char-equal uplo #\U)
   (setf kplus1 (f2cl-lib:int-add k 1))
   (cond
     ((and (= incx 1) (= incy 1))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (setf temp1
                  (* alpha
                     (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
          (setf temp2 zero)
          (setf l (f2cl-lib:int-sub kplus1 j))
          (f2cl-lib:fdo (i
                         (max (the fixnum 1)
                              (the fixnum
                                   (f2cl-lib:int-add j
                                                     (f2cl-lib:int-sub
                                                      k))))
                         (f2cl-lib:int-add i 1))
                        ((> i
                            (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                         nil)
            (tagbody
              (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                      (+
                       (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                       (* temp1
                          (f2cl-lib:fref a-%data%
                                         ((f2cl-lib:int-add l i) j)
                                         ((1 lda) (1 *))
                                         a-%offset%))))
              (setf temp2
                      (+ temp2
                         (*
                          (f2cl-lib:fref a-%data%
                                         ((f2cl-lib:int-add l i) j)
                                         ((1 lda) (1 *))
                                         a-%offset%)
                          (f2cl-lib:fref x-%data%
```

```
                                        (i)
                                        ((1 *))
                                        x-%offset%))))))
          (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                  (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                     (* temp1
                        (f2cl-lib:fref a-%data%
                                       (kplus1 j)
                                       ((1 lda) (1 *))
                                       a-%offset%))
                     (* alpha temp2))))))
      (t
       (setf jx kx)
       (setf jy ky)
       (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                     ((> j n) nil)
         (tagbody
           (setf temp1
                   (* alpha
                      (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
           (setf temp2 zero)
           (setf ix kx)
           (setf iy ky)
           (setf l (f2cl-lib:int-sub kplus1 j))
           (f2cl-lib:fdo (i
                          (max (the fixnum 1)
                               (the fixnum
                                    (f2cl-lib:int-add j
                                                      (f2cl-lib:int-sub
                                                       k))))
                          (f2cl-lib:int-add i 1))
                         ((> i
                             (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                          nil)
             (tagbody
               (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                       (+
                        (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                        (* temp1
                           (f2cl-lib:fref a-%data%
                                          ((f2cl-lib:int-add l i) j)
                                          ((1 lda) (1 *))
                                          a-%offset%))))
               (setf temp2
                       (+ temp2
                          (*
```

```
                                (f2cl-lib:fref a-%data%
                                               ((f2cl-lib:int-add l i) j)
                                               ((1 lda) (1 *))
                                               a-%offset%)
                                (f2cl-lib:fref x-%data%
                                               (ix)
                                               ((1 *))
                                               x-%offset%))))
                    (setf ix (f2cl-lib:int-add ix incx))
                    (setf iy (f2cl-lib:int-add iy incy))))
                  (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                        (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                           (* temp1
                              (f2cl-lib:fref a-%data%
                                             (kplus1 j)
                                             ((1 lda) (1 *))
                                             a-%offset%))
                           (* alpha temp2)))
                  (setf jx (f2cl-lib:int-add jx incx))
                  (setf jy (f2cl-lib:int-add jy incy))
                  (cond
                    ((> j k)
                     (setf kx (f2cl-lib:int-add kx incx))
                     (setf ky (f2cl-lib:int-add ky incy)))))))))
          (t
           (cond
             ((and (= incx 1) (= incy 1))
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (setf temp1
                        (* alpha
                           (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
                  (setf temp2 zero)
                  (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                        (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                           (* temp1
                              (f2cl-lib:fref a-%data%
                                             (1 j)
                                             ((1 lda) (1 *))
                                             a-%offset%))))
                  (setf l (f2cl-lib:int-sub 1 j))
                  (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                   (f2cl-lib:int-add i 1))
                                ((> i
                                    (min (the fixnum n)
```

```
                                    (the fixnum
                                         (f2cl-lib:int-add j k))))
                              nil)
                    (tagbody
                      (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                              (+
                               (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                               (* temp1
                                  (f2cl-lib:fref a-%data%
                                                 ((f2cl-lib:int-add l i) j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))))
                      (setf temp2
                              (+ temp2
                                 (*
                                  (f2cl-lib:fref a-%data%
                                                 ((f2cl-lib:int-add l i) j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%)
                                  (f2cl-lib:fref x-%data%
                                                 (i)
                                                 ((1 *))
                                                 x-%offset%))))))
                  (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                          (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                             (* alpha temp2))))))
          (t
           (setf jx kx)
           (setf jy ky)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf temp1
                       (* alpha
                          (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
               (setf temp2 zero)
               (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                       (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                          (* temp1
                             (f2cl-lib:fref a-%data%
                                            (1 j)
                                            ((1 lda) (1 *))
                                            a-%offset%))))
               (setf l (f2cl-lib:int-sub 1 j))
               (setf ix jx)
               (setf iy jy)
```

```
                    (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                   (f2cl-lib:int-add i 1))
                                  ((> i
                                      (min (the fixnum n)
                                           (the fixnum
                                                (f2cl-lib:int-add j k))))
                                   nil)
                      (tagbody
                        (setf ix (f2cl-lib:int-add ix incx))
                        (setf iy (f2cl-lib:int-add iy incy))
                        (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                                (+
                                 (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                                 (* temp1
                                    (f2cl-lib:fref a-%data%
                                                   ((f2cl-lib:int-add l i) j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%))))
                        (setf temp2
                                (+ temp2
                                   (*
                                    (f2cl-lib:fref a-%data%
                                                   ((f2cl-lib:int-add l i) j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%)
                                    (f2cl-lib:fref x-%data%
                                                   (ix)
                                                   ((1 *))
                                                   x-%offset%))))))
                    (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                            (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                               (* alpha temp2)))
                    (setf jx (f2cl-lib:int-add jx incx))
                    (setf jy (f2cl-lib:int-add jy incy))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil nil nil nil))))))
```

5.5 dspmv BLAS

⟨*dspmv.input*⟩≡

```
)set break resume
)sys rm -f dspmv.output
)spool dspmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dspmv.help*⟩≡

```
  ======================================================================
  dspmv examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        DSPMV - perform the matrix-vector operation   y := alpha*A*x
        + beta*y,

   SYNOPSIS
        SUBROUTINE DSPMV ( UPLO, N, ALPHA, AP, X, INCX, BETA, Y,
                           INCY )

            DOUBLE       PRECISION ALPHA, BETA

            INTEGER      INCX, INCY, N

            CHARACTER*1  UPLO

            DOUBLE       PRECISION AP( * ), X( * ), Y( * )

   PURPOSE
        DSPMV  performs the matrix-vector operation

        where alpha and beta are scalars, x and y are n element vec-
        tors and A is an n by n symmetric matrix, supplied in packed
        form.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On entry, UPLO specifies whether the upper or lower
                 triangular part of the matrix A is supplied in the
                 packed array AP as follows:

                 UPLO = 'U' or 'u'   The upper triangular part of A is
                 supplied in AP.

                 UPLO = 'L' or 'l'   The lower triangular part of A is
                 supplied in AP.

                 Unchanged on exit.
```

```
N      - INTEGER.
       On entry, N specifies the order of the matrix A.  N
       must be at least zero.  Unchanged on exit.

ALPHA  - DOUBLE PRECISION.
       On entry, ALPHA specifies the scalar alpha.
       Unchanged on exit.

AP     - DOUBLE PRECISION array of DIMENSION at least
       ( ( n*( n + 1 ) )/2 ).  Before entry with UPLO = 'U'
       or 'u', the array AP must contain the upper triangu-
       lar part of the symmetric matrix packed sequentially,
       column by column, so that AP( 1 ) contains a( 1, 1 ),
       AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 )

       respectively, and so on.  Before entry with UPLO =
       'L' or 'l', the array AP must contain the lower tri-
       angular part of the symmetric matrix packed sequen-
       tially, column by column, so that AP( 1 ) contains a(
       1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1 ) and a(
       3, 1 ) respectively, and so on.  Unchanged on exit.

X      - DOUBLE PRECISION array of dimension at least
       ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
       incremented array X must contain the n element vector
       x.  Unchanged on exit.

INCX   - INTEGER.
       On entry, INCX specifies the increment for the ele-
       ments of X. INCX must not be zero.  Unchanged on
       exit.

BETA   - DOUBLE PRECISION.
       On entry, BETA specifies the scalar beta. When BETA
       is supplied as zero then Y need not be set on input.
       Unchanged on exit.

Y      - DOUBLE PRECISION array of dimension at least
       ( 1 + ( n - 1 )*abs( INCY ) ).  Before entry, the
       incremented array Y must contain the n element vector
       y. On exit, Y is overwritten by the updated vector y.

INCY   - INTEGER.
       On entry, INCY specifies the increment for the ele-
       ments of Y. INCY must not be zero.  Unchanged on
       exit.
```

⟨*BLAS 2 dspmv*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dspmv (uplo n alpha ap x incx beta y incy)
    (declare (type (simple-array double-float (*)) y x ap)
             (type (double-float) beta alpha)
             (type fixnum incy incx n)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (ap double-float ap-%data% ap-%offset%)
         (x double-float x-%data% x-%offset%)
         (y double-float y-%data% y-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (k 0) (kk 0)
             (kx 0) (ky 0) (temp1 0.0) (temp2 0.0))
        (declare (type fixnum i info ix iy j jx jy k kk kx ky)
                 (type (double-float) temp1 temp2))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((= incx 0)
           (setf info 6))
          ((= incy 0)
           (setf info 9)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DSPMV" info)
           (go end_label)))
        (if (or (= n 0) (and (= alpha zero) (= beta one))) (go end_label))
        (cond
          ((> incx 0)
           (setf kx 1))
          (t
           (setf kx
                   (f2cl-lib:int-sub 1
                                     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                       incx)))))
        (cond
          ((> incy 0)
           (setf ky 1))
```

```
            (t
             (setf ky
                     (f2cl-lib:int-sub 1
                                       (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                         incy)))))
          (cond
            ((/= beta one)
             (cond
               ((= incy 1)
                (cond
                  ((= beta zero)
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                               zero))))
                  (t
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                               (* beta
                                  (f2cl-lib:fref y-%data%
                                                 (i)
                                                 ((1 *))
                                                 y-%offset%)))))))))
               (t
                (setf iy ky)
                (cond
                  ((= beta zero)
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                               zero)
                       (setf iy (f2cl-lib:int-add iy incy)))))
                  (t
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                               (* beta
                                  (f2cl-lib:fref y-%data%
                                                 (iy)
                                                 ((1 *))
                                                 y-%offset%)))
```

```
              (setf iy (f2cl-lib:int-add iy incy)))))))))))
(if (= alpha zero) (go end_label))
(setf kk 1)
(cond
  ((char-equal uplo #\U)
   (cond
     ((and (= incx 1) (= incy 1))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (setf temp1
                  (* alpha
                     (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
          (setf temp2 zero)
          (setf k kk)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i
                            (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                         nil)
            (tagbody
              (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                      (+
                       (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                       (* temp1
                          (f2cl-lib:fref ap-%data%
                                         (k)
                                         ((1 *))
                                         ap-%offset%))))
              (setf temp2
                      (+ temp2
                         (*
                          (f2cl-lib:fref ap-%data%
                                         (k)
                                         ((1 *))
                                         ap-%offset%)
                          (f2cl-lib:fref x-%data%
                                         (i)
                                         ((1 *))
                                         x-%offset%))))
              (setf k (f2cl-lib:int-add k 1))))
          (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                  (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                     (* temp1
                        (f2cl-lib:fref ap-%data%
                                       ((f2cl-lib:int-sub
                                         (f2cl-lib:int-add kk j)
```

```
                                        1))
                                      ((1 *))
                                      ap-%offset%))
                    (* alpha temp2)))
        (setf kk (f2cl-lib:int-add kk j)))))
   (t
    (setf jx kx)
    (setf jy ky)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  ((> j n) nil)
      (tagbody
        (setf temp1
                (* alpha
                   (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
        (setf temp2 zero)
        (setf ix kx)
        (setf iy ky)
        (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
                      ((> k
                          (f2cl-lib:int-add kk
                                            j
                                            (f2cl-lib:int-sub 2)))
                       nil)
          (tagbody
            (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                    (+
                     (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                     (* temp1
                        (f2cl-lib:fref ap-%data%
                                       (k)
                                       ((1 *))
                                       ap-%offset%))))
            (setf temp2
                    (+ temp2
                       (*
                        (f2cl-lib:fref ap-%data%
                                       (k)
                                       ((1 *))
                                       ap-%offset%)
                        (f2cl-lib:fref x-%data%
                                       (ix)
                                       ((1 *))
                                       x-%offset%))))
            (setf ix (f2cl-lib:int-add ix incx))
            (setf iy (f2cl-lib:int-add iy incy))))
        (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
```

```
                        (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                           (* temp1
                              (f2cl-lib:fref ap-%data%
                                             ((f2cl-lib:int-sub
                                               (f2cl-lib:int-add kk j)
                                               1))
                                             ((1 *))
                                             ap-%offset%))
                           (* alpha temp2)))
                  (setf jx (f2cl-lib:int-add jx incx))
                  (setf jy (f2cl-lib:int-add jy incy))
                  (setf kk (f2cl-lib:int-add kk j)))))))
          (t
           (cond
             ((and (= incx 1) (= incy 1))
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (setf temp1
                          (* alpha
                             (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
                  (setf temp2 zero)
                  (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                          (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                             (* temp1
                                (f2cl-lib:fref ap-%data%
                                               (kk)
                                               ((1 *))
                                               ap-%offset%))))
                  (setf k (f2cl-lib:int-add kk 1))
                  (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                   (f2cl-lib:int-add i 1))
                                ((> i n) nil)
                    (tagbody
                      (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                              (+
                               (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                               (* temp1
                                  (f2cl-lib:fref ap-%data%
                                                 (k)
                                                 ((1 *))
                                                 ap-%offset%))))
                      (setf temp2
                              (+ temp2
                                 (*
                                  (f2cl-lib:fref ap-%data%
```

```
                                        (k)
                                        ((1 *))
                                        ap-%offset%)
                          (f2cl-lib:fref x-%data%
                                        (i)
                                        ((1 *))
                                        x-%offset%))))
              (setf k (f2cl-lib:int-add k 1))))
          (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                  (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                     (* alpha temp2)))
          (setf kk
                  (f2cl-lib:int-add kk
                                    (f2cl-lib:int-add
                                     (f2cl-lib:int-sub n j)
                                     1))))))
     (t
      (setf jx kx)
      (setf jy ky)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (setf temp1
                  (* alpha
                     (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
          (setf temp2 zero)
          (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                  (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                     (* temp1
                        (f2cl-lib:fref ap-%data%
                                       (kk)
                                       ((1 *))
                                       ap-%offset%))))
          (setf ix jx)
          (setf iy jy)
          (f2cl-lib:fdo (k (f2cl-lib:int-add kk 1)
                           (f2cl-lib:int-add k 1))
                        ((> k
                            (f2cl-lib:int-add kk
                                              n
                                              (f2cl-lib:int-sub j)))
                         nil)
            (tagbody
              (setf ix (f2cl-lib:int-add ix incx))
              (setf iy (f2cl-lib:int-add iy incy))
              (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
```

```
                                (+
                                 (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                                 (* temp1
                                    (f2cl-lib:fref ap-%data%
                                                   (k)
                                                   ((1 *))
                                                   ap-%offset%))))
                            (setf temp2
                                    (+ temp2
                                       (*
                                        (f2cl-lib:fref ap-%data%
                                                       (k)
                                                       ((1 *))
                                                       ap-%offset%)
                                        (f2cl-lib:fref x-%data%
                                                       (ix)
                                                       ((1 *))
                                                       x-%offset%))))))
                          (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                  (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                     (* alpha temp2)))
                          (setf jx (f2cl-lib:int-add jx incx))
                          (setf jy (f2cl-lib:int-add jy incy))
                          (setf kk
                                  (f2cl-lib:int-add kk
                                                    (f2cl-lib:int-add
                                                     (f2cl-lib:int-sub n j)
                                                     1)))))))))
   end_label
          (return (values nil nil nil nil nil nil nil nil nil))))))
```

5.6 dspr2 BLAS

⟨*dspr2.input*⟩≡

```
  )set break resume
  )sys rm -f dspr2.output
  )spool dspr2.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dspr2.help*⟩≡

```
  ======================================================================
  dspr2 examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        DSPR2 - perform the symmetric rank 2 operation   A :=
        alpha*x*y' + alpha*y*x' + A,

   SYNOPSIS
        SUBROUTINE DSPR2 ( UPLO, N, ALPHA, X, INCX, Y, INCY, AP )

            DOUBLE       PRECISION ALPHA

            INTEGER      INCX, INCY, N

            CHARACTER*1  UPLO

            DOUBLE       PRECISION AP( * ), X( * ), Y( * )

   PURPOSE
        DSPR2  performs the symmetric rank 2 operation

        where alpha is a scalar, x and y are n element vectors and A
        is an n by n symmetric matrix, supplied in packed form.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On entry, UPLO specifies whether the upper or lower
                 triangular part of the matrix A is supplied in the
                 packed array AP as follows:

                 UPLO = 'U' or 'u'   The upper triangular part of A is
                 supplied in AP.

                 UPLO = 'L' or 'l'   The lower triangular part of A is
                 supplied in AP.

                 Unchanged on exit.

        N      - INTEGER.
                 On entry, N specifies the order of the matrix A.  N
```

```
       must be at least zero.  Unchanged on exit.

ALPHA  - DOUBLE PRECISION.
       On entry, ALPHA specifies the scalar alpha.
       Unchanged on exit.

X      - DOUBLE PRECISION array of dimension at least
       ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
       incremented array X must contain the n element vector
       x.  Unchanged on exit.

INCX   - INTEGER.
       On entry, INCX specifies the increment for the ele-
       ments of X. INCX must not be zero.  Unchanged on

       exit.

Y      - DOUBLE PRECISION array of dimension at least
       ( 1 + ( n - 1 )*abs( INCY ) ).  Before entry, the
       incremented array Y must contain the n element vector
       y.  Unchanged on exit.

INCY   - INTEGER.
       On entry, INCY specifies the increment for the ele-
       ments of Y. INCY must not be zero.  Unchanged on
       exit.

AP     - DOUBLE PRECISION array of DIMENSION at least
       ( ( n*( n + 1 ) )/2 ).  Before entry with  UPLO = 'U'
       or 'u', the array AP must contain the upper triangu-
       lar part of the symmetric matrix packed sequentially,
       column by column, so that AP( 1 ) contains a( 1, 1 ),
       AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 )
       respectively, and so on. On exit, the array AP is
       overwritten by the upper triangular part of the
       updated matrix.  Before entry with UPLO = 'L' or 'l',
       the array AP must contain the lower triangular part
       of the symmetric matrix packed sequentially, column
       by column, so that AP( 1 ) contains a( 1, 1 ), AP( 2
       ) and AP( 3 ) contain a( 2, 1 ) and a( 3, 1 ) respec-
       tively, and so on. On exit, the array AP is overwrit-
       ten by the lower triangular part of the updated
       matrix.
```

⟨*BLAS 2 dspr2*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dspr2 (uplo n alpha x incx y incy ap)
    (declare (type (simple-array double-float (*)) ap y x)
             (type (double-float) alpha)
             (type fixnum incy incx n)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (x double-float x-%data% x-%offset%)
         (y double-float y-%data% y-%offset%)
         (ap double-float ap-%data% ap-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (k 0) (kk 0)
             (kx 0) (ky 0) (temp1 0.0) (temp2 0.0))
        (declare (type fixnum i info ix iy j jx jy k kk kx ky)
                 (type (double-float) temp1 temp2))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((= incx 0)
           (setf info 5))
          ((= incy 0)
           (setf info 7)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DSPR2" info)
           (go end_label)))
        (if (or (= n 0) (= alpha zero)) (go end_label))
        (cond
          ((or (/= incx 1) (/= incy 1))
           (cond
             ((> incx 0)
              (setf kx 1))
             (t
              (setf kx
                      (f2cl-lib:int-sub 1
                                        (f2cl-lib:int-mul
                                         (f2cl-lib:int-sub n 1)
                                         incx)))))
           (cond
```

```
      ((> incy 0)
       (setf ky 1))
      (t
       (setf ky
             (f2cl-lib:int-sub 1
                               (f2cl-lib:int-mul
                                (f2cl-lib:int-sub n 1)
                                incy)))))
   (setf jx kx)
   (setf jy ky)))
(setf kk 1)
(cond
  ((char-equal uplo #\U)
   (cond
     ((and (= incx 1) (= incy 1))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (cond
            ((or (/= (f2cl-lib:fref x (j) ((1 *))) zero)
                 (/= (f2cl-lib:fref y (j) ((1 *))) zero))
             (setf temp1
                     (* alpha
                        (f2cl-lib:fref y-%data%
                                       (j)
                                       ((1 *))
                                       y-%offset%)))
             (setf temp2
                     (* alpha
                        (f2cl-lib:fref x-%data%
                                       (j)
                                       ((1 *))
                                       x-%offset%)))
             (setf k kk)
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i j) nil)
               (tagbody
                 (setf (f2cl-lib:fref ap-%data%
                                      (k)
                                      ((1 *))
                                      ap-%offset%)
                       (+
                        (f2cl-lib:fref ap-%data%
                                       (k)
                                       ((1 *))
                                       ap-%offset%)
```

```
                              (*
                               (f2cl-lib:fref x-%data%
                                              (i)
                                              ((1 *))
                                              x-%offset%)
                               temp1)
                              (*
                               (f2cl-lib:fref y-%data%
                                              (i)
                                              ((1 *))
                                              y-%offset%)
                               temp2)))
                    (setf k (f2cl-lib:int-add k 1))))))
           (setf kk (f2cl-lib:int-add kk j)))))
      (t
       (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                     ((> j n) nil)
         (tagbody
           (cond
             ((or (/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                  (/= (f2cl-lib:fref y (jy) ((1 *))) zero))
              (setf temp1
                      (* alpha
                         (f2cl-lib:fref y-%data%
                                        (jy)
                                        ((1 *))
                                        y-%offset%)))
              (setf temp2
                      (* alpha
                         (f2cl-lib:fref x-%data%
                                        (jx)
                                        ((1 *))
                                        x-%offset%)))
              (setf ix kx)
              (setf iy ky)
              (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
                            ((> k
                                (f2cl-lib:int-add kk
                                                  j
                                                  (f2cl-lib:int-sub 1)))
                             nil)
                (tagbody
                  (setf (f2cl-lib:fref ap-%data%
                                       (k)
                                       ((1 *))
                                       ap-%offset%)
```

```
                                (+
                                 (f2cl-lib:fref ap-%data%
                                                (k)
                                                ((1 *))
                                                ap-%offset%)
                                 (*
                                  (f2cl-lib:fref x-%data%
                                                 (ix)
                                                 ((1 *))
                                                 x-%offset%)
                                  temp1)
                                 (*
                                  (f2cl-lib:fref y-%data%
                                                 (iy)
                                                 ((1 *))
                                                 y-%offset%)
                                  temp2)))
                           (setf ix (f2cl-lib:int-add ix incx))
                           (setf iy (f2cl-lib:int-add iy incy))))))
                   (setf jx (f2cl-lib:int-add jx incx))
                   (setf jy (f2cl-lib:int-add jy incy))
                   (setf kk (f2cl-lib:int-add kk j)))))))
          (t
           (cond
             ((and (= incx 1) (= incy 1))
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (cond
                    ((or (/= (f2cl-lib:fref x (j) ((1 *))) zero)
                         (/= (f2cl-lib:fref y (j) ((1 *))) zero))
                     (setf temp1
                             (* alpha
                                (f2cl-lib:fref y-%data%
                                               (j)
                                               ((1 *))
                                               y-%offset%)))
                     (setf temp2
                             (* alpha
                                (f2cl-lib:fref x-%data%
                                               (j)
                                               ((1 *))
                                               x-%offset%)))
                     (setf k kk)
                     (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                   ((> i n) nil)
```

```
                (tagbody
                  (setf (f2cl-lib:fref ap-%data%
                                       (k)
                                       ((1 *))
                                       ap-%offset%)
                        (+
                         (f2cl-lib:fref ap-%data%
                                        (k)
                                        ((1 *))
                                        ap-%offset%)
                         (*
                          (f2cl-lib:fref x-%data%
                                         (i)
                                         ((1 *))
                                         x-%offset%)
                          temp1)
                         (*
                          (f2cl-lib:fref y-%data%
                                         (i)
                                         ((1 *))
                                         y-%offset%)
                          temp2)))
                  (setf k (f2cl-lib:int-add k 1))))))
     (setf kk
             (f2cl-lib:int-add
              (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
              1)))))
(t
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j n) nil)
   (tagbody
     (cond
       ((or (/= (f2cl-lib:fref x (jx) ((1 *))) zero)
            (/= (f2cl-lib:fref y (jy) ((1 *))) zero))
        (setf temp1
                (* alpha
                   (f2cl-lib:fref y-%data%
                                  (jy)
                                  ((1 *))
                                  y-%offset%)))
        (setf temp2
                (* alpha
                   (f2cl-lib:fref x-%data%
                                  (jx)
                                  ((1 *))
                                  x-%offset%)))
```

```
                        (setf ix jx)
                        (setf iy jy)
                        (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
                                      ((> k
                                          (f2cl-lib:int-add kk
                                                            n
                                                            (f2cl-lib:int-sub j)))
                                       nil)
                          (tagbody
                            (setf (f2cl-lib:fref ap-%data%
                                                 (k)
                                                 ((1 *))
                                                 ap-%offset%)
                                  (+
                                   (f2cl-lib:fref ap-%data%
                                                  (k)
                                                  ((1 *))
                                                  ap-%offset%)
                                   (*
                                    (f2cl-lib:fref x-%data%
                                                   (ix)
                                                   ((1 *))
                                                   x-%offset%)
                                    temp1)
                                   (*
                                    (f2cl-lib:fref y-%data%
                                                   (iy)
                                                   ((1 *))
                                                   y-%offset%)
                                    temp2)))
                            (setf ix (f2cl-lib:int-add ix incx))
                            (setf iy (f2cl-lib:int-add iy incy))))))
                      (setf jx (f2cl-lib:int-add jx incx))
                      (setf jy (f2cl-lib:int-add jy incy))
                      (setf kk
                            (f2cl-lib:int-add
                             (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
                             1))))))
  end_label
       (return (values nil nil nil nil nil nil nil nil))))))
```

5.7 dspr BLAS

⟨*dspr.input*⟩≡

```
)set break resume
)sys rm -f dspr.output
)spool dspr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dspr.help*⟩≡

```
 ========================================================================
 dspr examples
 ========================================================================

 ========================================================================
 Man Page Details
 ========================================================================

  NAME
       DSPR - perform the symmetric rank 1 operation   A :=
       alpha*x*x' + A,

  SYNOPSIS
       SUBROUTINE DSPR ( UPLO, N, ALPHA, X, INCX, AP )

           DOUBLE      PRECISION ALPHA

           INTEGER     INCX, N

           CHARACTER*1 UPLO

           DOUBLE      PRECISION AP( * ), X( * )

  PURPOSE
       DSPR    performs the symmetric rank 1 operation

       where alpha is a real scalar, x is an n element vector and A
       is an n by n symmetric matrix, supplied in packed form.

  PARAMETERS
       UPLO   - CHARACTER*1.
              On entry, UPLO specifies whether the upper or lower
              triangular part of the matrix A is supplied in the
              packed array AP as follows:

              UPLO = 'U' or 'u'   The upper triangular part of A is
              supplied in AP.

              UPLO = 'L' or 'l'   The lower triangular part of A is
              supplied in AP.

              Unchanged on exit.

       N      - INTEGER.
              On entry, N specifies the order of the matrix A.  N
```

```
       must be at least zero.  Unchanged on exit.

ALPHA  - DOUBLE PRECISION.
       On entry, ALPHA specifies the scalar alpha.
       Unchanged on exit.

X      - DOUBLE PRECISION array of dimension at least
       ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
       incremented array X must contain the n element vector
       x.  Unchanged on exit.

INCX   - INTEGER.
       On entry, INCX specifies the increment for the ele-
       ments of X. INCX must not be zero.  Unchanged on

       exit.

AP     - DOUBLE PRECISION array of DIMENSION at least
       ( ( n*( n + 1 ) )/2 ).  Before entry with  UPLO = 'U'
       or 'u', the array AP must contain the upper triangu-
       lar part of the symmetric matrix packed sequentially,
       column by column, so that AP( 1 ) contains a( 1, 1 ),
       AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 )
       respectively, and so on. On exit, the array AP is
       overwritten by the upper triangular part of the
       updated matrix.  Before entry with UPLO = 'L' or 'l',
       the array AP must contain the lower triangular part
       of the symmetric matrix packed sequentially, column
       by column, so that AP( 1 ) contains a( 1, 1 ), AP( 2
       ) and AP( 3 ) contain a( 2, 1 ) and a( 3, 1 ) respec-
       tively, and so on. On exit, the array AP is overwrit-
       ten by the lower triangular part of the updated
       matrix.
```

⟨*BLAS 2 dspr*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dspr (uplo n alpha x incx ap)
    (declare (type (simple-array double-float (*)) ap x)
             (type (double-float) alpha)
             (type fixnum incx n)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (x double-float x-%data% x-%offset%)
         (ap double-float ap-%data% ap-%offset%))
      (prog ((i 0) (info 0) (ix 0) (j 0) (jx 0) (k 0) (kk 0) (kx 0) (temp 0.0))
        (declare (type fixnum i info ix j jx k kk kx)
                 (type (double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((= incx 0)
           (setf info 5)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
             "DSPR" info)
           (go end_label)))
        (if (or (= n 0) (= alpha zero)) (go end_label))
        (cond
          ((<= incx 0)
           (setf kx
                   (f2cl-lib:int-sub 1
                                     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                       incx))))
          ((/= incx 1)
           (setf kx 1)))
        (setf kk 1)
        (cond
          ((char-equal uplo #\U)
           (cond
             ((= incx 1)
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
```

```
          (cond
            ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
             (setf temp
                     (* alpha
                        (f2cl-lib:fref x-%data%
                                       (j)
                                       ((1 *))
                                       x-%offset%)))
             (setf k kk)
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i j) nil)
               (tagbody
                 (setf (f2cl-lib:fref ap-%data%
                                      (k)
                                      ((1 *))
                                      ap-%offset%)
                       (+
                        (f2cl-lib:fref ap-%data%
                                       (k)
                                       ((1 *))
                                       ap-%offset%)
                        (*
                         (f2cl-lib:fref x-%data%
                                        (i)
                                        ((1 *))
                                        x-%offset%)
                         temp)))
                 (setf k (f2cl-lib:int-add k 1))))))
          (setf kk (f2cl-lib:int-add kk j)))))
     (t
      (setf jx kx)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (cond
            ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
             (setf temp
                     (* alpha
                        (f2cl-lib:fref x-%data%
                                       (jx)
                                       ((1 *))
                                       x-%offset%)))
             (setf ix kx)
             (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
                           ((> k
                               (f2cl-lib:int-add kk
```

```
                                                    j
                                                    (f2cl-lib:int-sub 1)))
                                  nil)
                        (tagbody
                          (setf (f2cl-lib:fref ap-%data%
                                               (k)
                                               ((1 *))
                                               ap-%offset%)
                                (+
                                 (f2cl-lib:fref ap-%data%
                                                (k)
                                                ((1 *))
                                                ap-%offset%)
                                 (*
                                  (f2cl-lib:fref x-%data%
                                                 (ix)
                                                 ((1 *))
                                                 x-%offset%)
                                  temp)))
                          (setf ix (f2cl-lib:int-add ix incx))))))
                (setf jx (f2cl-lib:int-add jx incx))
                (setf kk (f2cl-lib:int-add kk j)))))))
        (t
         (cond
           ((= incx 1)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                          ((> j n) nil)
              (tagbody
                (cond
                  ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                   (setf temp
                           (* alpha
                              (f2cl-lib:fref x-%data%
                                             (j)
                                             ((1 *))
                                             x-%offset%)))
                   (setf k kk)
                   (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref ap-%data%
                                            (k)
                                            ((1 *))
                                            ap-%offset%)
                             (+
                              (f2cl-lib:fref ap-%data%
```

```
                                        (k)
                                        ((1 *))
                                        ap-%offset%)
                                (*
                                 (f2cl-lib:fref x-%data%
                                                (i)
                                                ((1 *))
                                                x-%offset%)
                                 temp)))
                      (setf k (f2cl-lib:int-add k 1))))))
             (setf kk
                    (f2cl-lib:int-add
                     (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
                     1)))))
        (t
         (setf jx kx)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (cond
               ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                (setf temp
                        (* alpha
                           (f2cl-lib:fref x-%data%
                                          (jx)
                                          ((1 *))
                                          x-%offset%)))
                (setf ix jx)
                (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
                              ((> k
                                  (f2cl-lib:int-add kk
                                                    n
                                                    (f2cl-lib:int-sub j)))
                               nil)
                  (tagbody
                    (setf (f2cl-lib:fref ap-%data%
                                         (k)
                                         ((1 *))
                                         ap-%offset%)
                          (+
                           (f2cl-lib:fref ap-%data%
                                          (k)
                                          ((1 *))
                                          ap-%offset%)
                           (*
                            (f2cl-lib:fref x-%data%
```

```
                                                    (ix)
                                                    ((1 *))
                                                    x-%offset%)
                                        temp)))
                               (setf ix (f2cl-lib:int-add ix incx))))))
                        (setf jx (f2cl-lib:int-add jx incx))
                        (setf kk
                                (f2cl-lib:int-add
                                 (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
                                 1))))))))
   end_label
          (return (values nil nil nil nil nil nil))))))
```

5.8 dsymv BLAS

⟨*dsymv.input*⟩≡

```
  )set break resume
  )sys rm -f dsymv.output
  )spool dsymv.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dsymv.help*⟩≡

```
  ======================================================================
  dsymv examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        DSYMV - perform the matrix-vector operation   y := alpha*A*x
        + beta*y,

   SYNOPSIS
        SUBROUTINE DSYMV ( UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y,
                           INCY )

            DOUBLE       PRECISION ALPHA, BETA

            INTEGER      INCX, INCY, LDA, N

            CHARACTER*1  UPLO

            DOUBLE       PRECISION A( LDA, * ), X( * ), Y( * )

   PURPOSE
        DSYMV  performs the matrix-vector  operation

        where alpha and beta are scalars, x and y are n element vec-
        tors and A is an n by n symmetric matrix.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On entry, UPLO specifies whether the upper or lower
                 triangular part of the array A is to be referenced as
                 follows:

                 UPLO = 'U' or 'u'   Only the upper triangular part of
                 A is to be referenced.

                 UPLO = 'L' or 'l'   Only the lower triangular part of
                 A is to be referenced.

                 Unchanged on exit.

        N      - INTEGER.
```

```
          On entry, N specifies the order of the matrix A.  N
          must be at least zero.  Unchanged on exit.

ALPHA     - DOUBLE PRECISION.
          On entry, ALPHA specifies the scalar alpha.
          Unchanged on exit.

A         - DOUBLE PRECISION array of DIMENSION ( LDA, n ).
          Before entry with  UPLO = 'U' or 'u', the leading n
          by n upper triangular part of the array A must con-
          tain the upper triangular part of the symmetric
          matrix and the strictly lower triangular part of A is
          not referenced.  Before entry with UPLO = 'L' or 'l',
          the leading n by n lower triangular part of the array

          A must contain the lower triangular part of the sym-
          metric matrix and the strictly upper triangular part
          of A is not referenced.  Unchanged on exit.

LDA       - INTEGER.
          On entry, LDA specifies the first dimension of A as
          declared in the calling (sub) program. LDA must be at
          least max( 1, n ).  Unchanged on exit.

X         - DOUBLE PRECISION array of dimension at least
          ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
          incremented array X must contain the n element vector
          x.  Unchanged on exit.

INCX      - INTEGER.
          On entry, INCX specifies the increment for the ele-
          ments of X. INCX must not be zero.  Unchanged on
          exit.

BETA      - DOUBLE PRECISION.
          On entry, BETA specifies the scalar beta. When BETA
          is supplied as zero then Y need not be set on input.
          Unchanged on exit.

Y         - DOUBLE PRECISION array of dimension at least
          ( 1 + ( n - 1 )*abs( INCY ) ).  Before entry, the
          incremented array Y must contain the n element vector
          y. On exit, Y is overwritten by the updated vector y.

INCY      - INTEGER.
          On entry, INCY specifies the increment for the ele-
```

```
ments of Y. INCY must not be zero.  Unchanged on
exit.
```

⟨*BLAS 2 dsymv*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dsymv (uplo n alpha a lda x incx beta y incy)
    (declare (type (simple-array double-float (*)) y x a)
             (type (double-float) beta alpha)
             (type fixnum incy incx lda n)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (a double-float a-%data% a-%offset%)
         (x double-float x-%data% x-%offset%)
         (y double-float y-%data% y-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (kx 0) (ky 0)
             (temp1 0.0) (temp2 0.0))
        (declare (type fixnum i info ix iy j jx jy kx ky)
                 (type (double-float) temp1 temp2))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info 5))
          ((= incx 0)
           (setf info 7))
          ((= incy 0)
           (setf info 10)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DSYMV" info)
           (go end_label)))
        (if (or (= n 0) (and (= alpha zero) (= beta one))) (go end_label))
        (cond
          ((> incx 0)
           (setf kx 1))
          (t
           (setf kx
                 (f2cl-lib:int-sub 1
                                   (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                     incx)))))
        (cond
```

```
          ((> incy 0)
           (setf ky 1))
          (t
           (setf ky
                 (f2cl-lib:int-sub 1
                                   (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                     incy)))))
        (cond
          ((/= beta one)
           (cond
             ((= incy 1)
              (cond
                ((= beta zero)
                 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                               ((> i n) nil)
                   (tagbody
                     (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                             zero))))
                (t
                 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                               ((> i n) nil)
                   (tagbody
                     (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                             (* beta
                                (f2cl-lib:fref y-%data%
                                               (i)
                                               ((1 *))
                                               y-%offset%)))))))))
             (t
              (setf iy ky)
              (cond
                ((= beta zero)
                 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                               ((> i n) nil)
                   (tagbody
                     (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                             zero)
                     (setf iy (f2cl-lib:int-add iy incy)))))
                (t
                 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                               ((> i n) nil)
                   (tagbody
                     (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                             (* beta
                                (f2cl-lib:fref y-%data%
                                               (iy)
```

```
                                          ((1 *))
                                          y-%offset%)))
                (setf iy (f2cl-lib:int-add iy incy))))))))))
(if (= alpha zero) (go end_label))
(cond
  ((char-equal uplo #\U)
   (cond
     ((and (= incx 1) (= incy 1))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (setf temp1
                  (* alpha
                     (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
          (setf temp2 zero)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i
                            (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                         nil)
            (tagbody
              (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                      (+
                       (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                       (* temp1
                          (f2cl-lib:fref a-%data%
                                         (i j)
                                         ((1 lda) (1 *))
                                         a-%offset%))))
              (setf temp2
                      (+ temp2
                         (*
                          (f2cl-lib:fref a-%data%
                                         (i j)
                                         ((1 lda) (1 *))
                                         a-%offset%)
                          (f2cl-lib:fref x-%data%
                                         (i)
                                         ((1 *))
                                         x-%offset%))))))
          (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                  (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                     (* temp1
                        (f2cl-lib:fref a-%data%
                                       (j j)
                                       ((1 lda) (1 *))
                                       a-%offset%))
```

```
                (* alpha temp2))))))
          (t
           (setf jx kx)
           (setf jy ky)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf temp1
                       (* alpha
                          (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
               (setf temp2 zero)
               (setf ix kx)
               (setf iy ky)
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i
                                 (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                              nil)
                 (tagbody
                   (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                         (+
                          (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                          (* temp1
                             (f2cl-lib:fref a-%data%
                                            (i j)
                                            ((1 lda) (1 *))
                                            a-%offset%))))
                   (setf temp2
                         (+ temp2
                            (*
                             (f2cl-lib:fref a-%data%
                                            (i j)
                                            ((1 lda) (1 *))
                                            a-%offset%)
                             (f2cl-lib:fref x-%data%
                                            (ix)
                                            ((1 *))
                                            x-%offset%))))
                   (setf ix (f2cl-lib:int-add ix incx))
                   (setf iy (f2cl-lib:int-add iy incy))))
               (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                       (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                          (* temp1
                             (f2cl-lib:fref a-%data%
                                           (j j)
                                           ((1 lda) (1 *))
                                           a-%offset%))
```

```
                    (* alpha temp2)))
            (setf jx (f2cl-lib:int-add jx incx))
            (setf jy (f2cl-lib:int-add jy incy)))))))
    (t
     (cond
       ((and (= incx 1) (= incy 1))
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                      ((> j n) nil)
          (tagbody
            (setf temp1
                    (* alpha
                       (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
            (setf temp2 zero)
            (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                    (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                       (* temp1
                          (f2cl-lib:fref a-%data%
                                         (j j)
                                         ((1 lda) (1 *))
                                         a-%offset%))))
            (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                             (f2cl-lib:int-add i 1))
                          ((> i n) nil)
              (tagbody
                (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                        (+
                         (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                         (* temp1
                            (f2cl-lib:fref a-%data%
                                           (i j)
                                           ((1 lda) (1 *))
                                           a-%offset%))))
                (setf temp2
                        (+ temp2
                           (*
                            (f2cl-lib:fref a-%data%
                                           (i j)
                                           ((1 lda) (1 *))
                                           a-%offset%)
                            (f2cl-lib:fref x-%data%
                                           (i)
                                           ((1 *))
                                           x-%offset%))))))
            (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                    (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                       (* alpha temp2))))))
```

```
(t
 (setf jx kx)
 (setf jy ky)
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j n) nil)
   (tagbody
     (setf temp1
             (* alpha
                (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
     (setf temp2 zero)
     (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
             (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                (* temp1
                   (f2cl-lib:fref a-%data%
                                  (j j)
                                  ((1 lda) (1 *))
                                  a-%offset%))))
     (setf ix jx)
     (setf iy jy)
     (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                      (f2cl-lib:int-add i 1))
                   ((> i n) nil)
       (tagbody
         (setf ix (f2cl-lib:int-add ix incx))
         (setf iy (f2cl-lib:int-add iy incy))
         (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                 (+
                  (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                  (* temp1
                     (f2cl-lib:fref a-%data%
                                    (i j)
                                    ((1 lda) (1 *))
                                    a-%offset%))))
         (setf temp2
                 (+ temp2
                    (*
                     (f2cl-lib:fref a-%data%
                                    (i j)
                                    ((1 lda) (1 *))
                                    a-%offset%)
                     (f2cl-lib:fref x-%data%
                                    (ix)
                                    ((1 *))
                                    x-%offset%))))))
     (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
             (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
```

```
                                (* alpha temp2)))
                    (setf jx (f2cl-lib:int-add jx incx))
                    (setf jy (f2cl-lib:int-add jy incy))))))))
   end_label
          (return (values nil nil nil nil nil nil nil nil nil nil))))))
```

5.9 dsyr2 BLAS

⟨*dsyr2.input*⟩≡

```
  )set break resume
  )sys rm -f dsyr2.output
  )spool dsyr2.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dsyr2.help*⟩≡

```
  ======================================================================
  dsyr2 examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        DSYR2 - perform the symmetric rank 2 operation   A :=
        alpha*x*y' + alpha*y*x' + A,

   SYNOPSIS
        SUBROUTINE DSYR2 ( UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA
                         )

            DOUBLE       PRECISION ALPHA

            INTEGER      INCX, INCY, LDA, N

            CHARACTER*1  UPLO

            DOUBLE       PRECISION A( LDA, * ), X( * ), Y( * )

   PURPOSE
        DSYR2  performs the symmetric rank 2 operation

        where alpha is a scalar, x and y are n element vectors and A
        is an n by n symmetric matrix.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On entry, UPLO specifies whether the upper or lower
                 triangular part of the array A is to be referenced as
                 follows:

                 UPLO = 'U' or 'u'   Only the upper triangular part of
                 A is to be referenced.

                 UPLO = 'L' or 'l'   Only the lower triangular part of
                 A is to be referenced.

                 Unchanged on exit.

        N      - INTEGER.
```

```
          On entry, N specifies the order of the matrix A.  N
          must be at least zero.  Unchanged on exit.

ALPHA     - DOUBLE PRECISION.
          On entry, ALPHA specifies the scalar alpha.
          Unchanged on exit.

X         - DOUBLE PRECISION array of dimension at least
          ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
          incremented array X must contain the n element vector
          x.  Unchanged on exit.

INCX      - INTEGER.
          On entry, INCX specifies the increment for the

          elements of X. INCX must not be zero.  Unchanged on
          exit.

Y         - DOUBLE PRECISION array of dimension at least
          ( 1 + ( n - 1 )*abs( INCY ) ).  Before entry, the
          incremented array Y must contain the n element vector
          y.  Unchanged on exit.

INCY      - INTEGER.
          On entry, INCY specifies the increment for the ele-
          ments of Y. INCY must not be zero.  Unchanged on
          exit.

A         - DOUBLE PRECISION array of DIMENSION ( LDA, n ).
          Before entry with  UPLO = 'U' or 'u', the leading n
          by n upper triangular part of the array A must con-
          tain the upper triangular part of the symmetric
          matrix and the strictly lower triangular part of A is
          not referenced. On exit, the upper triangular part of
          the array A is overwritten by the upper triangular
          part of the updated matrix.  Before entry with UPLO =
          'L' or 'l', the leading n by n lower triangular part
          of the array A must contain the lower triangular part
          of the symmetric matrix and the strictly upper tri-
          angular part of A is not referenced. On exit, the
          lower triangular part of the array A is overwritten
          by the lower triangular part of the updated matrix.

LDA       - INTEGER.
          On entry, LDA specifies the first dimension of A as
          declared in the calling (sub) program. LDA must be at
```

```
                least max( 1, n ).  Unchanged on exit.
```

⟨*BLAS 2 dsyr2*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dsyr2 (uplo n alpha x incx y incy a lda)
    (declare (type (simple-array double-float (*)) a y x)
             (type (double-float) alpha)
             (type fixnum lda incy incx n)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (x double-float x-%data% x-%offset%)
         (y double-float y-%data% y-%offset%)
         (a double-float a-%data% a-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (kx 0) (ky 0)
             (temp1 0.0) (temp2 0.0))
        (declare (type fixnum i info ix iy j jx jy kx ky)
                 (type (double-float) temp1 temp2))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((= incx 0)
           (setf info 5))
          ((= incy 0)
           (setf info 7))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info 9)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DSYR2" info)
           (go end_label)))
        (if (or (= n 0) (= alpha zero)) (go end_label))
        (cond
          ((or (/= incx 1) (/= incy 1))
           (cond
             ((> incx 0)
              (setf kx 1))
             (t
              (setf kx
                      (f2cl-lib:int-sub 1
                                        (f2cl-lib:int-mul
                                         (f2cl-lib:int-sub n 1)
```

```
                                    incx)))))
    (cond
      ((> incy 0)
       (setf ky 1))
      (t
       (setf ky
             (f2cl-lib:int-sub 1
                               (f2cl-lib:int-mul
                                (f2cl-lib:int-sub n 1)
                                incy)))))
    (setf jx kx)
    (setf jy ky)))
  (cond
    ((char-equal uplo #\U)
     (cond
       ((and (= incx 1) (= incy 1))
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                      ((> j n) nil)
          (tagbody
            (cond
              ((or (/= (f2cl-lib:fref x (j) ((1 *))) zero)
                   (/= (f2cl-lib:fref y (j) ((1 *))) zero))
               (setf temp1
                       (* alpha
                          (f2cl-lib:fref y-%data%
                                         (j)
                                         ((1 *))
                                         y-%offset%)))
               (setf temp2
                       (* alpha
                          (f2cl-lib:fref x-%data%
                                         (j)
                                         ((1 *))
                                         x-%offset%)))
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i j) nil)
                 (tagbody
                   (setf (f2cl-lib:fref a-%data%
                                        (i j)
                                        ((1 lda) (1 *))
                                        a-%offset%)
                         (+
                          (f2cl-lib:fref a-%data%
                                         (i j)
                                         ((1 lda) (1 *))
                                         a-%offset%)
```

```
                                (*
                                 (f2cl-lib:fref x-%data%
                                                (i)
                                                ((1 *))
                                                x-%offset%)
                                 temp1)
                                (*
                                 (f2cl-lib:fref y-%data%
                                                (i)
                                                ((1 *))
                                                y-%offset%)
                                 temp2))))))))))
                        (t
                         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                       ((> j n) nil)
                           (tagbody
                             (cond
                               ((or (/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                                    (/= (f2cl-lib:fref y (jy) ((1 *))) zero))
                                (setf temp1
                                        (* alpha
                                           (f2cl-lib:fref y-%data%
                                                          (jy)
                                                          ((1 *))
                                                          y-%offset%)))
                                (setf temp2
                                        (* alpha
                                           (f2cl-lib:fref x-%data%
                                                          (jx)
                                                          ((1 *))
                                                          x-%offset%)))
                                (setf ix kx)
                                (setf iy ky)
                                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                              ((> i j) nil)
                                  (tagbody
                                    (setf (f2cl-lib:fref a-%data%
                                                         (i j)
                                                         ((1 lda) (1 *))
                                                         a-%offset%)
                                            (+
                                             (f2cl-lib:fref a-%data%
                                                            (i j)
                                                            ((1 lda) (1 *))
                                                            a-%offset%)
                                             (*
```

```
                              (f2cl-lib:fref x-%data%
                                             (ix)
                                             ((1 *))
                                             x-%offset%)
                              temp1)
                             (*
                              (f2cl-lib:fref y-%data%
                                             (iy)
                                             ((1 *))
                                             y-%offset%)
                              temp2)))
                    (setf ix (f2cl-lib:int-add ix incx))
                    (setf iy (f2cl-lib:int-add iy incy))))))
              (setf jx (f2cl-lib:int-add jx incx))
              (setf jy (f2cl-lib:int-add jy incy)))))))
      (t
       (cond
         ((and (= incx 1) (= incy 1))
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                        ((> j n) nil)
            (tagbody
              (cond
                ((or (/= (f2cl-lib:fref x (j) ((1 *))) zero)
                     (/= (f2cl-lib:fref y (j) ((1 *))) zero))
                 (setf temp1
                         (* alpha
                            (f2cl-lib:fref y-%data%
                                           (j)
                                           ((1 *))
                                           y-%offset%)))
                 (setf temp2
                         (* alpha
                            (f2cl-lib:fref x-%data%
                                           (j)
                                           ((1 *))
                                           x-%offset%)))
                 (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                               ((> i n) nil)
                   (tagbody
                     (setf (f2cl-lib:fref a-%data%
                                          (i j)
                                          ((1 lda) (1 *))
                                          a-%offset%)
                           (+
                            (f2cl-lib:fref a-%data%
                                           (i j)
```

```
                                          ((1 lda) (1 *))
                                          a-%offset%)
                                  (*
                                   (f2cl-lib:fref x-%data%
                                                  (i)
                                                  ((1 *))
                                                  x-%offset%)
                                   temp1)
                                  (*
                                   (f2cl-lib:fref y-%data%
                                                  (i)
                                                  ((1 *))
                                                  y-%offset%)
                                   temp2))))))))))
                (t
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (cond
                       ((or (/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                            (/= (f2cl-lib:fref y (jy) ((1 *))) zero))
                        (setf temp1
                                (* alpha
                                   (f2cl-lib:fref y-%data%
                                                  (jy)
                                                  ((1 *))
                                                  y-%offset%)))
                        (setf temp2
                                (* alpha
                                   (f2cl-lib:fref x-%data%
                                                  (jx)
                                                  ((1 *))
                                                  x-%offset%)))
                        (setf ix jx)
                        (setf iy jy)
                        (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                      ((> i n) nil)
                          (tagbody
                            (setf (f2cl-lib:fref a-%data%
                                                 (i j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%)
                                  (+
                                   (f2cl-lib:fref a-%data%
                                                  (i j)
                                                  ((1 lda) (1 *))
```

```
                                                a-%offset%)
                                        (*
                                         (f2cl-lib:fref x-%data%
                                                        (ix)
                                                        ((1 *))
                                                        x-%offset%)
                                         temp1)
                                        (*
                                         (f2cl-lib:fref y-%data%
                                                        (iy)
                                                        ((1 *))
                                                        y-%offset%)
                                         temp2)))
                                (setf ix (f2cl-lib:int-add ix incx))
                                (setf iy (f2cl-lib:int-add iy incy))))))
                        (setf jx (f2cl-lib:int-add jx incx))
                        (setf jy (f2cl-lib:int-add jy incy))))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil nil))))))
```

5.10 dsyr BLAS

⟨*dsyr.input*⟩≡

```
  )set break resume
  )sys rm -f dsyr.output
  )spool dsyr.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dsyr.help*⟩≡

```
 ========================================================================
 dsyr examples
 ========================================================================

 ========================================================================
 Man Page Details
 ========================================================================

  NAME
       DSYR - perform the symmetric rank 1 operation   A :=
       alpha*x*x' + A,

  SYNOPSIS
       SUBROUTINE DSYR ( UPLO, N, ALPHA, X, INCX, A, LDA )

           DOUBLE      PRECISION ALPHA

           INTEGER     INCX, LDA, N

           CHARACTER*1 UPLO

           DOUBLE      PRECISION A( LDA, * ), X( * )

  PURPOSE
       DSYR   performs the symmetric rank 1 operation

       where alpha is a real scalar, x is an n element vector and A
       is an n by n symmetric matrix.

  PARAMETERS
       UPLO   - CHARACTER*1.
              On entry, UPLO specifies whether the upper or lower
              triangular part of the array A is to be referenced as
              follows:

              UPLO = 'U' or 'u'   Only the upper triangular part of
              A is to be referenced.

              UPLO = 'L' or 'l'   Only the lower triangular part of
              A is to be referenced.

              Unchanged on exit.

       N      - INTEGER.
              On entry, N specifies the order of the matrix A.  N
```

```
       must be at least zero.  Unchanged on exit.

ALPHA  - DOUBLE PRECISION.
       On entry, ALPHA specifies the scalar alpha.
       Unchanged on exit.

X      - DOUBLE PRECISION array of dimension at least
       ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
       incremented array X must contain the n element vector
       x.  Unchanged on exit.

INCX   - INTEGER.
       On entry, INCX specifies the increment for the ele-
       ments of X. INCX must not be zero.  Unchanged on

       exit.

A      - DOUBLE PRECISION array of DIMENSION ( LDA, n ).
       Before entry with  UPLO = 'U' or 'u', the leading n
       by n upper triangular part of the array A must con-
       tain the upper triangular part of the symmetric
       matrix and the strictly lower triangular part of A is
       not referenced. On exit, the upper triangular part of
       the array A is overwritten by the upper triangular
       part of the updated matrix.  Before entry with UPLO =
       'L' or 'l', the leading n by n lower triangular part
       of the array A must contain the lower triangular part
       of the symmetric matrix and the strictly upper tri-
       angular part of A is not referenced. On exit, the
       lower triangular part of the array A is overwritten
       by the lower triangular part of the updated matrix.

LDA    - INTEGER.
       On entry, LDA specifies the first dimension of A as
       declared in the calling (sub) program. LDA must be at
       least max( 1, n ).  Unchanged on exit.
```

⟨*BLAS 2 dsyr*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dsyr (uplo n alpha x incx a lda)
    (declare (type (simple-array double-float (*)) a x)
             (type (double-float) alpha)
             (type fixnum lda incx n)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (x double-float x-%data% x-%offset%)
         (a double-float a-%data% a-%offset%))
      (prog ((i 0) (info 0) (ix 0) (j 0) (jx 0) (kx 0) (temp 0.0))
        (declare (type fixnum i info ix j jx kx)
                 (type (double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((= incx 0)
           (setf info 5))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info 7)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DSYR" info)
           (go end_label)))
        (if (or (= n 0) (= alpha zero)) (go end_label))
        (cond
          ((<= incx 0)
           (setf kx
                   (f2cl-lib:int-sub 1
                                     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                       incx))))
          ((/= incx 1)
           (setf kx 1)))
        (cond
          ((char-equal uplo #\U)
           (cond
             ((= incx 1)
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
```

```
      (tagbody
        (cond
          ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
           (setf temp
                   (* alpha
                      (f2cl-lib:fref x-%data%
                                     (j)
                                     ((1 *))
                                     x-%offset%)))
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                         ((> i j) nil)
             (tagbody
               (setf (f2cl-lib:fref a-%data%
                                    (i j)
                                    ((1 lda) (1 *))
                                    a-%offset%)
                     (+
                      (f2cl-lib:fref a-%data%
                                     (i j)
                                     ((1 lda) (1 *))
                                     a-%offset%)
                      (*
                       (f2cl-lib:fref x-%data%
                                      (i)
                                      ((1 *))
                                      x-%offset%)
                       temp)))))))))
    (t
     (setf jx kx)
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   ((> j n) nil)
       (tagbody
         (cond
           ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
            (setf temp
                    (* alpha
                       (f2cl-lib:fref x-%data%
                                      (jx)
                                      ((1 *))
                                      x-%offset%)))
            (setf ix kx)
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                          ((> i j) nil)
              (tagbody
                (setf (f2cl-lib:fref a-%data%
                                     (i j)
```

```
                                        ((1 lda) (1 *))
                                        a-%offset%)
                                (+
                                 (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%)
                                 (*
                                  (f2cl-lib:fref x-%data%
                                                 (ix)
                                                 ((1 *))
                                                 x-%offset%)
                                  temp)))
                          (setf ix (f2cl-lib:int-add ix incx))))))
                  (setf jx (f2cl-lib:int-add jx incx)))))))
        (t
         (cond
           ((= incx 1)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                          ((> j n) nil)
              (tagbody
                (cond
                  ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                   (setf temp
                           (* alpha
                              (f2cl-lib:fref x-%data%
                                             (j)
                                             ((1 *))
                                             x-%offset%)))
                   (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref a-%data%
                                            (i j)
                                            ((1 lda) (1 *))
                                            a-%offset%)
                                (+
                                 (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%)
                                 (*
                                  (f2cl-lib:fref x-%data%
                                                 (i)
                                                 ((1 *))
                                                 x-%offset%)
```

```
                                temp))))))))))
              (t
               (setf jx kx)
               (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                             ((> j n) nil)
                 (tagbody
                   (cond
                     ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                      (setf temp
                              (* alpha
                                 (f2cl-lib:fref x-%data%
                                                (jx)
                                                ((1 *))
                                                x-%offset%)))
                      (setf ix jx)
                      (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                    ((> i n) nil)
                        (tagbody
                          (setf (f2cl-lib:fref a-%data%
                                               (i j)
                                               ((1 lda) (1 *))
                                               a-%offset%)
                                (+
                                 (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%)
                                 (*
                                  (f2cl-lib:fref x-%data%
                                                 (ix)
                                                 ((1 *))
                                                 x-%offset%)
                                  temp)))
                          (setf ix (f2cl-lib:int-add ix incx))))))
                   (setf jx (f2cl-lib:int-add jx incx)))))))
  end_label
        (return (values nil nil nil nil nil nil nil))))))
```

5.11 dtbmv BLAS

⟨*dtbmv.input*⟩≡

```
)set break resume
)sys rm -f dtbmv.output
)spool dtbmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dtbmv.help*⟩≡

```
  ======================================================================
  dtbmv examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        DTBMV - perform one of the matrix-vector operations   x :=
        A*x, or x := A'*x,

   SYNOPSIS
        SUBROUTINE DTBMV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX
                         )

            INTEGER      INCX, K, LDA, N

            CHARACTER*1  DIAG, TRANS, UPLO

            DOUBLE       PRECISION A( LDA, * ), X( * )

   PURPOSE
        DTBMV  performs one of the matrix-vector operations

        where x is an n element vector and  A is an n by n unit, or
        non-unit, upper or lower triangular band matrix, with ( k +
        1 ) diagonals.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On entry, UPLO specifies whether the matrix is an
                 upper or lower triangular matrix as follows:

                 UPLO = 'U' or 'u'   A is an upper triangular matrix.

                 UPLO = 'L' or 'l'   A is a lower triangular matrix.

                 Unchanged on exit.

        TRANS  - CHARACTER*1.
                 On entry, TRANS specifies the operation to be per-
                 formed as follows:

                 TRANS = 'N' or 'n'   x := A*x.
```

```
        TRANS = 'T' or 't'   x := A'*x.

        TRANS = 'C' or 'c'   x := A'*x.

        Unchanged on exit.

DIAG    - CHARACTER*1.
        On entry, DIAG specifies whether or not A is unit
        triangular as follows:

        DIAG = 'U' or 'u'   A is assumed to be unit triangu-
        lar.

        DIAG = 'N' or 'n'   A is not assumed to be unit tri-
        angular.

        Unchanged on exit.

N       - INTEGER.
        On entry, N specifies the order of the matrix A.  N
        must be at least zero.  Unchanged on exit.

K       - INTEGER.
        On entry with UPLO = 'U' or 'u', K specifies the
        number of super-diagonals of the matrix A.  On entry
        with UPLO = 'L' or 'l', K specifies the number of
        sub-diagonals of the matrix A.  K must satisfy  0
        .le. K.  Unchanged on exit.

A       - DOUBLE PRECISION array of DIMENSION ( LDA, n ).
        Before entry with UPLO = 'U' or 'u', the leading ( k
        + 1 ) by n part of the array A must contain the upper
        triangular band part of the matrix of coefficients,
        supplied column by column, with the leading diagonal
        of the matrix in row ( k + 1 ) of the array, the
        first super-diagonal starting at position 2 in row k,
        and so on. The top left k by k triangle of the array
        A is not referenced.  The following program segment
        will transfer an upper triangular band matrix from
        conventional full matrix storage to band storage:

        DO 20, J = 1, N M = K + 1 - J DO 10, I = MAX( 1, J -
        K ), J A( M + I, J ) = matrix( I, J ) 10    CONTINUE
        20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

DO 20, J = 1, N M = 1 - J DO 10, I = J, MIN(N, J + K) A(M + I, J) = matrix(I, J) 10 CONTINUE 20 CONTINUE

Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity. Unchanged on exit.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1). Unchanged on exit.

X - DOUBLE PRECISION array of dimension at least (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

⟨*BLAS 2 dtbmv*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dtbmv (uplo trans diag n k a lda x incx)
    (declare (type (simple-array double-float (*)) x a)
             (type fixnum incx lda k n)
             (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (diag character diag-%data% diag-%offset%)
         (a double-float a-%data% a-%offset%)
         (x double-float x-%data% x-%offset%))
      (prog ((nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0) (kplus1 0) (kx 0)
             (l 0) (temp 0.0))
        (declare (type (member t nil) nounit)
                 (type fixnum i info ix j jx kplus1 kx l)
                 (type (double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 2))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((< k 0)
           (setf info 5))
          ((< lda (f2cl-lib:int-add k 1))
           (setf info 7))
          ((= incx 0)
           (setf info 9)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DTBMV" info)
           (go end_label)))
        (if (= n 0) (go end_label))
        (setf nounit (char-equal diag #\N))
        (cond
          ((<= incx 0)
```

```
            (setf kx
                    (f2cl-lib:int-sub 1
                                      (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                        incx))))
          ((/= incx 1)
           (setf kx 1)))
        (cond
          ((char-equal trans #\N)
           (cond
             ((char-equal uplo #\U)
              (setf kplus1 (f2cl-lib:int-add k 1))
              (cond
                ((= incx 1)
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (cond
                       ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                        (setf temp
                                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                        (setf l (f2cl-lib:int-sub kplus1 j))
                        (f2cl-lib:fdo (i
                                       (max (the fixnum 1)
                                            (the fixnum
                                                 (f2cl-lib:int-add j
                                                                   (f2cl-lib:int-sub
                                                                    k))))
                                       (f2cl-lib:int-add i 1))
                                      ((> i
                                          (f2cl-lib:int-add j
                                                            (f2cl-lib:int-sub
                                                             1)))
                                       nil)
                          (tagbody
                            (setf (f2cl-lib:fref x-%data%
                                                 (i)
                                                 ((1 *))
                                                 x-%offset%)
                                  (+
                                   (f2cl-lib:fref x-%data%
                                                  (i)
                                                  ((1 *))
                                                  x-%offset%)
                                   (* temp
                                      (f2cl-lib:fref a-%data%
                                                     ((f2cl-lib:int-add l i)
```

```
                                                j)
                                               ((1 lda) (1 *))
                                               a-%offset%))))))
                        (if nounit
                            (setf (f2cl-lib:fref x-%data%
                                                 (j)
                                                 ((1 *))
                                                 x-%offset%)
                                  (*
                                   (f2cl-lib:fref x-%data%
                                                  (j)
                                                  ((1 *))
                                                  x-%offset%)
                                   (f2cl-lib:fref a-%data%
                                                  (kplus1 j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%)))))))))
                (t
                 (setf jx kx)
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (cond
                       ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                        (setf temp
                                (f2cl-lib:fref x-%data%
                                               (jx)
                                               ((1 *))
                                               x-%offset%))
                        (setf ix kx)
                        (setf l (f2cl-lib:int-sub kplus1 j))
                        (f2cl-lib:fdo (i
                                       (max (the fixnum 1)
                                            (the fixnum
                                                 (f2cl-lib:int-add j
                                                                   (f2cl-lib:int-sub
                                                                    k))))
                                       (f2cl-lib:int-add i 1))
                                      ((> i
                                          (f2cl-lib:int-add j
                                                            (f2cl-lib:int-sub
                                                             1)))
                                       nil)
                          (tagbody
                            (setf (f2cl-lib:fref x-%data%
                                                 (ix)
```

```
                                        ((1 *))
                                        x-%offset%)
                                (+
                                 (f2cl-lib:fref x-%data%
                                                (ix)
                                                ((1 *))
                                                x-%offset%)
                                 (* temp
                                    (f2cl-lib:fref a-%data%
                                                   ((f2cl-lib:int-add l i)
                                                    j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%))))
                            (setf ix (f2cl-lib:int-add ix incx))))
                        (if nounit
                            (setf (f2cl-lib:fref x-%data%
                                                 (jx)
                                                 ((1 *))
                                                 x-%offset%)
                                  (*
                                   (f2cl-lib:fref x-%data%
                                                  (jx)
                                                  ((1 *))
                                                  x-%offset%)
                                   (f2cl-lib:fref a-%data%
                                                  (kplus1 j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))))))
                        (setf jx (f2cl-lib:int-add jx incx))
                        (if (> j k) (setf kx (f2cl-lib:int-add kx incx))))))))
                (t
                 (cond
                   ((= incx 1)
                    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                  ((> j 1) nil)
                      (tagbody
                        (cond
                          ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                           (setf temp
                                   (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                           (setf l (f2cl-lib:int-sub 1 j))
                           (f2cl-lib:fdo (i
                                          (min (the fixnum n)
                                               (the fixnum
                                                    (f2cl-lib:int-add j k)))
                                          (f2cl-lib:int-add i
```

```
                                                (f2cl-lib:int-sub 1)))
                                ((> i (f2cl-lib:int-add j 1)) nil)
                          (tagbody
                            (setf (f2cl-lib:fref x-%data%
                                                 (i)
                                                 ((1 *))
                                                 x-%offset%)
                                  (+
                                   (f2cl-lib:fref x-%data%
                                                  (i)
                                                  ((1 *))
                                                  x-%offset%)
                                   (* temp
                                      (f2cl-lib:fref a-%data%
                                                     ((f2cl-lib:int-add l i)
                                                      j)
                                                     ((1 lda) (1 *))
                                                     a-%offset%))))))
                          (if nounit
                              (setf (f2cl-lib:fref x-%data%
                                                   (j)
                                                   ((1 *))
                                                   x-%offset%)
                                    (*
                                     (f2cl-lib:fref x-%data%
                                                    (j)
                                                    ((1 *))
                                                    x-%offset%)
                                     (f2cl-lib:fref a-%data%
                                                    (1 j)
                                                    ((1 lda) (1 *))
                                                    a-%offset%)))))))))
                  (t
                   (setf kx
                         (f2cl-lib:int-add kx
                                           (f2cl-lib:int-mul
                                            (f2cl-lib:int-sub n 1)
                                            incx)))
                   (setf jx kx)
                   (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                 ((> j 1) nil)
                     (tagbody
                       (cond
                         ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                          (setf temp
                                  (f2cl-lib:fref x-%data%
```

```
                                (jx)
                                ((1 *))
                                x-%offset%))
          (setf ix kx)
          (setf l (f2cl-lib:int-sub 1 j))
          (f2cl-lib:fdo (i
                         (min (the fixnum n)
                              (the fixnum
                                   (f2cl-lib:int-add j k)))
                         (f2cl-lib:int-add i
                                           (f2cl-lib:int-sub 1)))
                        ((> i (f2cl-lib:int-add j 1)) nil)
            (tagbody
              (setf (f2cl-lib:fref x-%data%
                                   (ix)
                                   ((1 *))
                                   x-%offset%)
                    (+
                     (f2cl-lib:fref x-%data%
                                    (ix)
                                    ((1 *))
                                    x-%offset%)
                     (* temp
                        (f2cl-lib:fref a-%data%
                                       ((f2cl-lib:int-add l i)
                                        j)
                                       ((1 lda) (1 *))
                                       a-%offset%))))
              (setf ix (f2cl-lib:int-sub ix incx))))
          (if nounit
              (setf (f2cl-lib:fref x-%data%
                                   (jx)
                                   ((1 *))
                                   x-%offset%)
                    (*
                     (f2cl-lib:fref x-%data%
                                    (jx)
                                    ((1 *))
                                    x-%offset%)
                     (f2cl-lib:fref a-%data%
                                    (1 j)
                                    ((1 lda) (1 *))
                                    a-%offset%))))))
      (setf jx (f2cl-lib:int-sub jx incx))
      (if (>= (f2cl-lib:int-sub n j) k)
          (setf kx (f2cl-lib:int-sub kx incx)))))))))
```

```
(t
 (cond
   ((char-equal uplo #\U)
    (setf kplus1 (f2cl-lib:int-add k 1))
    (cond
      ((= incx 1)
       (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                     ((> j 1) nil)
         (tagbody
           (setf temp
                   (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
           (setf l (f2cl-lib:int-sub kplus1 j))
           (if nounit
               (setf temp
                       (* temp
                          (f2cl-lib:fref a-%data%
                                         (kplus1 j)
                                         ((1 lda) (1 *))
                                         a-%offset%))))
           (f2cl-lib:fdo (i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))
                            (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                         ((> i
                             (max (the fixnum 1)
                                  (the fixnum
                                       (f2cl-lib:int-add j
                                                         (f2cl-lib:int-sub
                                                          k)))))
                            nil)
             (tagbody
               (setf temp
                       (+ temp
                          (*
                           (f2cl-lib:fref a-%data%
                                          ((f2cl-lib:int-add l i) j)
                                          ((1 lda) (1 *))
                                          a-%offset%)
                           (f2cl-lib:fref x-%data%
                                          (i)
                                          ((1 *))
                                          x-%offset%))))))
           (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                   temp))))
      (t
       (setf kx
               (f2cl-lib:int-add kx
                                 (f2cl-lib:int-mul
```

```
                                    (f2cl-lib:int-sub n 1)
                                    incx)))
            (setf jx kx)
            (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                          ((> j 1) nil)
              (tagbody
                (setf temp
                        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
                (setf kx (f2cl-lib:int-sub kx incx))
                (setf ix kx)
                (setf l (f2cl-lib:int-sub kplus1 j))
                (if nounit
                    (setf temp
                            (* temp
                               (f2cl-lib:fref a-%data%
                                              (kplus1 j)
                                              ((1 lda) (1 *))
                                              a-%offset%))))
                (f2cl-lib:fdo (i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))
                                 (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                              ((> i
                                  (max (the fixnum 1)
                                       (the fixnum
                                            (f2cl-lib:int-add j
                                                              (f2cl-lib:int-sub
                                                               k)))))
                                 nil)
                  (tagbody
                    (setf temp
                            (+ temp
                               (*
                                (f2cl-lib:fref a-%data%
                                               ((f2cl-lib:int-add l i) j)
                                               ((1 lda) (1 *))
                                               a-%offset%)
                                (f2cl-lib:fref x-%data%
                                               (ix)
                                               ((1 *))
                                               x-%offset%))))
                    (setf ix (f2cl-lib:int-sub ix incx))))
                (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                        temp)
                (setf jx (f2cl-lib:int-sub jx incx)))))))
          (t
           (cond
             ((= incx 1)
```

```
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j n) nil)
   (tagbody
     (setf temp
             (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
     (setf l (f2cl-lib:int-sub 1 j))
     (if nounit
         (setf temp
                 (* temp
                    (f2cl-lib:fref a-%data%
                                   (1 j)
                                   ((1 lda) (1 *))
                                   a-%offset%))))
     (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                      (f2cl-lib:int-add i 1))
                   ((> i
                       (min (the fixnum n)
                            (the fixnum
                                 (f2cl-lib:int-add j k))))
                    nil)
       (tagbody
         (setf temp
                 (+ temp
                    (*
                     (f2cl-lib:fref a-%data%
                                    ((f2cl-lib:int-add l i) j)
                                    ((1 lda) (1 *))
                                    a-%offset%)
                     (f2cl-lib:fref x-%data%
                                    (i)
                                    ((1 *))
                                    x-%offset%))))))
     (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
             temp))))
(t
 (setf jx kx)
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j n) nil)
   (tagbody
     (setf temp
             (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
     (setf kx (f2cl-lib:int-add kx incx))
     (setf ix kx)
     (setf l (f2cl-lib:int-sub 1 j))
     (if nounit
         (setf temp
```

```
                                    (* temp
                                       (f2cl-lib:fref a-%data%
                                                      (1 j)
                                                      ((1 lda) (1 *))
                                                      a-%offset%))))
                          (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                          (f2cl-lib:int-add i 1))
                                        ((> i
                                            (min (the fixnum n)
                                                 (the fixnum
                                                      (f2cl-lib:int-add j k))))
                                         nil)
                            (tagbody
                              (setf temp
                                      (+ temp
                                         (*
                                          (f2cl-lib:fref a-%data%
                                                         ((f2cl-lib:int-add l i) j)
                                                         ((1 lda) (1 *))
                                                         a-%offset%)
                                          (f2cl-lib:fref x-%data%
                                                         (ix)
                                                         ((1 *))
                                                         x-%offset%))))
                              (setf ix (f2cl-lib:int-add ix incx))))
                          (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                                temp)
                          (setf jx (f2cl-lib:int-add jx incx)))))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil nil))))))
```

5.12 dtbsv BLAS

⟨*dtbsv.input*⟩≡

```
)set break resume
)sys rm -f dtbsv.output
)spool dtbsv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dtbsv.help*⟩≡

```
 ======================================================================
 dtbsv examples
 ======================================================================

 ======================================================================
 Man Page Details
 ======================================================================

  NAME
       DTBSV - solve one of the systems of equations   A*x = b, or
       A'*x = b,

  SYNOPSIS
       SUBROUTINE DTBSV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX
                        )

           INTEGER      INCX, K, LDA, N

           CHARACTER*1  DIAG, TRANS, UPLO

           DOUBLE       PRECISION A( LDA, * ), X( * )

  PURPOSE
       DTBSV  solves one of the systems of equations

       where b and x are n element vectors and A is an n by n unit,
       or non-unit, upper or lower triangular band matrix, with ( k
       + 1 ) diagonals.

       No test for singularity or near-singularity is included in
       this routine. Such tests must be performed before calling
       this routine.

  PARAMETERS
       UPLO   - CHARACTER*1.
              On entry, UPLO specifies whether the matrix is an
              upper or lower triangular matrix as follows:

              UPLO = 'U' or 'u'   A is an upper triangular matrix.

              UPLO = 'L' or 'l'   A is a lower triangular matrix.

              Unchanged on exit.

       TRANS  - CHARACTER*1.
```

```
          On entry, TRANS specifies the equations to be solved
          as follows:

          TRANS = 'N' or 'n'   A*x = b.

          TRANS = 'T' or 't'   A'*x = b.

          TRANS = 'C' or 'c'   A'*x = b.

          Unchanged on exit.

DIAG      - CHARACTER*1.
          On entry, DIAG specifies whether or not A is unit
          triangular as follows:

          DIAG = 'U' or 'u'   A is assumed to be unit triangu-
          lar.

          DIAG = 'N' or 'n'   A is not assumed to be unit tri-
          angular.

          Unchanged on exit.

N         - INTEGER.
          On entry, N specifies the order of the matrix A.  N
          must be at least zero.  Unchanged on exit.

K         - INTEGER.
          On entry with UPLO = 'U' or 'u', K specifies the
          number of super-diagonals of the matrix A.  On entry
          with UPLO = 'L' or 'l', K specifies the number of
          sub-diagonals of the matrix A.  K must satisfy  0
          .le. K.  Unchanged on exit.

A         - DOUBLE PRECISION array of DIMENSION ( LDA, n ).
          Before entry with UPLO = 'U' or 'u', the leading ( k
          + 1 ) by n part of the array A must contain the upper
          triangular band part of the matrix of coefficients,
          supplied column by column, with the leading diagonal
          of the matrix in row ( k + 1 ) of the array, the
          first super-diagonal starting at position 2 in row k,
          and so on. The top left k by k triangle of the array
          A is not referenced.  The following program segment
          will transfer an upper triangular band matrix from
          conventional full matrix storage to band storage:
```

DO 20, J = 1, N M = K + 1 - J DO 10, I = MAX(1, J - K), J A(M + I, J) = matrix(I, J) 10 CONTINUE 20 CONTINUE

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

DO 20, J = 1, N M = 1 - J DO 10, I = J, MIN(N, J + K) A(M + I, J) = matrix(I, J) 10 CONTINUE 20 CONTINUE

Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the

matrix are not referenced, but are assumed to be unity. Unchanged on exit.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1). Unchanged on exit.

X - DOUBLE PRECISION array of dimension at least (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

⟨*BLAS 2 dtbsv*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dtbsv (uplo trans diag n k a lda x incx)
    (declare (type (simple-array double-float (*)) x a)
             (type fixnum incx lda k n)
             (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (diag character diag-%data% diag-%offset%)
         (a double-float a-%data% a-%offset%)
         (x double-float x-%data% x-%offset%))
      (prog ((nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0) (kplus1 0) (kx 0)
             (l 0) (temp 0.0))
        (declare (type (member t nil) nounit)
                 (type fixnum i info ix j jx kplus1 kx l)
                 (type (double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 2))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((< k 0)
           (setf info 5))
          ((< lda (f2cl-lib:int-add k 1))
           (setf info 7))
          ((= incx 0)
           (setf info 9)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DTBSV" info)
           (go end_label)))
        (if (= n 0) (go end_label))
        (setf nounit (char-equal diag #\N))
        (cond
          ((<= incx 0)
```

```
   (setf kx
           (f2cl-lib:int-sub 1
                             (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                               incx))))
  ((/= incx 1)
   (setf kx 1)))
(cond
  ((char-equal trans #\N)
   (cond
     ((char-equal uplo #\U)
      (setf kplus1 (f2cl-lib:int-add k 1))
      (cond
        ((= incx 1)
         (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                       ((> j 1) nil)
           (tagbody
             (cond
               ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                (setf l (f2cl-lib:int-sub kplus1 j))
                (if nounit
                    (setf (f2cl-lib:fref x-%data%
                                         (j)
                                         ((1 *))
                                         x-%offset%)
                                  (/
                                   (f2cl-lib:fref x-%data%
                                                  (j)
                                                  ((1 *))
                                                  x-%offset%)
                                   (f2cl-lib:fref a-%data%
                                                  (kplus1 j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))))
                (setf temp
                        (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                (f2cl-lib:fdo (i
                               (f2cl-lib:int-add j
                                                 (f2cl-lib:int-sub 1))
                               (f2cl-lib:int-add i
                                                 (f2cl-lib:int-sub 1)))
                              ((> i
                                  (max (the fixnum 1)
                                       (the fixnum
                                            (f2cl-lib:int-add j
                                                              (f2cl-lib:int-s
                                                               k)))))
```

```
                              nil)
                        (tagbody
                          (setf (f2cl-lib:fref x-%data%
                                               (i)
                                               ((1 *))
                                               x-%offset%)
                                (-
                                 (f2cl-lib:fref x-%data%
                                                (i)
                                                ((1 *))
                                                x-%offset%)
                                 (* temp
                                    (f2cl-lib:fref a-%data%
                                                   ((f2cl-lib:int-add l i)
                                                    j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%)))))))))))
                (t
                 (setf kx
                       (f2cl-lib:int-add kx
                                         (f2cl-lib:int-mul
                                          (f2cl-lib:int-sub n 1)
                                          incx)))
                 (setf jx kx)
                 (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                               ((> j 1) nil)
                   (tagbody
                     (setf kx (f2cl-lib:int-sub kx incx))
                     (cond
                       ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                        (setf ix kx)
                        (setf l (f2cl-lib:int-sub kplus1 j))
                        (if nounit
                            (setf (f2cl-lib:fref x-%data%
                                                 (jx)
                                                 ((1 *))
                                                 x-%offset%)
                                  (/
                                   (f2cl-lib:fref x-%data%
                                                  (jx)
                                                  ((1 *))
                                                  x-%offset%)
                                   (f2cl-lib:fref a-%data%
                                                  (kplus1 j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))))
```

```
                (setf temp
                        (f2cl-lib:fref x-%data%
                                       (jx)
                                       ((1 *))
                                       x-%offset%))
                (f2cl-lib:fdo (i
                               (f2cl-lib:int-add j
                                                 (f2cl-lib:int-sub 1))
                               (f2cl-lib:int-add i
                                                 (f2cl-lib:int-sub 1)))
                              ((> i
                                  (max (the fixnum 1)
                                       (the fixnum
                                            (f2cl-lib:int-add j
                                                              (f2cl-lib:int-s
                                                               k)))))
                               nil)
                  (tagbody
                    (setf (f2cl-lib:fref x-%data%
                                         (ix)
                                         ((1 *))
                                         x-%offset%)
                          (-
                           (f2cl-lib:fref x-%data%
                                         (ix)
                                         ((1 *))
                                         x-%offset%)
                           (* temp
                              (f2cl-lib:fref a-%data%
                                             ((f2cl-lib:int-add l i)
                                              j)
                                             ((1 lda) (1 *))
                                             a-%offset%))))
                    (setf ix (f2cl-lib:int-sub ix incx))))))
          (setf jx (f2cl-lib:int-sub jx incx)))))))
(t
 (cond
   ((= incx 1)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  ((> j n) nil)
      (tagbody
        (cond
          ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
           (setf l (f2cl-lib:int-sub 1 j))
           (if nounit
               (setf (f2cl-lib:fref x-%data%
```

```
                                        (j)
                                        ((1 *))
                                        x-%offset%)
                                (/
                                 (f2cl-lib:fref x-%data%
                                                (j)
                                                ((1 *))
                                                x-%offset%)
                                 (f2cl-lib:fref a-%data%
                                                (1 j)
                                                ((1 lda) (1 *))
                                                a-%offset%))))
                        (setf temp
                                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                        (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                       (f2cl-lib:int-add i 1))
                                      ((> i
                                          (min (the fixnum n)
                                               (the fixnum
                                                    (f2cl-lib:int-add j k))))
                                       nil)
                          (tagbody
                            (setf (f2cl-lib:fref x-%data%
                                                 (i)
                                                 ((1 *))
                                                 x-%offset%)
                                  (-
                                   (f2cl-lib:fref x-%data%
                                                  (i)
                                                  ((1 *))
                                                  x-%offset%)
                                   (* temp
                                      (f2cl-lib:fref a-%data%
                                                     ((f2cl-lib:int-add l i)
                                                      j)
                                                     ((1 lda) (1 *))
                                                     a-%offset%))))))))
                (t
                 (setf jx kx)
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (setf kx (f2cl-lib:int-add kx incx))
                     (cond
                       ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                        (setf ix kx)
```

```
                    (setf l (f2cl-lib:int-sub 1 j))
                    (if nounit
                        (setf (f2cl-lib:fref x-%data%
                                             (jx)
                                             ((1 *))
                                             x-%offset%)
                              (/
                               (f2cl-lib:fref x-%data%
                                              (jx)
                                              ((1 *))
                                              x-%offset%)
                               (f2cl-lib:fref a-%data%
                                              (1 j)
                                              ((1 lda) (1 *))
                                              a-%offset%))))
                    (setf temp
                            (f2cl-lib:fref x-%data%
                                           (jx)
                                           ((1 *))
                                           x-%offset%))
                    (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                   (f2cl-lib:int-add i 1))
                                  ((> i
                                      (min (the fixnum n)
                                           (the fixnum
                                                (f2cl-lib:int-add j k))))
                                   nil)
                      (tagbody
                        (setf (f2cl-lib:fref x-%data%
                                             (ix)
                                             ((1 *))
                                             x-%offset%)
                              (-
                               (f2cl-lib:fref x-%data%
                                              (ix)
                                              ((1 *))
                                              x-%offset%)
                               (* temp
                                  (f2cl-lib:fref a-%data%
                                                 ((f2cl-lib:int-add l i)
                                                  j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))))
                        (setf ix (f2cl-lib:int-add ix incx))))))
                (setf jx (f2cl-lib:int-add jx incx)))))))))
           (t
```

```
(cond
  ((char-equal uplo #\U)
   (setf kplus1 (f2cl-lib:int-add k 1))
   (cond
     ((= incx 1)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (setf temp
                  (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
          (setf l (f2cl-lib:int-sub kplus1 j))
          (f2cl-lib:fdo (i
                         (max (the fixnum 1)
                              (the fixnum
                                   (f2cl-lib:int-add j
                                                     (f2cl-lib:int-sub
                                                      k))))
                         (f2cl-lib:int-add i 1))
                        ((> i
                            (f2cl-lib:int-add j
                                              (f2cl-lib:int-sub 1)))
                         nil)
            (tagbody
              (setf temp
                      (- temp
                         (*
                          (f2cl-lib:fref a-%data%
                                         ((f2cl-lib:int-add l i) j)
                                         ((1 lda) (1 *))
                                         a-%offset%)
                          (f2cl-lib:fref x-%data%
                                         (i)
                                         ((1 *))
                                         x-%offset%))))))
          (if nounit
              (setf temp
                      (/ temp
                         (f2cl-lib:fref a-%data%
                                        (kplus1 j)
                                        ((1 lda) (1 *))
                                        a-%offset%))))
          (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                  temp))))
     (t
      (setf jx kx)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
```

```
                    ((> j n) nil)
                  (tagbody
                    (setf temp
                            (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
                    (setf ix kx)
                    (setf l (f2cl-lib:int-sub kplus1 j))
                    (f2cl-lib:fdo (i
                                   (max (the fixnum 1)
                                        (the fixnum
                                             (f2cl-lib:int-add j
                                                               (f2cl-lib:int-sub
                                                                k))))
                                   (f2cl-lib:int-add i 1))
                                  ((> i
                                      (f2cl-lib:int-add j
                                                        (f2cl-lib:int-sub 1)))
                                   nil)
                      (tagbody
                        (setf temp
                                (- temp
                                   (*
                                    (f2cl-lib:fref a-%data%
                                                   ((f2cl-lib:int-add l i) j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%)
                                    (f2cl-lib:fref x-%data%
                                                   (ix)
                                                   ((1 *))
                                                   x-%offset%))))
                        (setf ix (f2cl-lib:int-add ix incx))))
                    (if nounit
                        (setf temp
                                (/ temp
                                   (f2cl-lib:fref a-%data%
                                                  (kplus1 j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))))
                    (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                            temp)
                    (setf jx (f2cl-lib:int-add jx incx))
                    (if (> j k) (setf kx (f2cl-lib:int-add kx incx))))))))
            (t
             (cond
               ((= incx 1)
                (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                              ((> j 1) nil)
```

```
      (tagbody
        (setf temp
                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
        (setf l (f2cl-lib:int-sub 1 j))
        (f2cl-lib:fdo (i
                       (min (the fixnum n)
                            (the fixnum
                                 (f2cl-lib:int-add j k)))
                       (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                      ((> i (f2cl-lib:int-add j 1)) nil)
          (tagbody
            (setf temp
                    (- temp
                       (*
                        (f2cl-lib:fref a-%data%
                                       ((f2cl-lib:int-add l i) j)
                                       ((1 lda) (1 *))
                                       a-%offset%)
                        (f2cl-lib:fref x-%data%
                                       (i)
                                       ((1 *))
                                       x-%offset%))))))
        (if nounit
            (setf temp
                    (/ temp
                       (f2cl-lib:fref a-%data%
                                      (1 j)
                                      ((1 lda) (1 *))
                                      a-%offset%))))
        (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                temp))))
     (t
      (setf kx
              (f2cl-lib:int-add kx
                                (f2cl-lib:int-mul
                                 (f2cl-lib:int-sub n 1)
                                 incx)))
      (setf jx kx)
      (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                    ((> j 1) nil)
        (tagbody
          (setf temp
                  (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
          (setf ix kx)
          (setf l (f2cl-lib:int-sub 1 j))
          (f2cl-lib:fdo (i
```

```
                                    (min (the fixnum n)
                                         (the fixnum
                                              (f2cl-lib:int-add j k)))
                                    (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                                   ((> i (f2cl-lib:int-add j 1)) nil)
                             (tagbody
                               (setf temp
                                       (- temp
                                          (*
                                           (f2cl-lib:fref a-%data%
                                                          ((f2cl-lib:int-add l i) j)
                                                          ((1 lda) (1 *))
                                                          a-%offset%)
                                           (f2cl-lib:fref x-%data%
                                                          (ix)
                                                          ((1 *))
                                                          x-%offset%))))
                                  (setf ix (f2cl-lib:int-sub ix incx))))
                            (if nounit
                                (setf temp
                                       (/ temp
                                          (f2cl-lib:fref a-%data%
                                                         (1 j)
                                                         ((1 lda) (1 *))
                                                         a-%offset%))))
                            (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                                  temp)
                            (setf jx (f2cl-lib:int-sub jx incx))
                            (if (>= (f2cl-lib:int-sub n j) k)
                                (setf kx (f2cl-lib:int-sub kx incx)))))))))
  end_label
       (return (values nil nil nil nil nil nil nil nil nil))))))
```

5.13 dtpmv BLAS

⟨*dtpmv.input*⟩≡

```
)set break resume
)sys rm -f dtpmv.output
)spool dtpmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dtpmv.help*⟩≡

```
 ======================================================================
 dtpmv examples
 ======================================================================

 ======================================================================
 Man Page Details
 ======================================================================

  NAME
       DTPMV - perform one of the matrix-vector operations   x :=
       A*x, or x := A'*x,

  SYNOPSIS
       SUBROUTINE DTPMV ( UPLO, TRANS, DIAG, N, AP, X, INCX )

           INTEGER      INCX, N

           CHARACTER*1  DIAG, TRANS, UPLO

           DOUBLE       PRECISION AP( * ), X( * )

  PURPOSE
       DTPMV  performs one of the matrix-vector operations

       where x is an n element vector and  A is an n by n unit, or
       non-unit, upper or lower triangular matrix, supplied in
       packed form.

  PARAMETERS
       UPLO   - CHARACTER*1.
              On entry, UPLO specifies whether the matrix is an
              upper or lower triangular matrix as follows:

              UPLO = 'U' or 'u'   A is an upper triangular matrix.

              UPLO = 'L' or 'l'   A is a lower triangular matrix.

              Unchanged on exit.

       TRANS  - CHARACTER*1.
              On entry, TRANS specifies the operation to be per-
              formed as follows:

              TRANS = 'N' or 'n'   x := A*x.
```

```
              TRANS = 'T' or 't'   x := A'*x.

              TRANS = 'C' or 'c'   x := A'*x.

              Unchanged on exit.

       DIAG   - CHARACTER*1.
              On entry, DIAG specifies whether or not A is unit
              triangular as follows:

              DIAG = 'U' or 'u'   A is assumed to be unit triangu-
              lar.

              DIAG = 'N' or 'n'   A is not assumed to be unit

              triangular.

              Unchanged on exit.

       N      - INTEGER.
              On entry, N specifies the order of the matrix A.  N
              must be at least zero.  Unchanged on exit.

       AP     - DOUBLE PRECISION array of DIMENSION at least
              ( ( n*( n + 1 ) )/2 ).  Before entry with  UPLO = 'U'
              or 'u', the array AP must contain the upper triangu-
              lar matrix packed sequentially, column by column, so
              that AP( 1 ) contains a( 1, 1 ), AP( 2 ) and AP( 3 )
              contain a( 1, 2 ) and a( 2, 2 ) respectively, and so
              on.  Before entry with UPLO = 'L' or 'l', the array
              AP must contain the lower triangular matrix packed
              sequentially, column by column, so that AP( 1 ) con-
              tains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1
              ) and a( 3, 1 ) respectively, and so on.  Note that
              when  DIAG = 'U' or 'u', the diagonal elements of A
              are not referenced, but are assumed to be unity.
              Unchanged on exit.

       X      - DOUBLE PRECISION array of dimension at least
              ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
              incremented array X must contain the n element vector
              x. On exit, X is overwritten with the transformed
              vector x.

       INCX   - INTEGER.
              On entry, INCX specifies the increment for the ele-
```

```
ments of X. INCX must not be zero.  Unchanged on
exit.
```

⟨*BLAS 2 dtpmv*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dtpmv (uplo trans diag n ap x incx)
    (declare (type (simple-array double-float (*)) x ap)
             (type fixnum incx n)
             (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (diag character diag-%data% diag-%offset%)
         (ap double-float ap-%data% ap-%offset%)
         (x double-float x-%data% x-%offset%))
      (prog ((nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0) (k 0) (kk 0)
             (kx 0) (temp 0.0))
        (declare (type (member t nil) nounit)
                 (type fixnum i info ix j jx k kk kx)
                 (type (double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 2))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((= incx 0)
           (setf info 7)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DTPMV" info)
           (go end_label)))
        (if (= n 0) (go end_label))
        (setf nounit (char-equal diag #\N))
        (cond
          ((<= incx 0)
           (setf kx
                 (f2cl-lib:int-sub 1
                                   (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                     incx))))
```

```
  ((/= incx 1)
   (setf kx 1)))
(cond
  ((char-equal trans #\N)
   (cond
     ((char-equal uplo #\U)
      (setf kk 1)
      (cond
        ((= incx 1)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (cond
               ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                (setf temp
                        (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                (setf k kk)
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                              ((> i
                                  (f2cl-lib:int-add j
                                                    (f2cl-lib:int-sub
                                                     1)))
                               nil)
                  (tagbody
                    (setf (f2cl-lib:fref x-%data%
                                         (i)
                                         ((1 *))
                                         x-%offset%)
                          (+
                           (f2cl-lib:fref x-%data%
                                          (i)
                                          ((1 *))
                                          x-%offset%)
                           (* temp
                              (f2cl-lib:fref ap-%data%
                                             (k)
                                             ((1 *))
                                             ap-%offset%))))
                    (setf k (f2cl-lib:int-add k 1))))
                (if nounit
                    (setf (f2cl-lib:fref x-%data%
                                         (j)
                                         ((1 *))
                                         x-%offset%)
                          (*
                           (f2cl-lib:fref x-%data%
```

```
                                        (j)
                                        ((1 *))
                                        x-%offset%)
                              (f2cl-lib:fref ap-%data%
                                        ((f2cl-lib:int-sub
                                          (f2cl-lib:int-add kk j)
                                          1))
                                        ((1 *))
                                        ap-%offset%))))))
            (setf kk (f2cl-lib:int-add kk j)))))
        (t
         (setf jx kx)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (cond
               ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                (setf temp
                        (f2cl-lib:fref x-%data%
                                       (jx)
                                       ((1 *))
                                       x-%offset%))
                (setf ix kx)
                (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
                              ((> k
                                  (f2cl-lib:int-add kk
                                                    j
                                                    (f2cl-lib:int-sub
                                                     2)))
                               nil)
                  (tagbody
                    (setf (f2cl-lib:fref x-%data%
                                         (ix)
                                         ((1 *))
                                         x-%offset%)
                          (+
                           (f2cl-lib:fref x-%data%
                                         (ix)
                                         ((1 *))
                                         x-%offset%)
                           (* temp
                              (f2cl-lib:fref ap-%data%
                                             (k)
                                             ((1 *))
                                             ap-%offset%))))
                    (setf ix (f2cl-lib:int-add ix incx))))
```

```
                    (if nounit
                        (setf (f2cl-lib:fref x-%data%
                                             (jx)
                                             ((1 *))
                                             x-%offset%)
                              (*
                               (f2cl-lib:fref x-%data%
                                              (jx)
                                              ((1 *))
                                              x-%offset%)
                               (f2cl-lib:fref ap-%data%
                                              ((f2cl-lib:int-sub
                                                (f2cl-lib:int-add kk j)
                                                1))
                                              ((1 *))
                                              ap-%offset%))))))
                      (setf jx (f2cl-lib:int-add jx incx))
                      (setf kk (f2cl-lib:int-add kk j)))))))
          (t
           (setf kk (the fixnum (truncate (* n (+ n 1)) 2)))
           (cond
             ((= incx 1)
              (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                            ((> j 1) nil)
                (tagbody
                  (cond
                    ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                     (setf temp
                             (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                     (setf k kk)
                     (f2cl-lib:fdo (i n
                                    (f2cl-lib:int-add i
                                                      (f2cl-lib:int-sub 1)))
                                   ((> i (f2cl-lib:int-add j 1)) nil)
                       (tagbody
                         (setf (f2cl-lib:fref x-%data%
                                              (i)
                                              ((1 *))
                                              x-%offset%)
                               (+
                                (f2cl-lib:fref x-%data%
                                               (i)
                                               ((1 *))
                                               x-%offset%)
                                (* temp
                                   (f2cl-lib:fref ap-%data%
```

```
                                          (k)
                                          ((1 *))
                                          ap-%offset%))))
                    (setf k (f2cl-lib:int-sub k 1))))
                (if nounit
                    (setf (f2cl-lib:fref x-%data%
                                         (j)
                                         ((1 *))
                                         x-%offset%)
                          (*
                           (f2cl-lib:fref x-%data%
                                          (j)
                                          ((1 *))
                                          x-%offset%)
                           (f2cl-lib:fref ap-%data%
                                          ((f2cl-lib:int-add
                                            (f2cl-lib:int-sub kk n)
                                            j))
                                          ((1 *))
                                          ap-%offset%))))))
            (setf kk
                  (f2cl-lib:int-sub kk
                                    (f2cl-lib:int-add
                                     (f2cl-lib:int-sub n j)
                                     1))))))
          (t
           (setf kx
                 (f2cl-lib:int-add kx
                                   (f2cl-lib:int-mul
                                    (f2cl-lib:int-sub n 1)
                                    incx)))
           (setf jx kx)
           (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                         ((> j 1) nil)
             (tagbody
               (cond
                 ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                  (setf temp
                        (f2cl-lib:fref x-%data%
                                       (jx)
                                       ((1 *))
                                       x-%offset%))
                  (setf ix kx)
                  (f2cl-lib:fdo (k kk
                                 (f2cl-lib:int-add k
                                                   (f2cl-lib:int-sub 1)))
```

```
                ((> k
                    (f2cl-lib:int-add kk
                                      (f2cl-lib:int-sub
                                       (f2cl-lib:int-add
                                        n
                                        (f2cl-lib:int-sub
                                         (f2cl-lib:int-add
                                          j
                                          1))))))
                 nil)
           (tagbody
             (setf (f2cl-lib:fref x-%data%
                                  (ix)
                                  ((1 *))
                                  x-%offset%)
                   (+
                    (f2cl-lib:fref x-%data%
                                   (ix)
                                   ((1 *))
                                   x-%offset%)
                    (* temp
                       (f2cl-lib:fref ap-%data%
                                      (k)
                                      ((1 *))
                                      ap-%offset%))))
             (setf ix (f2cl-lib:int-sub ix incx))))
          (if nounit
              (setf (f2cl-lib:fref x-%data%
                                   (jx)
                                   ((1 *))
                                   x-%offset%)
                    (*
                     (f2cl-lib:fref x-%data%
                                    (jx)
                                    ((1 *))
                                    x-%offset%)
                     (f2cl-lib:fref ap-%data%
                                    ((f2cl-lib:int-add
                                      (f2cl-lib:int-sub kk n)
                                      j))
                                    ((1 *))
                                    ap-%offset%))))))
      (setf jx (f2cl-lib:int-sub jx incx))
      (setf kk
            (f2cl-lib:int-sub kk
                              (f2cl-lib:int-add
```

```
                                          (f2cl-lib:int-sub n j)
                                          1)))))))))))
            (t
             (cond
               ((char-equal uplo #\U)
                (setf kk (the fixnum (truncate (* n (+ n 1)) 2)))
                (cond
                  ((= incx 1)
                   (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                 ((> j 1) nil)
                     (tagbody
                       (setf temp
                               (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                       (if nounit
                           (setf temp
                                   (* temp
                                      (f2cl-lib:fref ap-%data%
                                                     (kk)
                                                     ((1 *))
                                                     ap-%offset%))))
                       (setf k (f2cl-lib:int-sub kk 1))
                       (f2cl-lib:fdo (i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))
                                        (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                                     ((> i 1) nil)
                         (tagbody
                           (setf temp
                                   (+ temp
                                      (*
                                       (f2cl-lib:fref ap-%data%
                                                      (k)
                                                      ((1 *))
                                                      ap-%offset%)
                                       (f2cl-lib:fref x-%data%
                                                      (i)
                                                      ((1 *))
                                                      x-%offset%))))
                           (setf k (f2cl-lib:int-sub k 1))))
                       (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                               temp)
                       (setf kk (f2cl-lib:int-sub kk j)))))
                  (t
                   (setf jx
                           (f2cl-lib:int-add kx
                                             (f2cl-lib:int-mul
                                              (f2cl-lib:int-sub n 1)
                                              incx)))
```

```
    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                  ((> j 1) nil)
      (tagbody
        (setf temp
                (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
        (setf ix jx)
        (if nounit
            (setf temp
                    (* temp
                       (f2cl-lib:fref ap-%data%
                                      (kk)
                                      ((1 *))
                                      ap-%offset%))))
        (f2cl-lib:fdo (k
                       (f2cl-lib:int-add kk (f2cl-lib:int-sub 1))
                       (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
                      ((> k
                          (f2cl-lib:int-add kk
                                            (f2cl-lib:int-sub j)
                                            1))
                       nil)
          (tagbody
            (setf ix (f2cl-lib:int-sub ix incx))
            (setf temp
                    (+ temp
                       (*
                        (f2cl-lib:fref ap-%data%
                                       (k)
                                       ((1 *))
                                       ap-%offset%)
                        (f2cl-lib:fref x-%data%
                                       (ix)
                                       ((1 *))
                                       x-%offset%))))))
        (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                temp)
        (setf jx (f2cl-lib:int-sub jx incx))
        (setf kk (f2cl-lib:int-sub kk j)))))))
(t
 (setf kk 1)
 (cond
   ((= incx 1)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  ((> j n) nil)
      (tagbody
        (setf temp
```

```
                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
            (if nounit
                (setf temp
                        (* temp
                           (f2cl-lib:fref ap-%data%
                                          (kk)
                                          ((1 *))
                                          ap-%offset%))))
            (setf k (f2cl-lib:int-add kk 1))
            (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                             (f2cl-lib:int-add i 1))
                          ((> i n) nil)
              (tagbody
                (setf temp
                        (+ temp
                           (*
                            (f2cl-lib:fref ap-%data%
                                           (k)
                                           ((1 *))
                                           ap-%offset%)
                            (f2cl-lib:fref x-%data%
                                           (i)
                                           ((1 *))
                                           x-%offset%))))
                (setf k (f2cl-lib:int-add k 1))))
            (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                    temp)
            (setf kk
                    (f2cl-lib:int-add kk
                                      (f2cl-lib:int-add
                                       (f2cl-lib:int-sub n j)
                                       1))))))
          (t
           (setf jx kx)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf temp
                       (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
               (setf ix jx)
               (if nounit
                   (setf temp
                           (* temp
                              (f2cl-lib:fref ap-%data%
                                             (kk)
                                             ((1 *))
```

```
                                          ap-%offset%))))
                        (f2cl-lib:fdo (k (f2cl-lib:int-add kk 1)
                                       (f2cl-lib:int-add k 1))
                                      ((> k
                                          (f2cl-lib:int-add kk
                                                            n
                                                            (f2cl-lib:int-sub j)))
                                       nil)
                          (tagbody
                            (setf ix (f2cl-lib:int-add ix incx))
                            (setf temp
                                    (+ temp
                                       (*
                                        (f2cl-lib:fref ap-%data%
                                                       (k)
                                                       ((1 *))
                                                       ap-%offset%)
                                        (f2cl-lib:fref x-%data%
                                                       (ix)
                                                       ((1 *))
                                                       x-%offset%))))))
                        (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                                temp)
                        (setf jx (f2cl-lib:int-add jx incx))
                        (setf kk
                                (f2cl-lib:int-add kk
                                                  (f2cl-lib:int-add
                                                   (f2cl-lib:int-sub n j)
                                                   1)))))))))
   end_label
          (return (values nil nil nil nil nil nil nil))))))
```

5.14 dtpsv BLAS

⟨*dtpsv.input*⟩≡

```
)set break resume
)sys rm -f dtpsv.output
)spool dtpsv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dtpsv.help*⟩≡

```
  ==========================================================================
  dtpsv examples
  ==========================================================================

  ==========================================================================
  Man Page Details
  ==========================================================================

   NAME
        DTPSV - solve one of the systems of equations   A*x = b, or
        A'*x = b,

   SYNOPSIS
        SUBROUTINE DTPSV ( UPLO, TRANS, DIAG, N, AP, X, INCX )

            INTEGER      INCX, N

            CHARACTER*1  DIAG, TRANS, UPLO

            DOUBLE       PRECISION AP( * ), X( * )

   PURPOSE
        DTPSV  solves one of the systems of equations

        where b and x are n element vectors and A is an n by n unit,
        or non-unit, upper or lower triangular matrix, supplied in
        packed form.

        No test for singularity or near-singularity is included in
        this routine. Such tests must be performed before calling
        this routine.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On entry, UPLO specifies whether the matrix is an
                 upper or lower triangular matrix as follows:

                 UPLO = 'U' or 'u'   A is an upper triangular matrix.

                 UPLO = 'L' or 'l'   A is a lower triangular matrix.

                 Unchanged on exit.

        TRANS  - CHARACTER*1.
                 On entry, TRANS specifies the equations to be solved
```

```
        as follows:

        TRANS = 'N' or 'n'   A*x = b.

        TRANS = 'T' or 't'   A'*x = b.

        TRANS = 'C' or 'c'   A'*x = b.

        Unchanged on exit.

DIAG    - CHARACTER*1.
        On entry, DIAG specifies whether or not A is unit
        triangular as follows:

        DIAG = 'U' or 'u'   A is assumed to be unit triangu-
        lar.

        DIAG = 'N' or 'n'   A is not assumed to be unit tri-
        angular.

        Unchanged on exit.

N       - INTEGER.
        On entry, N specifies the order of the matrix A.  N
        must be at least zero.  Unchanged on exit.

AP      - DOUBLE PRECISION array of DIMENSION at least
        ( ( n*( n + 1 ) )/2 ).  Before entry with  UPLO = 'U'
        or 'u', the array AP must contain the upper triangu-
        lar matrix packed sequentially, column by column, so
        that AP( 1 ) contains a( 1, 1 ), AP( 2 ) and AP( 3 )
        contain a( 1, 2 ) and a( 2, 2 ) respectively, and so
        on.  Before entry with UPLO = 'L' or 'l', the array
        AP must contain the lower triangular matrix packed
        sequentially, column by column, so that AP( 1 ) con-
        tains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1
        ) and a( 3, 1 ) respectively, and so on.  Note that
        when  DIAG = 'U' or 'u', the diagonal elements of A
        are not referenced, but are assumed to be unity.
        Unchanged on exit.

X       - DOUBLE PRECISION array of dimension at least
        ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
        incremented array X must contain the n element
        right-hand side vector b. On exit, X is overwritten
        with the solution vector x.
```

```
INCX   - INTEGER.
         On entry, INCX specifies the increment for the ele-
         ments of X. INCX must not be zero.  Unchanged on
         exit.
```

⟨*BLAS 2 dtpsv*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dtpsv (uplo trans diag n ap x incx)
    (declare (type (simple-array double-float (*)) x ap)
             (type fixnum incx n)
             (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (diag character diag-%data% diag-%offset%)
         (ap double-float ap-%data% ap-%offset%)
         (x double-float x-%data% x-%offset%))
      (prog ((nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0) (k 0) (kk 0)
             (kx 0) (temp 0.0))
        (declare (type (member t nil) nounit)
                 (type fixnum i info ix j jx k kk kx)
                 (type (double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 2))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((= incx 0)
           (setf info 7)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DTPSV" info)
           (go end_label)))
        (if (= n 0) (go end_label))
        (setf nounit (char-equal diag #\N))
        (cond
          ((<= incx 0)
           (setf kx
                 (f2cl-lib:int-sub 1
                                   (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                     incx))))
```

```
  ((/= incx 1)
   (setf kx 1)))
(cond
  ((char-equal trans #\N)
   (cond
     ((char-equal uplo #\U)
      (setf kk (the fixnum (truncate (* n (+ n 1)) 2)))
      (cond
        ((= incx 1)
         (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                       ((> j 1) nil)
           (tagbody
             (cond
               ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                (if nounit
                    (setf (f2cl-lib:fref x-%data%
                                         (j)
                                         ((1 *))
                                         x-%offset%)
                          (/
                           (f2cl-lib:fref x-%data%
                                          (j)
                                          ((1 *))
                                          x-%offset%)
                           (f2cl-lib:fref ap-%data%
                                          (kk)
                                          ((1 *))
                                          ap-%offset%))))
                (setf temp
                        (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                (setf k (f2cl-lib:int-sub kk 1))
                (f2cl-lib:fdo (i
                               (f2cl-lib:int-add j
                                                 (f2cl-lib:int-sub 1))
                               (f2cl-lib:int-add i
                                                 (f2cl-lib:int-sub 1)))
                              ((> i 1) nil)
                  (tagbody
                    (setf (f2cl-lib:fref x-%data%
                                         (i)
                                         ((1 *))
                                         x-%offset%)
                          (-
                           (f2cl-lib:fref x-%data%
                                          (i)
                                          ((1 *))
```

```
                                          x-%offset%)
                              (* temp
                                 (f2cl-lib:fref ap-%data%
                                                (k)
                                                ((1 *))
                                                ap-%offset%))))
                        (setf k (f2cl-lib:int-sub k 1))))))
                    (setf kk (f2cl-lib:int-sub kk j)))))
                (t
                 (setf jx
                       (f2cl-lib:int-add kx
                                         (f2cl-lib:int-mul
                                          (f2cl-lib:int-sub n 1)
                                          incx)))
                 (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                               ((> j 1) nil)
                   (tagbody
                     (cond
                       ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                        (if nounit
                            (setf (f2cl-lib:fref x-%data%
                                                 (jx)
                                                 ((1 *))
                                                 x-%offset%)
                                  (/
                                   (f2cl-lib:fref x-%data%
                                                  (jx)
                                                  ((1 *))
                                                  x-%offset%)
                                   (f2cl-lib:fref ap-%data%
                                                  (kk)
                                                  ((1 *))
                                                  ap-%offset%))))
                        (setf temp
                              (f2cl-lib:fref x-%data%
                                             (jx)
                                             ((1 *))
                                             x-%offset%))
                        (setf ix jx)
                        (f2cl-lib:fdo (k
                                       (f2cl-lib:int-add kk
                                                         (f2cl-lib:int-sub 1))
                                       (f2cl-lib:int-add k
                                                         (f2cl-lib:int-sub 1)))
                                      ((> k
                                          (f2cl-lib:int-add kk
```

```
                                                    (f2cl-lib:int-sub
                                                     j)
                                                    1))
                                  nil)
                            (tagbody
                              (setf ix (f2cl-lib:int-sub ix incx))
                              (setf (f2cl-lib:fref x-%data%
                                                   (ix)
                                                   ((1 *))
                                                   x-%offset%)
                                    (-
                                     (f2cl-lib:fref x-%data%
                                                    (ix)
                                                    ((1 *))
                                                    x-%offset%)
                                     (* temp
                                        (f2cl-lib:fref ap-%data%
                                                       (k)
                                                       ((1 *))
                                                       ap-%offset%))))))))
                    (setf jx (f2cl-lib:int-sub jx incx))
                    (setf kk (f2cl-lib:int-sub kk j)))))))
            (t
             (setf kk 1)
             (cond
               ((= incx 1)
                (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                              ((> j n) nil)
                  (tagbody
                    (cond
                      ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                       (if nounit
                           (setf (f2cl-lib:fref x-%data%
                                                (j)
                                                ((1 *))
                                                x-%offset%)
                                 (/
                                  (f2cl-lib:fref x-%data%
                                                 (j)
                                                 ((1 *))
                                                 x-%offset%)
                                  (f2cl-lib:fref ap-%data%
                                                 (kk)
                                                 ((1 *))
                                                 ap-%offset%))))
                           (setf temp
```

```
                    (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                (setf k (f2cl-lib:int-add kk 1))
                (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                 (f2cl-lib:int-add i 1))
                              ((> i n) nil)
                  (tagbody
                    (setf (f2cl-lib:fref x-%data%
                                         (i)
                                         ((1 *))
                                         x-%offset%)
                          (-
                           (f2cl-lib:fref x-%data%
                                          (i)
                                          ((1 *))
                                          x-%offset%)
                           (* temp
                              (f2cl-lib:fref ap-%data%
                                             (k)
                                             ((1 *))
                                             ap-%offset%))))
                          (setf k (f2cl-lib:int-add k 1))))))
            (setf kk
                  (f2cl-lib:int-add kk
                                    (f2cl-lib:int-add
                                     (f2cl-lib:int-sub n j)
                                     1))))))
          (t
           (setf jx kx)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (cond
                 ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                  (if nounit
                      (setf (f2cl-lib:fref x-%data%
                                           (jx)
                                           ((1 *))
                                           x-%offset%)
                            (/
                             (f2cl-lib:fref x-%data%
                                            (jx)
                                            ((1 *))
                                            x-%offset%)
                             (f2cl-lib:fref ap-%data%
                                            (kk)
                                            ((1 *))
```

```
                                        ap-%offset%))))
                        (setf temp
                                (f2cl-lib:fref x-%data%
                                               (jx)
                                               ((1 *))
                                               x-%offset%))
                        (setf ix jx)
                        (f2cl-lib:fdo (k (f2cl-lib:int-add kk 1)
                                         (f2cl-lib:int-add k 1))
                                      ((> k
                                          (f2cl-lib:int-add kk
                                                            n
                                                            (f2cl-lib:int-sub
                                                             j)))
                                       nil)
                          (tagbody
                            (setf ix (f2cl-lib:int-add ix incx))
                            (setf (f2cl-lib:fref x-%data%
                                                 (ix)
                                                 ((1 *))
                                                 x-%offset%)
                                  (-
                                   (f2cl-lib:fref x-%data%
                                                  (ix)
                                                  ((1 *))
                                                  x-%offset%)
                                   (* temp
                                      (f2cl-lib:fref ap-%data%
                                                     (k)
                                                     ((1 *))
                                                     ap-%offset%))))))))
                    (setf jx (f2cl-lib:int-add jx incx))
                    (setf kk
                            (f2cl-lib:int-add kk
                                              (f2cl-lib:int-add
                                               (f2cl-lib:int-sub n j)
                                               1)))))))))
              (t
               (cond
                 ((char-equal uplo #\U)
                  (setf kk 1)
                  (cond
                    ((= incx 1)
                     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                   ((> j n) nil)
                       (tagbody
```

```
                (setf temp
                        (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                (setf k kk)
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                              ((> i
                                  (f2cl-lib:int-add j
                                                    (f2cl-lib:int-sub 1)))
                               nil)
                  (tagbody
                    (setf temp
                            (- temp
                               (*
                                (f2cl-lib:fref ap-%data%
                                               (k)
                                               ((1 *))
                                               ap-%offset%)
                                (f2cl-lib:fref x-%data%
                                               (i)
                                               ((1 *))
                                               x-%offset%))))
                    (setf k (f2cl-lib:int-add k 1))))
                (if nounit
                    (setf temp
                            (/ temp
                               (f2cl-lib:fref ap-%data%
                                              ((f2cl-lib:int-sub
                                                (f2cl-lib:int-add kk j)
                                                1))
                                              ((1 *))
                                              ap-%offset%))))
                (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                        temp)
                (setf kk (f2cl-lib:int-add kk j)))))
           (t
            (setf jx kx)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                          ((> j n) nil)
              (tagbody
                (setf temp
                        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
                (setf ix kx)
                (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
                              ((> k
                                  (f2cl-lib:int-add kk
                                                    j
                                                    (f2cl-lib:int-sub 2)))
```

```
                        nil)
                  (tagbody
                    (setf temp
                            (- temp
                               (*
                                (f2cl-lib:fref ap-%data%
                                               (k)
                                               ((1 *))
                                               ap-%offset%)
                                (f2cl-lib:fref x-%data%
                                               (ix)
                                               ((1 *))
                                               x-%offset%))))
                    (setf ix (f2cl-lib:int-add ix incx))))
                (if nounit
                    (setf temp
                            (/ temp
                               (f2cl-lib:fref ap-%data%
                                              ((f2cl-lib:int-sub
                                                (f2cl-lib:int-add kk j)
                                                1))
                                              ((1 *))
                                              ap-%offset%))))
                (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                        temp)
                (setf jx (f2cl-lib:int-add jx incx))
                (setf kk (f2cl-lib:int-add kk j)))))))
       (t
        (setf kk (the fixnum (truncate (* n (+ n 1)) 2)))
        (cond
          ((= incx 1)
           (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                         ((> j 1) nil)
             (tagbody
               (setf temp
                       (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
               (setf k kk)
               (f2cl-lib:fdo (i n
                              (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                             ((> i (f2cl-lib:int-add j 1)) nil)
                 (tagbody
                   (setf temp
                           (- temp
                              (*
                               (f2cl-lib:fref ap-%data%
                                              (k)
```

```
                                        ((1 *))
                                        ap-%offset%)
                                (f2cl-lib:fref x-%data%
                                        (i)
                                        ((1 *))
                                        x-%offset%))))
                    (setf k (f2cl-lib:int-sub k 1))))
                (if nounit
                    (setf temp
                            (/ temp
                               (f2cl-lib:fref ap-%data%
                                              ((f2cl-lib:int-add
                                                (f2cl-lib:int-sub kk n)
                                                j))
                                              ((1 *))
                                              ap-%offset%))))
                (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                        temp)
                (setf kk
                        (f2cl-lib:int-sub kk
                                          (f2cl-lib:int-add
                                           (f2cl-lib:int-sub n j)
                                           1))))))
            (t
             (setf kx
                    (f2cl-lib:int-add kx
                                      (f2cl-lib:int-mul
                                       (f2cl-lib:int-sub n 1)
                                       incx)))
             (setf jx kx)
             (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                           ((> j 1) nil)
               (tagbody
                 (setf temp
                        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
                 (setf ix kx)
                 (f2cl-lib:fdo (k kk
                                (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
                               ((> k
                                   (f2cl-lib:int-add kk
                                     (f2cl-lib:int-sub
                                      (f2cl-lib:int-add n
                                       (f2cl-lib:int-sub
                                        (f2cl-lib:int-add
                                          j
                                          1))))))
```

```
                                nil)
                          (tagbody
                            (setf temp
                                    (- temp
                                       (*
                                        (f2cl-lib:fref ap-%data%
                                                       (k)
                                                       ((1 *))
                                                       ap-%offset%)
                                        (f2cl-lib:fref x-%data%
                                                       (ix)
                                                       ((1 *))
                                                       x-%offset%))))
                            (setf ix (f2cl-lib:int-sub ix incx))))
                          (if nounit
                              (setf temp
                                      (/ temp
                                         (f2cl-lib:fref ap-%data%
                                                        ((f2cl-lib:int-add
                                                          (f2cl-lib:int-sub kk n)
                                                          j))
                                                        ((1 *))
                                                        ap-%offset%))))
                          (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                                  temp)
                          (setf jx (f2cl-lib:int-sub jx incx))
                          (setf kk
                                  (f2cl-lib:int-sub kk
                                                    (f2cl-lib:int-add
                                                     (f2cl-lib:int-sub n j)
                                                     1)))))))))
    end_label
           (return (values nil nil nil nil nil nil nil))))))
```

5.15 dtrmv BLAS

⟨*dtrmv.input*⟩≡

```
)set break resume
)sys rm -f dtrmv.output
)spool dtrmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dtrmv.help*⟩≡

```
 ======================================================================
 dtrmv examples
 ======================================================================

 ======================================================================
 Man Page Details
 ======================================================================

  NAME
       DTRMV - perform one of the matrix-vector operations   x :=
       A*x, or x := A'*x,

  SYNOPSIS
       SUBROUTINE DTRMV ( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )

           INTEGER      INCX, LDA, N

           CHARACTER*1  DIAG, TRANS, UPLO

           DOUBLE       PRECISION A( LDA, * ), X( * )

  PURPOSE
       DTRMV  performs one of the matrix-vector operations

       where x is an n element vector and  A is an n by n unit, or
       non-unit, upper or lower triangular matrix.

  PARAMETERS
       UPLO   - CHARACTER*1.
              On entry, UPLO specifies whether the matrix is an
              upper or lower triangular matrix as follows:

              UPLO = 'U' or 'u'   A is an upper triangular matrix.

              UPLO = 'L' or 'l'   A is a lower triangular matrix.

              Unchanged on exit.

       TRANS  - CHARACTER*1.
              On entry, TRANS specifies the operation to be per-
              formed as follows:

              TRANS = 'N' or 'n'   x := A*x.

              TRANS = 'T' or 't'   x := A'*x.
```

```
          TRANS = 'C' or 'c'   x := A'*x.

          Unchanged on exit.

DIAG   - CHARACTER*1.
          On entry, DIAG specifies whether or not A is unit
          triangular as follows:

          DIAG = 'U' or 'u'   A is assumed to be unit triangu-
          lar.

          DIAG = 'N' or 'n'   A is not assumed to be unit tri-
          angular.

          Unchanged on exit.

N      - INTEGER.
          On entry, N specifies the order of the matrix A.  N
          must be at least zero.  Unchanged on exit.

A      - DOUBLE PRECISION array of DIMENSION ( LDA, n ).
          Before entry with  UPLO = 'U' or 'u', the leading n
          by n upper triangular part of the array A must con-
          tain the upper triangular matrix and the strictly
          lower triangular part of A is not referenced.  Before
          entry with UPLO = 'L' or 'l', the leading n by n
          lower triangular part of the array A must contain the
          lower triangular matrix and the strictly upper tri-
          angular part of A is not referenced.  Note that when
          DIAG = 'U' or 'u', the diagonal elements of A are not
          referenced either, but are assumed to be unity.
          Unchanged on exit.

LDA    - INTEGER.
          On entry, LDA specifies the first dimension of A as
          declared in the calling (sub) program. LDA must be at
          least max( 1, n ).  Unchanged on exit.

X      - DOUBLE PRECISION array of dimension at least
          ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
          incremented array X must contain the n element vector
          x. On exit, X is overwritten with the transformed
          vector x.

INCX   - INTEGER.
```

```
          On entry, INCX specifies the increment for the ele-
          ments of X. INCX must not be zero.  Unchanged on
          exit.
```

⟨*BLAS 2 dtrmv*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dtrmv (uplo trans diag n a lda x incx)
    (declare (type (simple-array double-float (*)) x a)
             (type fixnum incx lda n)
             (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (diag character diag-%data% diag-%offset%)
         (a double-float a-%data% a-%offset%)
         (x double-float x-%data% x-%offset%))
      (prog ((nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0) (kx 0) (temp 0.0))
        (declare (type (member t nil) nounit)
                 (type fixnum i info ix j jx kx)
                 (type (double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 2))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info 6))
          ((= incx 0)
           (setf info 8)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DTRMV" info)
           (go end_label)))
        (if (= n 0) (go end_label))
        (setf nounit (char-equal diag #\N))
        (cond
          ((<= incx 0)
           (setf kx
                 (f2cl-lib:int-sub 1
                                   (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
```

```
                                                    incx))))
  ((/= incx 1)
   (setf kx 1)))
(cond
  ((char-equal trans #\N)
   (cond
     ((char-equal uplo #\U)
      (cond
        ((= incx 1)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (cond
               ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                (setf temp
                        (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                              ((> i
                                  (f2cl-lib:int-add j
                                                    (f2cl-lib:int-sub
                                                     1)))
                               nil)
                  (tagbody
                    (setf (f2cl-lib:fref x-%data%
                                         (i)
                                         ((1 *))
                                         x-%offset%)
                                  (+
                                   (f2cl-lib:fref x-%data%
                                                  (i)
                                                  ((1 *))
                                                  x-%offset%)
                                   (* temp
                                      (f2cl-lib:fref a-%data%
                                                     (i j)
                                                     ((1 lda) (1 *))
                                                     a-%offset%))))))
                (if nounit
                    (setf (f2cl-lib:fref x-%data%
                                         (j)
                                         ((1 *))
                                         x-%offset%)
                                  (*
                                   (f2cl-lib:fref x-%data%
                                                  (j)
                                                  ((1 *))
```

```
                                        x-%offset%)
                              (f2cl-lib:fref a-%data%
                                        (j j)
                                        ((1 lda) (1 *))
                                        a-%offset%)))))))))
                    (t
                     (setf jx kx)
                     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                   ((> j n) nil)
                       (tagbody
                         (cond
                           ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                            (setf temp
                                  (f2cl-lib:fref x-%data%
                                                (jx)
                                                ((1 *))
                                                x-%offset%))
                            (setf ix kx)
                            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                          ((> i
                                              (f2cl-lib:int-add j
                                                                (f2cl-lib:int-sub
                                                                 1)))
                                           nil)
                              (tagbody
                                (setf (f2cl-lib:fref x-%data%
                                                     (ix)
                                                     ((1 *))
                                                     x-%offset%)
                                      (+
                                       (f2cl-lib:fref x-%data%
                                                      (ix)
                                                      ((1 *))
                                                      x-%offset%)
                                       (* temp
                                          (f2cl-lib:fref a-%data%
                                                         (i j)
                                                         ((1 lda) (1 *))
                                                         a-%offset%))))
                                (setf ix (f2cl-lib:int-add ix incx))))
                            (if nounit
                                (setf (f2cl-lib:fref x-%data%
                                                     (jx)
                                                     ((1 *))
                                                     x-%offset%)
                                      (*
```

```
                                (f2cl-lib:fref x-%data%
                                               (jx)
                                               ((1 *))
                                               x-%offset%)
                                (f2cl-lib:fref a-%data%
                                               (j j)
                                               ((1 lda) (1 *))
                                               a-%offset%))))))
                        (setf jx (f2cl-lib:int-add jx incx)))))))
                (t
                 (cond
                   ((= incx 1)
                    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                  ((> j 1) nil)
                      (tagbody
                        (cond
                          ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                           (setf temp
                                   (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                           (f2cl-lib:fdo (i n
                                           (f2cl-lib:int-add i
                                                             (f2cl-lib:int-sub 1)))
                                         ((> i (f2cl-lib:int-add j 1)) nil)
                             (tagbody
                               (setf (f2cl-lib:fref x-%data%
                                                    (i)
                                                    ((1 *))
                                                    x-%offset%)
                                     (+
                                      (f2cl-lib:fref x-%data%
                                                     (i)
                                                     ((1 *))
                                                     x-%offset%)
                                      (* temp
                                         (f2cl-lib:fref a-%data%
                                                        (i j)
                                                        ((1 lda) (1 *))
                                                        a-%offset%))))))
                           (if nounit
                               (setf (f2cl-lib:fref x-%data%
                                                    (j)
                                                    ((1 *))
                                                    x-%offset%)
                                     (*
                                      (f2cl-lib:fref x-%data%
                                                     (j)
```

```
                                        ((1 *))
                                        x-%offset%)
                          (f2cl-lib:fref a-%data%
                                        (j j)
                                        ((1 lda) (1 *))
                                        a-%offset%)))))))))
                (t
                 (setf kx
                       (f2cl-lib:int-add kx
                                         (f2cl-lib:int-mul
                                          (f2cl-lib:int-sub n 1)
                                          incx)))
                 (setf jx kx)
                 (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                               ((> j 1) nil)
                   (tagbody
                     (cond
                       ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                        (setf temp
                              (f2cl-lib:fref x-%data%
                                             (jx)
                                             ((1 *))
                                             x-%offset%))
                        (setf ix kx)
                        (f2cl-lib:fdo (i n
                                       (f2cl-lib:int-add i
                                                         (f2cl-lib:int-sub 1)))
                                      ((> i (f2cl-lib:int-add j 1)) nil)
                          (tagbody
                            (setf (f2cl-lib:fref x-%data%
                                                 (ix)
                                                 ((1 *))
                                                 x-%offset%)
                                  (+
                                   (f2cl-lib:fref x-%data%
                                                  (ix)
                                                  ((1 *))
                                                  x-%offset%)
                                   (* temp
                                      (f2cl-lib:fref a-%data%
                                                     (i j)
                                                     ((1 lda) (1 *))
                                                     a-%offset%))))
                            (setf ix (f2cl-lib:int-sub ix incx))))
                        (if nounit
                            (setf (f2cl-lib:fref x-%data%
```

```
                                                (jx)
                                                ((1 *))
                                                x-%offset%)
                                        (*
                                         (f2cl-lib:fref x-%data%
                                                        (jx)
                                                        ((1 *))
                                                        x-%offset%)
                                         (f2cl-lib:fref a-%data%
                                                        (j j)
                                                        ((1 lda) (1 *))
                                                        a-%offset%))))))
                          (setf jx (f2cl-lib:int-sub jx incx)))))))))
                    (t
                     (cond
                       ((char-equal uplo #\U)
                        (cond
                          ((= incx 1)
                           (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                         ((> j 1) nil)
                             (tagbody
                               (setf temp
                                       (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                               (if nounit
                                   (setf temp
                                           (* temp
                                              (f2cl-lib:fref a-%data%
                                                             (j j)
                                                             ((1 lda) (1 *))
                                                             a-%offset%))))
                               (f2cl-lib:fdo (i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))
                                                (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                                             ((> i 1) nil)
                                 (tagbody
                                   (setf temp
                                           (+ temp
                                              (*
                                               (f2cl-lib:fref a-%data%
                                                              (i j)
                                                              ((1 lda) (1 *))
                                                              a-%offset%)
                                               (f2cl-lib:fref x-%data%
                                                              (i)
                                                              ((1 *))
                                                              x-%offset%))))))
                               (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
```

```
                    temp))))
          (t
           (setf jx
                 (f2cl-lib:int-add kx
                                   (f2cl-lib:int-mul
                                    (f2cl-lib:int-sub n 1)
                                    incx)))
           (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                         ((> j 1) nil)
             (tagbody
               (setf temp
                       (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
               (setf ix jx)
               (if nounit
                   (setf temp
                           (* temp
                              (f2cl-lib:fref a-%data%
                                             (j j)
                                             ((1 lda) (1 *))
                                             a-%offset%))))
               (f2cl-lib:fdo (i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))
                              (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                             ((> i 1) nil)
                 (tagbody
                   (setf ix (f2cl-lib:int-sub ix incx))
                   (setf temp
                           (+ temp
                              (*
                               (f2cl-lib:fref a-%data%
                                              (i j)
                                              ((1 lda) (1 *))
                                              a-%offset%)
                               (f2cl-lib:fref x-%data%
                                              (ix)
                                              ((1 *))
                                              x-%offset%))))))
               (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                       temp)
               (setf jx (f2cl-lib:int-sub jx incx)))))))
    (t
     (cond
       ((= incx 1)
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                      ((> j n) nil)
          (tagbody
            (setf temp
```

```
                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
            (if nounit
                (setf temp
                        (* temp
                           (f2cl-lib:fref a-%data%
                                          (j j)
                                          ((1 lda) (1 *))
                                          a-%offset%))))
            (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                             (f2cl-lib:int-add i 1))
                          ((> i n) nil)
              (tagbody
                (setf temp
                        (+ temp
                           (*
                            (f2cl-lib:fref a-%data%
                                           (i j)
                                           ((1 lda) (1 *))
                                           a-%offset%)
                            (f2cl-lib:fref x-%data%
                                           (i)
                                           ((1 *))
                                           x-%offset%))))))
            (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                    temp))))
          (t
           (setf jx kx)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf temp
                       (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
               (setf ix jx)
               (if nounit
                   (setf temp
                           (* temp
                              (f2cl-lib:fref a-%data%
                                             (j j)
                                             ((1 lda) (1 *))
                                             a-%offset%))))
               (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                (f2cl-lib:int-add i 1))
                             ((> i n) nil)
                 (tagbody
                   (setf ix (f2cl-lib:int-add ix incx))
                   (setf temp
```

```
                                    (+ temp
                                       (*
                                        (f2cl-lib:fref a-%data%
                                                       (i j)
                                                       ((1 lda) (1 *))
                                                       a-%offset%)
                                        (f2cl-lib:fref x-%data%
                                                       (ix)
                                                       ((1 *))
                                                       x-%offset%))))))
                          (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                                  temp)
                          (setf jx (f2cl-lib:int-add jx incx)))))))))
   end_label
          (return (values nil nil nil nil nil nil nil nil))))))
```

5.16 dtrsv BLAS

⟨*dtrsv.input*⟩≡

```
  )set break resume
  )sys rm -f dtrsv.output
  )spool dtrsv.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dtrsv.help*⟩≡

```
 ======================================================================
 dtrsv examples
 ======================================================================

 ======================================================================
 Man Page Details
 ======================================================================

  NAME
       DTRSV - solve one of the systems of equations   A*x = b, or
       A'*x = b,

  SYNOPSIS
       SUBROUTINE DTRSV ( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )

           INTEGER      INCX, LDA, N

           CHARACTER*1  DIAG, TRANS, UPLO

           DOUBLE       PRECISION A( LDA, * ), X( * )

  PURPOSE
       DTRSV  solves one of the systems of equations

       where b and x are n element vectors and A is an n by n unit,
       or non-unit, upper or lower triangular matrix.

       No test for singularity or near-singularity is included in
       this routine. Such tests must be performed before calling
       this routine.

  PARAMETERS
       UPLO   - CHARACTER*1.
              On entry, UPLO specifies whether the matrix is an
              upper or lower triangular matrix as follows:

              UPLO = 'U' or 'u'   A is an upper triangular matrix.

              UPLO = 'L' or 'l'   A is a lower triangular matrix.

              Unchanged on exit.

       TRANS  - CHARACTER*1.
              On entry, TRANS specifies the equations to be solved
              as follows:
```

```
          TRANS = 'N' or 'n'   A*x = b.

          TRANS = 'T' or 't'   A'*x = b.

          TRANS = 'C' or 'c'   A'*x = b.

          Unchanged on exit.

DIAG   - CHARACTER*1.
          On entry, DIAG specifies whether or not A is unit
          triangular as follows:

          DIAG = 'U' or 'u'   A is assumed to be unit

          triangular.

          DIAG = 'N' or 'n'   A is not assumed to be unit tri-
          angular.

          Unchanged on exit.

N      - INTEGER.
          On entry, N specifies the order of the matrix A.  N
          must be at least zero.  Unchanged on exit.

A      - DOUBLE PRECISION array of DIMENSION ( LDA, n ).
          Before entry with  UPLO = 'U' or 'u', the leading n
          by n upper triangular part of the array A must con-
          tain the upper triangular matrix and the strictly
          lower triangular part of A is not referenced.  Before
          entry with UPLO = 'L' or 'l', the leading n by n
          lower triangular part of the array A must contain the
          lower triangular matrix and the strictly upper tri-
          angular part of A is not referenced.  Note that when
          DIAG = 'U' or 'u', the diagonal elements of A are not
          referenced either, but are assumed to be unity.
          Unchanged on exit.

LDA    - INTEGER.
          On entry, LDA specifies the first dimension of A as
          declared in the calling (sub) program. LDA must be at
          least max( 1, n ).  Unchanged on exit.

X      - DOUBLE PRECISION array of dimension at least
          ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
```

```
          incremented array X must contain the n element
          right-hand side vector b. On exit, X is overwritten
          with the solution vector x.

  INCX    - INTEGER.
          On entry, INCX specifies the increment for the ele-
          ments of X. INCX must not be zero.  Unchanged on
          exit.
```

⟨*BLAS 2 dtrsv*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dtrsv (uplo trans diag n a lda x incx)
    (declare (type (simple-array double-float (*)) x a)
             (type fixnum incx lda n)
             (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (diag character diag-%data% diag-%offset%)
         (a double-float a-%data% a-%offset%)
         (x double-float x-%data% x-%offset%))
      (prog ((nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0) (kx 0) (temp 0.0))
        (declare (type (member t nil) nounit)
                 (type fixnum i info ix j jx kx)
                 (type (double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 2))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info 6))
          ((= incx 0)
           (setf info 8)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DTRSV" info)
           (go end_label)))
        (if (= n 0) (go end_label))
        (setf nounit (char-equal diag #\N))
        (cond
          ((<= incx 0)
           (setf kx
                   (f2cl-lib:int-sub 1
                                     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
```

```
                                                    incx))))
  ((/= incx 1)
   (setf kx 1)))
(cond
  ((char-equal trans #\N)
   (cond
     ((char-equal uplo #\U)
      (cond
        ((= incx 1)
         (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                       ((> j 1) nil)
           (tagbody
             (cond
               ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                (if nounit
                    (setf (f2cl-lib:fref x-%data%
                                         (j)
                                         ((1 *))
                                         x-%offset%)
                          (/
                           (f2cl-lib:fref x-%data%
                                          (j)
                                          ((1 *))
                                          x-%offset%)
                           (f2cl-lib:fref a-%data%
                                          (j j)
                                          ((1 lda) (1 *))
                                          a-%offset%))))
                (setf temp
                      (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                (f2cl-lib:fdo (i
                               (f2cl-lib:int-add j
                                                 (f2cl-lib:int-sub 1))
                               (f2cl-lib:int-add i
                                                 (f2cl-lib:int-sub 1)))
                              ((> i 1) nil)
                  (tagbody
                    (setf (f2cl-lib:fref x-%data%
                                         (i)
                                         ((1 *))
                                         x-%offset%)
                          (-
                           (f2cl-lib:fref x-%data%
                                          (i)
                                          ((1 *))
                                          x-%offset%)
```

```
                        (* temp
                           (f2cl-lib:fref a-%data%
                                          (i j)
                                          ((1 lda) (1 *))
                                          a-%offset%)))))))))))
                (t
                 (setf jx
                       (f2cl-lib:int-add kx
                                         (f2cl-lib:int-mul
                                          (f2cl-lib:int-sub n 1)
                                          incx)))
                 (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                               ((> j 1) nil)
                   (tagbody
                     (cond
                       ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                        (if nounit
                            (setf (f2cl-lib:fref x-%data%
                                                 (jx)
                                                 ((1 *))
                                                 x-%offset%)
                                  (/
                                   (f2cl-lib:fref x-%data%
                                                  (jx)
                                                  ((1 *))
                                                  x-%offset%)
                                   (f2cl-lib:fref a-%data%
                                                  (j j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))))
                        (setf temp
                              (f2cl-lib:fref x-%data%
                                             (jx)
                                             ((1 *))
                                             x-%offset%))
                        (setf ix jx)
                        (f2cl-lib:fdo (i
                                       (f2cl-lib:int-add j
                                                         (f2cl-lib:int-sub 1))
                                       (f2cl-lib:int-add i
                                                         (f2cl-lib:int-sub 1)))
                                      ((> i 1) nil)
                          (tagbody
                            (setf ix (f2cl-lib:int-sub ix incx))
                            (setf (f2cl-lib:fref x-%data%
                                                 (ix)
```

```
                                        ((1 *))
                                        x-%offset%)
                                    (-
                                     (f2cl-lib:fref x-%data%
                                                    (ix)
                                                    ((1 *))
                                                    x-%offset%)
                                     (* temp
                                        (f2cl-lib:fref a-%data%
                                                       (i j)
                                                       ((1 lda) (1 *))
                                                       a-%offset%))))))))
                (setf jx (f2cl-lib:int-sub jx incx)))))))
           (t
            (cond
              ((= incx 1)
               (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                             ((> j n) nil)
                 (tagbody
                   (cond
                     ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                      (if nounit
                          (setf (f2cl-lib:fref x-%data%
                                               (j)
                                               ((1 *))
                                               x-%offset%)
                                (/
                                 (f2cl-lib:fref x-%data%
                                                (j)
                                                ((1 *))
                                                x-%offset%)
                                 (f2cl-lib:fref a-%data%
                                                (j j)
                                                ((1 lda) (1 *))
                                                a-%offset%))))
                          (setf temp
                                  (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                          (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                           (f2cl-lib:int-add i 1))
                                        ((> i n) nil)
                            (tagbody
                              (setf (f2cl-lib:fref x-%data%
                                                   (i)
                                                   ((1 *))
                                                   x-%offset%)
                                    (-
```

```
                              (f2cl-lib:fref x-%data%
                                             (i)
                                             ((1 *))
                                             x-%offset%)
                              (* temp
                                 (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%))))))))))))
                    (t
                     (setf jx kx)
                     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                   ((> j n) nil)
                       (tagbody
                         (cond
                           ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                            (if nounit
                                (setf (f2cl-lib:fref x-%data%
                                                     (jx)
                                                     ((1 *))
                                                     x-%offset%)
                                      (/
                                       (f2cl-lib:fref x-%data%
                                                      (jx)
                                                      ((1 *))
                                                      x-%offset%)
                                       (f2cl-lib:fref a-%data%
                                                      (j j)
                                                      ((1 lda) (1 *))
                                                      a-%offset%))))
                            (setf temp
                                    (f2cl-lib:fref x-%data%
                                                   (jx)
                                                   ((1 *))
                                                   x-%offset%))
                            (setf ix jx)
                            (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                             (f2cl-lib:int-add i 1))
                                          ((> i n) nil)
                              (tagbody
                                (setf ix (f2cl-lib:int-add ix incx))
                                (setf (f2cl-lib:fref x-%data%
                                                     (ix)
                                                     ((1 *))
                                                     x-%offset%)
                                      (-
```

```
                                    (f2cl-lib:fref x-%data%
                                                   (ix)
                                                   ((1 *))
                                                   x-%offset%)
                                    (* temp
                                       (f2cl-lib:fref a-%data%
                                                      (i j)
                                                      ((1 lda) (1 *))
                                                      a-%offset%))))))))
                          (setf jx (f2cl-lib:int-add jx incx)))))))))
                 (t
                  (cond
                    ((char-equal uplo #\U)
                     (cond
                       ((= incx 1)
                        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                      ((> j n) nil)
                          (tagbody
                            (setf temp
                                    (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                          ((> i
                                              (f2cl-lib:int-add j
                                                                (f2cl-lib:int-sub 1)))
                                           nil)
                              (tagbody
                                (setf temp
                                        (- temp
                                           (*
                                            (f2cl-lib:fref a-%data%
                                                           (i j)
                                                           ((1 lda) (1 *))
                                                           a-%offset%)
                                            (f2cl-lib:fref x-%data%
                                                           (i)
                                                           ((1 *))
                                                           x-%offset%))))))
                            (if nounit
                                (setf temp
                                        (/ temp
                                           (f2cl-lib:fref a-%data%
                                                          (j j)
                                                          ((1 lda) (1 *))
                                                          a-%offset%))))
                            (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                                    temp))))
```

```
                (t
                 (setf jx kx)
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (setf temp
                             (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
                     (setf ix kx)
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i
                                       (f2cl-lib:int-add j
                                                         (f2cl-lib:int-sub 1)))
                                    nil)
                       (tagbody
                         (setf temp
                                 (- temp
                                    (*
                                     (f2cl-lib:fref a-%data%
                                                    (i j)
                                                    ((1 lda) (1 *))
                                                    a-%offset%)
                                     (f2cl-lib:fref x-%data%
                                                    (ix)
                                                    ((1 *))
                                                    x-%offset%))))
                         (setf ix (f2cl-lib:int-add ix incx))))
                     (if nounit
                         (setf temp
                                 (/ temp
                                    (f2cl-lib:fref a-%data%
                                                   (j j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%))))
                     (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                             temp)
                     (setf jx (f2cl-lib:int-add jx incx)))))))
               (t
                (cond
                  ((= incx 1)
                   (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                 ((> j 1) nil)
                     (tagbody
                       (setf temp
                               (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                       (f2cl-lib:fdo (i n
                                      (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
```

```
                    ((> i (f2cl-lib:int-add j 1)) nil)
              (tagbody
                (setf temp
                        (- temp
                           (*
                            (f2cl-lib:fref a-%data%
                                           (i j)
                                           ((1 lda) (1 *))
                                           a-%offset%)
                            (f2cl-lib:fref x-%data%
                                           (i)
                                           ((1 *))
                                           x-%offset%))))))
              (if nounit
                  (setf temp
                          (/ temp
                             (f2cl-lib:fref a-%data%
                                            (j j)
                                            ((1 lda) (1 *))
                                            a-%offset%))))
              (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                      temp))))
        (t
         (setf kx
                 (f2cl-lib:int-add kx
                                   (f2cl-lib:int-mul
                                    (f2cl-lib:int-sub n 1)
                                    incx)))
         (setf jx kx)
         (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                       ((> j 1) nil)
           (tagbody
             (setf temp
                     (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
             (setf ix kx)
             (f2cl-lib:fdo (i n
                            (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                           ((> i (f2cl-lib:int-add j 1)) nil)
               (tagbody
                 (setf temp
                         (- temp
                            (*
                             (f2cl-lib:fref a-%data%
                                            (i j)
                                            ((1 lda) (1 *))
                                            a-%offset%)
```

```
                                    (f2cl-lib:fref x-%data%
                                                   (ix)
                                                   ((1 *))
                                                   x-%offset%))))
                            (setf ix (f2cl-lib:int-sub ix incx))))
                        (if nounit
                            (setf temp
                                    (/ temp
                                       (f2cl-lib:fref a-%data%
                                                      (j j)
                                                      ((1 lda) (1 *))
                                                      a-%offset%))))
                        (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                                temp)
                        (setf jx (f2cl-lib:int-sub jx incx))))))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil))))))
```

5.17 zgbmv BLAS

⟨*zgbmv.input*⟩≡

```
)set break resume
)sys rm -f zgbmv.output
)spool zgbmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zgbmv.help*⟩≡

```
=====================================================================
zgbmv examples
=====================================================================

=====================================================================
Man Page Details
=====================================================================

 NAME
      ZGBMV - perform one of the matrix-vector operations   y :=
      alpha*A*x + beta*y, or y := alpha*A'*x + beta*y, or   y :=
      alpha*conjg( A' )*x + beta*y,

 SYNOPSIS
      SUBROUTINE ZGBMV ( TRANS, M, N, KL, KU, ALPHA, A, LDA, X,
                         INCX, BETA, Y, INCY )

          COMPLEX*16   ALPHA, BETA

          INTEGER      INCX, INCY, KL, KU, LDA, M, N

          CHARACTER*1  TRANS

          COMPLEX*16   A( LDA, * ), X( * ), Y( * )

 PURPOSE
      ZGBMV  performs one of the matrix-vector operations

      where alpha and beta are scalars, x and y are vectors and A
      is an m by n band matrix, with kl sub-diagonals and ku
      super-diagonals.

 PARAMETERS
      TRANS  - CHARACTER*1.
             On entry, TRANS specifies the operation to be per-
             formed as follows:

             TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.

             TRANS = 'T' or 't'   y := alpha*A'*x + beta*y.

             TRANS = 'C' or 'c'   y := alpha*conjg( A' )*x +
             beta*y.

             Unchanged on exit.
```

```
M      - INTEGER.
       On entry, M specifies the number of rows of the
       matrix A.  M must be at least zero.  Unchanged on
       exit.

N      - INTEGER.
       On entry, N specifies the number of columns of the
       matrix A.  N must be at least zero.  Unchanged on
       exit.

KL     - INTEGER.
       On entry, KL specifies the number of sub-diagonals of
       the matrix A. KL must satisfy  0 .le. KL.  Unchanged

       on exit.

KU     - INTEGER.
       On entry, KU specifies the number of super-diagonals
       of the matrix A. KU must satisfy  0 .le. KU.
       Unchanged on exit.

ALPHA  - COMPLEX*16      .
       On entry, ALPHA specifies the scalar alpha.
       Unchanged on exit.

A      - COMPLEX*16       array of DIMENSION ( LDA, n ).
       Before entry, the leading ( kl + ku + 1 ) by n part
       of the array A must contain the matrix of coeffi-
       cients, supplied column by column, with the leading
       diagonal of the matrix in row ( ku + 1 ) of the
       array, the first super-diagonal starting at position
       2 in row ku, the first sub-diagonal starting at posi-
       tion 1 in row ( ku + 2 ), and so on.  Elements in the
       array A that do not correspond to elements in the
       band matrix (such as the top left ku by ku triangle)
       are not referenced.  The following program segment
       will transfer a band matrix from conventional full
       matrix storage to band storage:

       DO 20, J = 1, N K = KU + 1 - J DO 10, I = MAX( 1, J -
       KU ), MIN( M, J + KL ) A( K + I, J ) = matrix( I, J )
       10    CONTINUE 20 CONTINUE

       Unchanged on exit.
```

```
LDA    - INTEGER.
       On entry, LDA specifies the first dimension of A as
       declared in the calling (sub) program. LDA must be at
       least ( kl + ku + 1 ).  Unchanged on exit.

X      - COMPLEX*16       array of DIMENSION at least
       ( 1 + ( n - 1 )*abs( INCX ) ) when TRANS = 'N' or 'n'
       and at least ( 1 + ( m - 1 )*abs( INCX ) ) otherwise.
       Before entry, the incremented array X must contain
       the vector x.  Unchanged on exit.

INCX   - INTEGER.
       On entry, INCX specifies the increment for the ele-
       ments of X. INCX must not be zero.  Unchanged on
       exit.

BETA   - COMPLEX*16      .
       On entry, BETA specifies the scalar beta. When BETA
       is supplied as zero then Y need not be set on input.
       Unchanged on exit.

Y      - COMPLEX*16       array of DIMENSION at least
       ( 1 + ( m - 1 )*abs( INCY ) ) when TRANS = 'N' or 'n'
       and at least ( 1 + ( n - 1 )*abs( INCY ) ) otherwise.
       Before entry, the incremented array Y must contain
       the vector y. On exit, Y is overwritten by the
       updated vector y.

INCY   - INTEGER.
       On entry, INCY specifies the increment for the ele-
       ments of Y. INCY must not be zero.  Unchanged on
       exit.
```

⟨*BLAS 2 zgbmv*⟩≡

```
(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
 (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zgbmv (trans m n kl ku alpha a lda x incx beta y incy)
    (declare (type (simple-array (complex double-float) (*)) y x a)
             (type (complex double-float) beta alpha)
             (type fixnum incy incx lda ku kl n m)
             (type character trans))
    (f2cl-lib:with-multi-array-data
        ((trans character trans-%data% trans-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (x (complex double-float) x-%data% x-%offset%)
         (y (complex double-float) y-%data% y-%offset%))
      (prog ((noconj nil) (i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0)
             (k 0) (kup1 0) (kx 0) (ky 0) (lenx 0) (leny 0) (temp #C(0.0 0.0)))
        (declare (type (member t nil) noconj)
                 (type fixnum i info ix iy j jx jy k kup1 kx ky
                                      lenx leny)
                 (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 1))
          ((< m 0)
           (setf info 2))
          ((< n 0)
           (setf info 3))
          ((< kl 0)
           (setf info 4))
          ((< ku 0)
           (setf info 5))
          ((< lda (f2cl-lib:int-add kl ku 1))
           (setf info 8))
          ((= incx 0)
           (setf info 10))
          ((= incy 0)
           (setf info 13)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZGBMV" info)
           (go end_label)))
        (if (or (= m 0) (= n 0) (and (= alpha zero) (= beta one)))
```

```
    (go end_label))
(setf noconj (char-equal trans #\T))
(cond
  ((char-equal trans #\N)
   (setf lenx n)
   (setf leny m))
  (t
   (setf lenx m)
   (setf leny n)))
(cond
  ((> incx 0)
   (setf kx 1))
  (t
   (setf kx
           (f2cl-lib:int-sub 1
                             (f2cl-lib:int-mul
                              (f2cl-lib:int-sub lenx 1)
                              incx)))))
(cond
  ((> incy 0)
   (setf ky 1))
  (t
   (setf ky
           (f2cl-lib:int-sub 1
                             (f2cl-lib:int-mul
                              (f2cl-lib:int-sub leny 1)
                              incy)))))
(cond
  ((/= beta one)
   (cond
     ((= incy 1)
      (cond
        ((= beta zero)
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i leny) nil)
           (tagbody
             (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                     zero))))
        (t
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i leny) nil)
           (tagbody
             (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                     (* beta
                        (f2cl-lib:fref y-%data%
                                       (i)
```

```
                                        ((1 *))
                                        y-%offset%))))))))
          (t
           (setf iy ky)
           (cond
             ((= beta zero)
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                            ((> i leny) nil)
                (tagbody
                  (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                          zero)
                  (setf iy (f2cl-lib:int-add iy incy)))))
             (t
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                            ((> i leny) nil)
                (tagbody
                  (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                          (* beta
                             (f2cl-lib:fref y-%data%
                                            (iy)
                                            ((1 *))
                                            y-%offset%)))
                  (setf iy (f2cl-lib:int-add iy incy)))))))))
    (if (= alpha zero) (go end_label))
    (setf kup1 (f2cl-lib:int-add ku 1))
    (cond
      ((char-equal trans #\N)
       (setf jx kx)
       (cond
         ((= incy 1)
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                        ((> j n) nil)
            (tagbody
              (cond
                ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                 (setf temp
                         (* alpha
                            (f2cl-lib:fref x-%data%
                                           (jx)
                                           ((1 *))
                                           x-%offset%)))
                 (setf k (f2cl-lib:int-sub kup1 j))
                 (f2cl-lib:fdo (i
                                (max (the fixnum 1)
                                     (the fixnum
                                          (f2cl-lib:int-add j
```

```
                                                    (f2cl-lib:int-sub
                                                     ku))))
                                 (f2cl-lib:int-add i 1))
                                ((> i
                                    (min (the fixnum m)
                                         (the fixnum
                                              (f2cl-lib:int-add j kl))))
                                 nil)
                           (tagbody
                             (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                                     (+
                                      (f2cl-lib:fref y-%data%
                                                     (i)
                                                     ((1 *))
                                                     y-%offset%)
                                      (* temp
                                         (f2cl-lib:fref a-%data%
                                                        ((f2cl-lib:int-add k i) j)
                                                        ((1 lda) (1 *))
                                                        a-%offset%))))))))
                 (setf jx (f2cl-lib:int-add jx incx)))))
            (t
             (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                           ((> j n) nil)
               (tagbody
                 (cond
                   ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                    (setf temp
                            (* alpha
                               (f2cl-lib:fref x-%data%
                                              (jx)
                                              ((1 *))
                                              x-%offset%)))
                    (setf iy ky)
                    (setf k (f2cl-lib:int-sub kup1 j))
                    (f2cl-lib:fdo (i
                                   (max (the fixnum 1)
                                        (the fixnum
                                             (f2cl-lib:int-add j
                                                               (f2cl-lib:int-sub
                                                                ku))))
                                   (f2cl-lib:int-add i 1))
                                  ((> i
                                      (min (the fixnum m)
                                           (the fixnum
                                                (f2cl-lib:int-add j kl))))
```

```
                                nil)
                        (tagbody
                          (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                                (+
                                 (f2cl-lib:fref y-%data%
                                                (iy)
                                                ((1 *))
                                                y-%offset%)
                                 (* temp
                                    (f2cl-lib:fref a-%data%
                                                   ((f2cl-lib:int-add k i) j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%))))
                          (setf iy (f2cl-lib:int-add iy incy))))))
                (setf jx (f2cl-lib:int-add jx incx))
                (if (> j ku) (setf ky (f2cl-lib:int-add ky incy)))))))))
          (t
           (setf jy ky)
           (cond
             ((= incx 1)
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (setf temp zero)
                  (setf k (f2cl-lib:int-sub kup1 j))
                  (cond
                    (noconj
                     (f2cl-lib:fdo (i
                                    (max (the fixnum 1)
                                         (the fixnum
                                              (f2cl-lib:int-add j
                                                                (f2cl-lib:int-sub
                                                                 ku))))
                                    (f2cl-lib:int-add i 1))
                                   ((> i
                                       (min (the fixnum m)
                                            (the fixnum
                                                 (f2cl-lib:int-add j kl))))
                                    nil)
                       (tagbody
                         (setf temp
                                 (+ temp
                                    (*
                                     (f2cl-lib:fref a-%data%
                                                    ((f2cl-lib:int-add k i) j)
                                                    ((1 lda) (1 *))
```

```
                                        a-%offset%)
                                (f2cl-lib:fref x-%data%
                                        (i)
                                        ((1 *))
                                        x-%offset%)))))))
                        (t
                         (f2cl-lib:fdo (i
                                        (max (the fixnum 1)
                                             (the fixnum
                                                  (f2cl-lib:int-add j
                                                                    (f2cl-lib:int-sub
                                                                     ku))))
                                        (f2cl-lib:int-add i 1))
                                       ((> i
                                           (min (the fixnum m)
                                                (the fixnum
                                                     (f2cl-lib:int-add j kl))))
                                        nil)
                           (tagbody
                             (setf temp
                                     (+ temp
                                        (*
                                         (f2cl-lib:dconjg
                                          (f2cl-lib:fref a-%data%
                                                         ((f2cl-lib:int-add k i) j)
                                                         ((1 lda) (1 *))
                                                         a-%offset%))
                                         (f2cl-lib:fref x-%data%
                                                        (i)
                                                        ((1 *))
                                                        x-%offset%))))))))
                     (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                             (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                (* alpha temp)))
                     (setf jy (f2cl-lib:int-add jy incy)))))
                (t
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (setf temp zero)
                     (setf ix kx)
                     (setf k (f2cl-lib:int-sub kup1 j))
                     (cond
                       (noconj
                        (f2cl-lib:fdo (i
                                       (max (the fixnum 1)
```

```
                                   (the fixnum
                                        (f2cl-lib:int-add j
                                                          (f2cl-lib:int-sub
                                                           ku))))
                               (f2cl-lib:int-add i 1))
                              ((> i
                                  (min (the fixnum m)
                                       (the fixnum
                                            (f2cl-lib:int-add j kl))))
                               nil)
                          (tagbody
                            (setf temp
                                  (+ temp
                                     (*
                                      (f2cl-lib:fref a-%data%
                                                     ((f2cl-lib:int-add k i) j)
                                                     ((1 lda) (1 *))
                                                     a-%offset%)
                                      (f2cl-lib:fref x-%data%
                                                     (ix)
                                                     ((1 *))
                                                     x-%offset%))))
                            (setf ix (f2cl-lib:int-add ix incx)))))
                        (t
                         (f2cl-lib:fdo (i
                                        (max (the fixnum 1)
                                             (the fixnum
                                                  (f2cl-lib:int-add j
                                                                    (f2cl-lib:int-sub
                                                                     ku))))
                                        (f2cl-lib:int-add i 1))
                                       ((> i
                                           (min (the fixnum m)
                                                (the fixnum
                                                     (f2cl-lib:int-add j kl))))
                                        nil)
                           (tagbody
                             (setf temp
                                   (+ temp
                                      (*
                                       (f2cl-lib:dconjg
                                        (f2cl-lib:fref a-%data%
                                                       ((f2cl-lib:int-add k i) j)
                                                       ((1 lda) (1 *))
                                                       a-%offset%))
                                       (f2cl-lib:fref x-%data%
```

```
                                                    (ix)
                                                    ((1 *))
                                                    x-%offset%))))
                                (setf ix (f2cl-lib:int-add ix incx))))))
                            (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                    (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                       (* alpha temp)))
                            (setf jy (f2cl-lib:int-add jy incy))
                            (if (> j ku) (setf kx (f2cl-lib:int-add kx incx)))))))))
  end_label
         (return
          (values nil nil nil nil nil nil nil nil nil nil nil nil nil))))))
```

5.18 zgemv BLAS

⟨*zgemv.input*⟩≡

```
)set break resume
)sys rm -f zgemv.output
)spool zgemv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zgemv.help*⟩≡

```
==========================================================================
zgemv examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

 NAME
      ZGEMV - perform one of the matrix-vector operations   y :=
      alpha*A*x + beta*y, or y := alpha*A'*x + beta*y, or   y :=
      alpha*conjg( A' )*x + beta*y,

 SYNOPSIS
      SUBROUTINE ZGEMV ( TRANS, M, N, ALPHA, A, LDA, X, INCX,
                         BETA, Y, INCY )

          COMPLEX*16   ALPHA, BETA

          INTEGER      INCX, INCY, LDA, M, N

          CHARACTER*1  TRANS

          COMPLEX*16   A( LDA, * ), X( * ), Y( * )

 PURPOSE
      ZGEMV  performs one of the matrix-vector operations

      where alpha and beta are scalars, x and y are vectors and A
      is an m by n matrix.

 PARAMETERS
      TRANS  - CHARACTER*1.
             On entry, TRANS specifies the operation to be per-
             formed as follows:

             TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.

             TRANS = 'T' or 't'   y := alpha*A'*x + beta*y.

             TRANS = 'C' or 'c'   y := alpha*conjg( A' )*x +
             beta*y.

             Unchanged on exit.
```

```
M      - INTEGER.
         On entry, M specifies the number of rows of the
         matrix A.  M must be at least zero.  Unchanged on
         exit.

N      - INTEGER.
         On entry, N specifies the number of columns of the
         matrix A.  N must be at least zero.  Unchanged on
         exit.

ALPHA  - COMPLEX*16      .
         On entry, ALPHA specifies the scalar alpha.
         Unchanged on exit.

A      - COMPLEX*16       array of DIMENSION ( LDA, n ).
         Before entry, the leading m by n part of the array A
         must contain the matrix of coefficients.  Unchanged
         on exit.

LDA    - INTEGER.
         On entry, LDA specifies the first dimension of A as
         declared in the calling (sub) program. LDA must be at
         least max( 1, m ).  Unchanged on exit.

X      - COMPLEX*16       array of DIMENSION at least
         ( 1 + ( n - 1 )*abs( INCX ) ) when TRANS = 'N' or 'n'
         and at least ( 1 + ( m - 1 )*abs( INCX ) ) otherwise.
         Before entry, the incremented array X must contain
         the vector x.  Unchanged on exit.

INCX   - INTEGER.
         On entry, INCX specifies the increment for the ele-
         ments of X. INCX must not be zero.  Unchanged on
         exit.

BETA   - COMPLEX*16      .
         On entry, BETA specifies the scalar beta. When BETA
         is supplied as zero then Y need not be set on input.
         Unchanged on exit.

Y      - COMPLEX*16       array of DIMENSION at least
         ( 1 + ( m - 1 )*abs( INCY ) ) when TRANS = 'N' or 'n'
         and at least ( 1 + ( n - 1 )*abs( INCY ) ) otherwise.
         Before entry with BETA non-zero, the incremented
         array Y must contain the vector y. On exit, Y is
         overwritten by the updated vector y.
```

```
INCY   - INTEGER.
         On entry, INCY specifies the increment for the ele-
         ments of Y. INCY must not be zero.  Unchanged on
         exit.
```

⟨*BLAS 2 zgemv*⟩≡

```
(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
 (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zgemv (trans m n alpha a lda x incx beta y incy)
    (declare (type (simple-array (complex double-float) (*)) y x a)
             (type (complex double-float) beta alpha)
             (type fixnum incy incx lda n m)
             (type character trans))
    (f2cl-lib:with-multi-array-data
        ((trans character trans-%data% trans-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (x (complex double-float) x-%data% x-%offset%)
         (y (complex double-float) y-%data% y-%offset%))
      (prog ((noconj nil) (i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0)
             (kx 0) (ky 0) (lenx 0) (leny 0) (temp #C(0.0 0.0)))
        (declare (type (member t nil) noconj)
                 (type fixnum i info ix iy j jx jy kx ky lenx
                                          leny)
                 (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 1))
          ((< m 0)
           (setf info 2))
          ((< n 0)
           (setf info 3))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info 6))
          ((= incx 0)
           (setf info 8))
          ((= incy 0)
           (setf info 11)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZGEMV" info)
           (go end_label)))
        (if (or (= m 0) (= n 0) (and (= alpha zero) (= beta one)))
            (go end_label))
        (setf noconj (char-equal trans #\T))
        (cond
          ((char-equal trans #\N)
```

```
          (setf lenx n)
          (setf leny m))
         (t
          (setf lenx m)
          (setf leny n)))
       (cond
         ((> incx 0)
          (setf kx 1))
         (t
          (setf kx
                  (f2cl-lib:int-sub 1
                                    (f2cl-lib:int-mul
                                     (f2cl-lib:int-sub lenx 1)
                                     incx)))))
       (cond
         ((> incy 0)
          (setf ky 1))
         (t
          (setf ky
                  (f2cl-lib:int-sub 1
                                    (f2cl-lib:int-mul
                                     (f2cl-lib:int-sub leny 1)
                                     incy)))))
       (cond
         ((/= beta one)
          (cond
            ((= incy 1)
             (cond
               ((= beta zero)
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                              ((> i leny) nil)
                  (tagbody
                    (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                            zero))))
               (t
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                              ((> i leny) nil)
                  (tagbody
                    (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                            (* beta
                               (f2cl-lib:fref y-%data%
                                              (i)
                                              ((1 *))
                                              y-%offset%)))))))))
            (t
             (setf iy ky)
```

```
        (cond
          ((= beta zero)
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                         ((> i leny) nil)
             (tagbody
               (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                       zero)
               (setf iy (f2cl-lib:int-add iy incy)))))
          (t
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                         ((> i leny) nil)
             (tagbody
               (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                       (* beta
                          (f2cl-lib:fref y-%data%
                                         (iy)
                                         ((1 *))
                                         y-%offset%)))
               (setf iy (f2cl-lib:int-add iy incy)))))))
(if (= alpha zero) (go end_label))
(cond
  ((char-equal trans #\N)
   (setf jx kx)
   (cond
     ((= incy 1)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (cond
            ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
             (setf temp
                     (* alpha
                        (f2cl-lib:fref x-%data%
                                       (jx)
                                       ((1 *))
                                       x-%offset%)))
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i m) nil)
               (tagbody
                 (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                         (+
                          (f2cl-lib:fref y-%data%
                                         (i)
                                         ((1 *))
                                         y-%offset%)
                          (* temp
```

```
                                (f2cl-lib:fref a-%data%
                                               (i j)
                                               ((1 lda) (1 *))
                                               a-%offset%)))))))
            (setf jx (f2cl-lib:int-add jx incx)))))
       (t
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                      ((> j n) nil)
          (tagbody
            (cond
              ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
               (setf temp
                       (* alpha
                          (f2cl-lib:fref x-%data%
                                         (jx)
                                         ((1 *))
                                         x-%offset%)))
               (setf iy ky)
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i m) nil)
                 (tagbody
                   (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                           (+
                            (f2cl-lib:fref y-%data%
                                           (iy)
                                           ((1 *))
                                           y-%offset%)
                            (* temp
                               (f2cl-lib:fref a-%data%
                                              (i j)
                                              ((1 lda) (1 *))
                                              a-%offset%))))
                   (setf iy (f2cl-lib:int-add iy incy))))))
            (setf jx (f2cl-lib:int-add jx incx)))))))
  (t
   (setf jy ky)
   (cond
     ((= incx 1)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (setf temp zero)
          (cond
            (noconj
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i m) nil)
```

```
                (tagbody
                  (setf temp
                          (+ temp
                             (*
                              (f2cl-lib:fref a-%data%
                                             (i j)
                                             ((1 lda) (1 *))
                                             a-%offset%)
                              (f2cl-lib:fref x-%data%
                                             (i)
                                             ((1 *))
                                             x-%offset%)))))))
               (t
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                              ((> i m) nil)
                  (tagbody
                    (setf temp
                            (+ temp
                               (*
                                (f2cl-lib:dconjg
                                 (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%))
                                (f2cl-lib:fref x-%data%
                                               (i)
                                               ((1 *))
                                               x-%offset%)))))))))
          (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                  (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                     (* alpha temp)))
          (setf jy (f2cl-lib:int-add jy incy)))))
     (t
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (setf temp zero)
          (setf ix kx)
          (cond
            (noconj
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i m) nil)
               (tagbody
                 (setf temp
                         (+ temp
                            (*
```

```
                                    (f2cl-lib:fref a-%data%
                                                   (i j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%)
                                    (f2cl-lib:fref x-%data%
                                                   (ix)
                                                   ((1 *))
                                                   x-%offset%))))
                            (setf ix (f2cl-lib:int-add ix incx)))))
                       (t
                        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                      ((> i m) nil)
                          (tagbody
                            (setf temp
                                    (+ temp
                                       (*
                                        (f2cl-lib:dconjg
                                         (f2cl-lib:fref a-%data%
                                                        (i j)
                                                        ((1 lda) (1 *))
                                                        a-%offset%))
                                        (f2cl-lib:fref x-%data%
                                                       (ix)
                                                       ((1 *))
                                                       x-%offset%))))
                            (setf ix (f2cl-lib:int-add ix incx))))))
                      (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                              (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                 (* alpha temp)))
                      (setf jy (f2cl-lib:int-add jy incy))))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil nil nil nil))))))
```

5.19 zgerc BLAS

⟨*zgerc.input*⟩≡

```
)set break resume
)sys rm -f zgerc.output
)spool zgerc.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zgerc.help*⟩≡

```
  ======================================================================
  zgerc examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        ZGERC - perform the rank 1 operation   A := alpha*x*conjg(
        y' ) + A,

   SYNOPSIS
        SUBROUTINE ZGERC ( M, N, ALPHA, X, INCX, Y, INCY, A, LDA )

            COMPLEX*16   ALPHA

            INTEGER      INCX, INCY, LDA, M, N

            COMPLEX*16   A( LDA, * ), X( * ), Y( * )

   PURPOSE
        ZGERC  performs the rank 1 operation

        where alpha is a scalar, x is an m element vector, y is an n
        element vector and A is an m by n matrix.

   PARAMETERS
        M      - INTEGER.
                 On entry, M specifies the number of rows of the
                 matrix A.  M must be at least zero.  Unchanged on
                 exit.

        N      - INTEGER.
                 On entry, N specifies the number of columns of the
                 matrix A.  N must be at least zero.  Unchanged on
                 exit.

        ALPHA  - COMPLEX*16      .
                 On entry, ALPHA specifies the scalar alpha.
                 Unchanged on exit.

        X      - COMPLEX*16       array of dimension at least
                 ( 1 + ( m - 1 )*abs( INCX ) ).  Before entry, the
                 incremented array X must contain the m element vector
```

```
          x.  Unchanged on exit.

INCX   - INTEGER.
          On entry, INCX specifies the increment for the ele-
          ments of X. INCX must not be zero.  Unchanged on
          exit.

Y      - COMPLEX*16       array of dimension at least
          ( 1 + ( n - 1 )*abs( INCY ) ).  Before entry, the
          incremented array Y must contain the n element vector
          y.  Unchanged on exit.

INCY   - INTEGER.
          On entry, INCY specifies the increment for the

          elements of Y. INCY must not be zero.  Unchanged on
          exit.

A      - COMPLEX*16       array of DIMENSION ( LDA, n ).
          Before entry, the leading m by n part of the array A
          must contain the matrix of coefficients. On exit, A
          is overwritten by the updated matrix.

LDA    - INTEGER.
          On entry, LDA specifies the first dimension of A as
          declared in the calling (sub) program. LDA must be at
          least max( 1, m ).  Unchanged on exit.
```

⟨*BLAS 2 zgerc*⟩≡

```
(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun zgerc (m n alpha x incx y incy a lda)
    (declare (type (simple-array (complex double-float) (*)) a y x)
             (type (complex double-float) alpha)
             (type fixnum lda incy incx n m))
    (f2cl-lib:with-multi-array-data
        ((x (complex double-float) x-%data% x-%offset%)
         (y (complex double-float) y-%data% y-%offset%)
         (a (complex double-float) a-%data% a-%offset%))
      (prog ((i 0) (info 0) (ix 0) (j 0) (jy 0) (kx 0) (temp #C(0.0 0.0)))
        (declare (type fixnum i info ix j jy kx)
                 (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((< m 0)
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((= incx 0)
           (setf info 5))
          ((= incy 0)
           (setf info 7))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info 9)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZGERC" info)
           (go end_label)))
        (if (or (= m 0) (= n 0) (= alpha zero)) (go end_label))
        (cond
          ((> incy 0)
           (setf jy 1))
          (t
           (setf jy
                 (f2cl-lib:int-sub 1
                                   (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                     incy)))))
        (cond
          ((= incx 1)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
```

```
      (cond
        ((/= (f2cl-lib:fref y (jy) ((1 *))) zero)
         (setf temp
                 (* alpha
                    (f2cl-lib:dconjg
                     (f2cl-lib:fref y-%data%
                                    (jy)
                                    ((1 *))
                                    y-%offset%))))
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i m) nil)
           (tagbody
             (setf (f2cl-lib:fref a-%data%
                                  (i j)
                                  ((1 lda) (1 *))
                                  a-%offset%)
                   (+
                    (f2cl-lib:fref a-%data%
                                   (i j)
                                   ((1 lda) (1 *))
                                   a-%offset%)
                    (*
                     (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%)
                     temp)))))))
      (setf jy (f2cl-lib:int-add jy incy)))))
(t
 (cond
   ((> incx 0)
    (setf kx 1))
   (t
    (setf kx
            (f2cl-lib:int-sub 1
                              (f2cl-lib:int-mul
                               (f2cl-lib:int-sub m 1)
                               incx)))))
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j n) nil)
   (tagbody
     (cond
       ((/= (f2cl-lib:fref y (jy) ((1 *))) zero)
        (setf temp
                (* alpha
                   (f2cl-lib:dconjg
                    (f2cl-lib:fref y-%data%
                                   (jy)
                                   ((1 *))
```

```
                                        y-%offset%))))
                    (setf ix kx)
                    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                  ((> i m) nil)
                      (tagbody
                        (setf (f2cl-lib:fref a-%data%
                                             (i j)
                                             ((1 lda) (1 *))
                                             a-%offset%)
                                      (+
                                       (f2cl-lib:fref a-%data%
                                                      (i j)
                                                      ((1 lda) (1 *))
                                                      a-%offset%)
                                       (*
                                        (f2cl-lib:fref x-%data%
                                                       (ix)
                                                       ((1 *))
                                                       x-%offset%)
                                        temp)))
                        (setf ix (f2cl-lib:int-add ix incx))))))
                  (setf jy (f2cl-lib:int-add jy incy))))))
   end_label
          (return (values nil nil nil nil nil nil nil nil nil))))))
```

5.20 zgeru BLAS

⟨*zgeru.input*⟩≡

```
  )set break resume
  )sys rm -f zgeru.output
  )spool zgeru.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*zgeru.help*⟩≡

```
==========================================================================
zgeru examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

 NAME
      ZGERU - perform the rank 1 operation   A := alpha*x*y' + A,

 SYNOPSIS
      SUBROUTINE ZGERU ( M, N, ALPHA, X, INCX, Y, INCY, A, LDA )

          COMPLEX*16   ALPHA

          INTEGER      INCX, INCY, LDA, M, N

          COMPLEX*16   A( LDA, * ), X( * ), Y( * )

 PURPOSE
      ZGERU  performs the rank 1 operation

      where alpha is a scalar, x is an m element vector, y is an n
      element vector and A is an m by n matrix.

 PARAMETERS
      M      - INTEGER.
               On entry, M specifies the number of rows of the
               matrix A.  M must be at least zero.  Unchanged on
               exit.

      N      - INTEGER.
               On entry, N specifies the number of columns of the
               matrix A.  N must be at least zero.  Unchanged on
               exit.

      ALPHA  - COMPLEX*16      .
               On entry, ALPHA specifies the scalar alpha.
               Unchanged on exit.

      X      - COMPLEX*16       array of dimension at least
               ( 1 + ( m - 1 )*abs( INCX ) ).  Before entry, the
               incremented array X must contain the m element vector
               x.  Unchanged on exit.
```

```
INCX   - INTEGER.
       On entry, INCX specifies the increment for the ele-
       ments of X. INCX must not be zero.  Unchanged on
       exit.

Y      - COMPLEX*16       array of dimension at least
       ( 1 + ( n - 1 )*abs( INCY ) ).  Before entry, the
       incremented array Y must contain the n element vector
       y.  Unchanged on exit.

INCY   - INTEGER.
       On entry, INCY specifies the increment for the ele-
       ments of Y. INCY must not be zero.  Unchanged on

       exit.

A      - COMPLEX*16       array of DIMENSION ( LDA, n ).
       Before entry, the leading m by n part of the array A
       must contain the matrix of coefficients. On exit, A
       is overwritten by the updated matrix.

LDA    - INTEGER.
       On entry, LDA specifies the first dimension of A as
       declared in the calling (sub) program. LDA must be at
       least max( 1, m ).  Unchanged on exit.
```

⟨*BLAS 2 zgeru*⟩≡

```
(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun zgeru (m n alpha x incx y incy a lda)
    (declare (type (simple-array (complex double-float) (*)) a y x)
             (type (complex double-float) alpha)
             (type fixnum lda incy incx n m))
    (f2cl-lib:with-multi-array-data
        ((x (complex double-float) x-%data% x-%offset%)
         (y (complex double-float) y-%data% y-%offset%)
         (a (complex double-float) a-%data% a-%offset%))
      (prog ((i 0) (info 0) (ix 0) (j 0) (jy 0) (kx 0) (temp #C(0.0 0.0)))
        (declare (type fixnum i info ix j jy kx)
                 (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((< m 0)
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((= incx 0)
           (setf info 5))
          ((= incy 0)
           (setf info 7))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info 9)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZGERU" info)
           (go end_label)))
        (if (or (= m 0) (= n 0) (= alpha zero)) (go end_label))
        (cond
          ((> incy 0)
           (setf jy 1))
          (t
           (setf jy
                   (f2cl-lib:int-sub 1
                                     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                       incy)))))
        (cond
          ((= incx 1)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
```

```
          (cond
            ((/= (f2cl-lib:fref y (jy) ((1 *))) zero)
             (setf temp
                     (* alpha
                        (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)))
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i m) nil)
               (tagbody
                 (setf (f2cl-lib:fref a-%data%
                                      (i j)
                                      ((1 lda) (1 *))
                                      a-%offset%)
                       (+
                        (f2cl-lib:fref a-%data%
                                       (i j)
                                       ((1 lda) (1 *))
                                       a-%offset%)
                        (*
                         (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%)
                         temp)))))))
          (setf jy (f2cl-lib:int-add jy incy)))))
      (t
       (cond
         ((> incx 0)
          (setf kx 1))
         (t
          (setf kx
                  (f2cl-lib:int-sub 1
                                    (f2cl-lib:int-mul
                                     (f2cl-lib:int-sub m 1)
                                     incx)))))
       (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                     ((> j n) nil)
         (tagbody
           (cond
             ((/= (f2cl-lib:fref y (jy) ((1 *))) zero)
              (setf temp
                      (* alpha
                         (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)))
              (setf ix kx)
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                            ((> i m) nil)
                (tagbody
                  (setf (f2cl-lib:fref a-%data%
                                       (i j)
                                       ((1 lda) (1 *))
```

```
                                        a-%offset%)
                                (+
                                 (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%)
                                 (*
                                  (f2cl-lib:fref x-%data%
                                                 (ix)
                                                 ((1 *))
                                                 x-%offset%)
                                  temp)))
                          (setf ix (f2cl-lib:int-add ix incx))))))
                  (setf jy (f2cl-lib:int-add jy incy))))))
   end_label
          (return (values nil nil nil nil nil nil nil nil nil))))))
```

5.21 zhbmv BLAS

⟨*zhbmv.input*⟩≡

```
  )set break resume
  )sys rm -f zhbmv.output
  )spool zhbmv.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*zhbmv.help*⟩≡

```
  ======================================================================
  zhbmv examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        ZHBMV - perform the matrix-vector operation   y := alpha*A*x
        + beta*y,

   SYNOPSIS
        SUBROUTINE ZHBMV ( UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA,
                           Y, INCY )

            COMPLEX*16   ALPHA, BETA

            INTEGER      INCX, INCY, K, LDA, N

            CHARACTER*1  UPLO

            COMPLEX*16   A( LDA, * ), X( * ), Y( * )

   PURPOSE
        ZHBMV  performs the matrix-vector  operation

        where alpha and beta are scalars, x and y are n element vec-
        tors and A is an n by n hermitian band matrix, with k
        super-diagonals.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On entry, UPLO specifies whether the upper or lower
                 triangular part of the band matrix A is being sup-
                 plied as follows:

                 UPLO = 'U' or 'u'   The upper triangular part of A is
                 being supplied.

                 UPLO = 'L' or 'l'   The lower triangular part of A is
                 being supplied.

                 Unchanged on exit.
```

N - INTEGER.
On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

K - INTEGER.
On entry, K specifies the number of super-diagonals of the matrix A. K must satisfy 0 .le. K. Unchanged on exit.

ALPHA - COMPLEX*16 .
On entry, ALPHA specifies the scalar alpha.
Unchanged on exit.

A - COMPLEX*16 array of DIMENSION (LDA, n).

Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the hermitian matrix, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer the upper triangular part of a hermitian band matrix from conventional full matrix storage to band storage:

DO 20, J = 1, N M = K + 1 - J DO 10, I = MAX(1, J - K), J A(M + I, J) = matrix(I, J) 10 CONTINUE 20 CONTINUE

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the hermitian matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer the lower triangular part of a hermitian band matrix from conventional full matrix storage to band storage:

DO 20, J = 1, N M = 1 - J DO 10, I = J, MIN(N, J + K) A(M + I, J) = matrix(I, J) 10 CONTINUE 20 CONTINUE

```
          Note that the imaginary parts of the diagonal ele-
          ments need not be set and are assumed to be zero.
          Unchanged on exit.

LDA       - INTEGER.
          On entry, LDA specifies the first dimension of A as
          declared in the calling (sub) program. LDA must be at
          least ( k + 1 ).  Unchanged on exit.

X         - COMPLEX*16       array of DIMENSION at least
          ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
          incremented array X must contain the vector x.
          Unchanged on exit.

INCX      - INTEGER.
          On entry, INCX specifies the increment for the ele-
          ments of X. INCX must not be zero.  Unchanged on
          exit.

BETA      - COMPLEX*16      .

          On entry, BETA specifies the scalar beta.  Unchanged
          on exit.

Y         - COMPLEX*16       array of DIMENSION at least
          ( 1 + ( n - 1 )*abs( INCY ) ).  Before entry, the
          incremented array Y must contain the vector y. On
          exit, Y is overwritten by the updated vector y.

INCY      - INTEGER.
          On entry, INCY specifies the increment for the ele-
          ments of Y. INCY must not be zero.  Unchanged on
          exit.
```

⟨*BLAS 2 zhbmv*⟩≡

```
(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
 (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zhbmv (uplo n k alpha a lda x incx beta y incy)
    (declare (type (simple-array (complex double-float) (*)) y x a)
             (type (complex double-float) beta alpha)
             (type fixnum incy incx lda k n)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (x (complex double-float) x-%data% x-%offset%)
         (y (complex double-float) y-%data% y-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (kplus1 0) (kx 0)
             (ky 0) (l 0) (temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)))
        (declare (type fixnum i info ix iy j jx jy kplus1 kx ky l)
                 (type (complex double-float) temp1 temp2))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((< k 0)
           (setf info 3))
          ((< lda (f2cl-lib:int-add k 1))
           (setf info 6))
          ((= incx 0)
           (setf info 8))
          ((= incy 0)
           (setf info 11)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZHBMV" info)
           (go end_label)))
        (if (or (= n 0) (and (= alpha zero) (= beta one))) (go end_label))
        (cond
          ((> incx 0)
           (setf kx 1))
          (t
           (setf kx
                   (f2cl-lib:int-sub 1
                                     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                       incx)))))
```

```
          (cond
            ((> incy 0)
             (setf ky 1))
            (t
             (setf ky
                   (f2cl-lib:int-sub 1
                                     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                       incy)))))
          (cond
            ((/= beta one)
             (cond
               ((= incy 1)
                (cond
                  ((= beta zero)
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                               zero))))
                  (t
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                               (* beta
                                  (f2cl-lib:fref y-%data%
                                                 (i)
                                                 ((1 *))
                                                 y-%offset%)))))))))
               (t
                (setf iy ky)
                (cond
                  ((= beta zero)
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                               zero)
                       (setf iy (f2cl-lib:int-add iy incy)))))
                  (t
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                               (* beta
                                  (f2cl-lib:fref y-%data%
```

```
                                        (iy)
                                        ((1 *))
                                        y-%offset%)))
              (setf iy (f2cl-lib:int-add iy incy))))))))))
(if (= alpha zero) (go end_label))
(cond
  ((char-equal uplo #\U)
   (setf kplus1 (f2cl-lib:int-add k 1))
   (cond
     ((and (= incx 1) (= incy 1))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (setf temp1
                  (* alpha
                     (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
          (setf temp2 zero)
          (setf l (f2cl-lib:int-sub kplus1 j))
          (f2cl-lib:fdo (i
                         (max (the fixnum 1)
                              (the fixnum
                                   (f2cl-lib:int-add j
                                                     (f2cl-lib:int-sub
                                                      k))))
                         (f2cl-lib:int-add i 1))
                        ((> i
                            (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                         nil)
            (tagbody
              (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                      (+
                       (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                       (* temp1
                          (f2cl-lib:fref a-%data%
                                         ((f2cl-lib:int-add l i) j)
                                         ((1 lda) (1 *))
                                         a-%offset%))))
              (setf temp2
                      (+ temp2
                         (*
                          (f2cl-lib:dconjg
                           (f2cl-lib:fref a-%data%
                                          ((f2cl-lib:int-add l i) j)
                                          ((1 lda) (1 *))
                                          a-%offset%))
                          (f2cl-lib:fref x-%data%
```

```
                                        (i)
                                        ((1 *))
                                        x-%offset%))))))
            (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                    (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                       (* temp1
                          (coerce (realpart
                           (f2cl-lib:fref a-%data%
                                          (kplus1 j)
                                          ((1 lda) (1 *))
                                          a-%offset%)) 'double-float))
                       (* alpha temp2))))))
          (t
           (setf jx kx)
           (setf jy ky)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf temp1
                       (* alpha
                          (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
               (setf temp2 zero)
               (setf ix kx)
               (setf iy ky)
               (setf l (f2cl-lib:int-sub kplus1 j))
               (f2cl-lib:fdo (i
                              (max (the fixnum 1)
                                   (the fixnum
                                        (f2cl-lib:int-add j
                                                          (f2cl-lib:int-sub
                                                           k))))
                              (f2cl-lib:int-add i 1))
                             ((> i
                                 (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                              nil)
                 (tagbody
                   (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                           (+
                            (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                            (* temp1
                               (f2cl-lib:fref a-%data%
                                              ((f2cl-lib:int-add l i) j)
                                              ((1 lda) (1 *))
                                              a-%offset%))))
                   (setf temp2
                           (+ temp2
```

```
                                (*
                                 (f2cl-lib:dconjg
                                  (f2cl-lib:fref a-%data%
                                                 ((f2cl-lib:int-add l i) j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))
                                 (f2cl-lib:fref x-%data%
                                                (ix)
                                                ((1 *))
                                                x-%offset%))))
                            (setf ix (f2cl-lib:int-add ix incx))
                            (setf iy (f2cl-lib:int-add iy incy))))
                      (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                              (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                 (* temp1
                                    (coerce (realpart
                                             (f2cl-lib:fref a-%data%
                                                            (kplus1 j)
                                                            ((1 lda) (1 *))
                                                            a-%offset%)) 'double-float))
                                 (* alpha temp2)))
                      (setf jx (f2cl-lib:int-add jx incx))
                      (setf jy (f2cl-lib:int-add jy incy))
                      (cond
                        ((> j k)
                         (setf kx (f2cl-lib:int-add kx incx))
                         (setf ky (f2cl-lib:int-add ky incy)))))))))
          (t
           (cond
             ((and (= incx 1) (= incy 1))
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (setf temp1
                          (* alpha
                             (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
                  (setf temp2 zero)
                  (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                          (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                             (* temp1
                                (coerce (realpart
                                         (f2cl-lib:fref a-%data%
                                                        (1 j)
                                                        ((1 lda) (1 *))
                                                        a-%offset%)) 'double-float))))
                  (setf l (f2cl-lib:int-sub 1 j))
```

```
                (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                               (f2cl-lib:int-add i 1))
                              ((> i
                                  (min (the fixnum n)
                                       (the fixnum
                                            (f2cl-lib:int-add j k))))
                               nil)
                  (tagbody
                    (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                            (+
                             (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                             (* temp1
                                (f2cl-lib:fref a-%data%
                                               ((f2cl-lib:int-add l i) j)
                                               ((1 lda) (1 *))
                                               a-%offset%))))
                    (setf temp2
                            (+ temp2
                               (*
                                (f2cl-lib:dconjg
                                 (f2cl-lib:fref a-%data%
                                                ((f2cl-lib:int-add l i) j)
                                                ((1 lda) (1 *))
                                                a-%offset%))
                                (f2cl-lib:fref x-%data%
                                               (i)
                                               ((1 *))
                                               x-%offset%))))))
                (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                        (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                           (* alpha temp2))))))
              (t
               (setf jx kx)
               (setf jy ky)
               (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                             ((> j n) nil)
                 (tagbody
                   (setf temp1
                           (* alpha
                              (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
                   (setf temp2 zero)
                   (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                           (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                              (* temp1
                                 (coerce (realpart
                                  (f2cl-lib:fref a-%data%
```

```
                                        (1 j)
                                        ((1 lda) (1 *))
                                        a-%offset%)) 'double-float))))
                    (setf l (f2cl-lib:int-sub 1 j))
                    (setf ix jx)
                    (setf iy jy)
                    (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                    (f2cl-lib:int-add i 1))
                                  ((> i
                                      (min (the fixnum n)
                                           (the fixnum
                                                (f2cl-lib:int-add j k))))
                                   nil)
                      (tagbody
                        (setf ix (f2cl-lib:int-add ix incx))
                        (setf iy (f2cl-lib:int-add iy incy))
                        (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                                (+
                                 (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                                 (* temp1
                                    (f2cl-lib:fref a-%data%
                                                   ((f2cl-lib:int-add l i) j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%))))
                        (setf temp2
                                (+ temp2
                                   (*
                                    (f2cl-lib:dconjg
                                     (f2cl-lib:fref a-%data%
                                                    ((f2cl-lib:int-add l i) j)
                                                    ((1 lda) (1 *))
                                                    a-%offset%))
                                    (f2cl-lib:fref x-%data%
                                                   (ix)
                                                   ((1 *))
                                                   x-%offset%))))))
                    (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                            (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                               (* alpha temp2)))
                    (setf jx (f2cl-lib:int-add jx incx))
                    (setf jy (f2cl-lib:int-add jy incy))))))))
  end_label
        (return (values nil nil nil nil nil nil nil nil nil nil nil))))))
```

5.22 zhemv BLAS

⟨*zhemv.input*⟩≡

```
)set break resume
)sys rm -f zhemv.output
)spool zhemv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zhemv.help*⟩≡

```
=====================================================================
zhemv examples
=====================================================================

=====================================================================
Man Page Details
=====================================================================

 NAME
      ZHEMV - perform the matrix-vector operation   y := alpha*A*x
      + beta*y,

 SYNOPSIS
      SUBROUTINE ZHEMV ( UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y,
                         INCY )

          COMPLEX*16   ALPHA, BETA

          INTEGER      INCX, INCY, LDA, N

          CHARACTER*1  UPLO

          COMPLEX*16   A( LDA, * ), X( * ), Y( * )

 PURPOSE
      ZHEMV  performs the matrix-vector  operation

      where alpha and beta are scalars, x and y are n element vec-
      tors and A is an n by n hermitian matrix.

 PARAMETERS
      UPLO   - CHARACTER*1.
             On entry, UPLO specifies whether the upper or lower
             triangular part of the array A is to be referenced as
             follows:

             UPLO = 'U' or 'u'   Only the upper triangular part of
             A is to be referenced.

             UPLO = 'L' or 'l'   Only the lower triangular part of
             A is to be referenced.

             Unchanged on exit.

      N      - INTEGER.
```

```
          On entry, N specifies the order of the matrix A.  N
          must be at least zero.  Unchanged on exit.

ALPHA  - COMPLEX*16      .
          On entry, ALPHA specifies the scalar alpha.
          Unchanged on exit.

A      - COMPLEX*16       array of DIMENSION ( LDA, n ).
          Before entry with  UPLO = 'U' or 'u', the leading n
          by n upper triangular part of the array A must con-
          tain the upper triangular part of the hermitian
          matrix and the strictly lower triangular part of A is
          not referenced.  Before entry with UPLO = 'L' or 'l',
          the leading n by n lower triangular part of the array

          A must contain the lower triangular part of the her-
          mitian matrix and the strictly upper triangular part
          of A is not referenced.  Note that the imaginary
          parts of the diagonal elements need not be set and
          are assumed to be zero.  Unchanged on exit.

LDA    - INTEGER.
          On entry, LDA specifies the first dimension of A as
          declared in the calling (sub) program. LDA must be at
          least max( 1, n ).  Unchanged on exit.

X      - COMPLEX*16       array of dimension at least
          ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
          incremented array X must contain the n element vector
          x.  Unchanged on exit.

INCX   - INTEGER.
          On entry, INCX specifies the increment for the ele-
          ments of X. INCX must not be zero.  Unchanged on
          exit.

BETA   - COMPLEX*16      .
          On entry, BETA specifies the scalar beta. When BETA
          is supplied as zero then Y need not be set on input.
          Unchanged on exit.

Y      - COMPLEX*16       array of dimension at least
          ( 1 + ( n - 1 )*abs( INCY ) ).  Before entry, the
          incremented array Y must contain the n element vector
          y. On exit, Y is overwritten by the updated vector y.
```

```
INCY   - INTEGER.
         On entry, INCY specifies the increment for the ele-
         ments of Y. INCY must not be zero.  Unchanged on
         exit.
```

⟨*BLAS 2 zhemv*⟩≡

```
(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
 (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zhemv (uplo n alpha a lda x incx beta y incy)
    (declare (type (simple-array (complex double-float) (*)) y x a)
             (type (complex double-float) beta alpha)
             (type fixnum incy incx lda n)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (x (complex double-float) x-%data% x-%offset%)
         (y (complex double-float) y-%data% y-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (kx 0) (ky 0)
             (temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)))
        (declare (type fixnum i info ix iy j jx jy kx ky)
                 (type (complex double-float) temp1 temp2))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info 5))
          ((= incx 0)
           (setf info 7))
          ((= incy 0)
           (setf info 10)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZHEMV" info)
           (go end_label)))
        (if (or (= n 0) (and (= alpha zero) (= beta one))) (go end_label))
        (cond
          ((> incx 0)
           (setf kx 1))
          (t
           (setf kx
                   (f2cl-lib:int-sub 1
                                     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                       incx)))))
        (cond
          ((> incy 0)
```

```
   (setf ky 1))
  (t
   (setf ky
         (f2cl-lib:int-sub 1
                           (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                             incy)))))
(cond
  ((/= beta one)
   (cond
     ((= incy 1)
      (cond
        ((= beta zero)
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i n) nil)
           (tagbody
             (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                     zero))))
        (t
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i n) nil)
           (tagbody
             (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                     (* beta
                        (f2cl-lib:fref y-%data%
                                       (i)
                                       ((1 *))
                                       y-%offset%)))))))))
     (t
      (setf iy ky)
      (cond
        ((= beta zero)
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i n) nil)
           (tagbody
             (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                     zero)
             (setf iy (f2cl-lib:int-add iy incy)))))
        (t
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i n) nil)
           (tagbody
             (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                     (* beta
                        (f2cl-lib:fref y-%data%
                                       (iy)
                                       ((1 *))
```

```
                                        y-%offset%)))
                (setf iy (f2cl-lib:int-add iy incy))))))))))
  (if (= alpha zero) (go end_label))
  (cond
    ((char-equal uplo #\U)
     (cond
       ((and (= incx 1) (= incy 1))
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                      ((> j n) nil)
          (tagbody
            (setf temp1
                    (* alpha
                       (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
            (setf temp2 zero)
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                          ((> i
                              (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                           nil)
              (tagbody
                (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                        (+
                         (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                         (* temp1
                            (f2cl-lib:fref a-%data%
                                           (i j)
                                           ((1 lda) (1 *))
                                           a-%offset%))))
                (setf temp2
                        (+ temp2
                           (*
                            (f2cl-lib:dconjg
                             (f2cl-lib:fref a-%data%
                                            (i j)
                                            ((1 lda) (1 *))
                                            a-%offset%))
                            (f2cl-lib:fref x-%data%
                                           (i)
                                           ((1 *))
                                           x-%offset%))))))
            (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                    (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                       (* temp1
                          (coerce (realpart
                           (f2cl-lib:fref a-%data%
                                          (j j)
                                          ((1 lda) (1 *))
```

```
                                        a-%offset%)) 'double-float))
                    (* alpha temp2))))))
               (t
                (setf jx kx)
                (setf jy ky)
                (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                              ((> j n) nil)
                  (tagbody
                    (setf temp1
                            (* alpha
                               (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
                    (setf temp2 zero)
                    (setf ix kx)
                    (setf iy ky)
                    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                  ((> i
                                      (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                   nil)
                      (tagbody
                        (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                                (+
                                 (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                                 (* temp1
                                    (f2cl-lib:fref a-%data%
                                                   (i j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%))))
                        (setf temp2
                                (+ temp2
                                   (*
                                    (f2cl-lib:dconjg
                                     (f2cl-lib:fref a-%data%
                                                    (i j)
                                                    ((1 lda) (1 *))
                                                    a-%offset%))
                                    (f2cl-lib:fref x-%data%
                                                   (ix)
                                                   ((1 *))
                                                   x-%offset%))))
                        (setf ix (f2cl-lib:int-add ix incx))
                        (setf iy (f2cl-lib:int-add iy incy))))
                    (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                            (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                               (* temp1
                                  (coerce (realpart
                                   (f2cl-lib:fref a-%data%
```

```
                                          (j j)
                                          ((1 lda) (1 *))
                                          a-%offset%)) 'double-float))
                        (* alpha temp2)))
               (setf jx (f2cl-lib:int-add jx incx))
               (setf jy (f2cl-lib:int-add jy incy)))))))
      (t
       (cond
         ((and (= incx 1) (= incy 1))
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                        ((> j n) nil)
            (tagbody
              (setf temp1
                      (* alpha
                         (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
              (setf temp2 zero)
              (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                      (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                         (* temp1
                            (coerce (realpart
                             (f2cl-lib:fref a-%data%
                                          (j j)
                                          ((1 lda) (1 *))
                                          a-%offset%)) 'double-float))))
              (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                             (f2cl-lib:int-add i 1))
                            ((> i n) nil)
                (tagbody
                  (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                          (+
                           (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                           (* temp1
                              (f2cl-lib:fref a-%data%
                                           (i j)
                                           ((1 lda) (1 *))
                                           a-%offset%))))
                  (setf temp2
                          (+ temp2
                             (*
                              (f2cl-lib:dconjg
                               (f2cl-lib:fref a-%data%
                                             (i j)
                                             ((1 lda) (1 *))
                                             a-%offset%))
                              (f2cl-lib:fref x-%data%
                                             (i)
```

```
                                        ((1 *))
                                        x-%offset%))))))
          (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                  (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                     (* alpha temp2))))))
        (t
         (setf jx kx)
         (setf jy ky)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (setf temp1
                     (* alpha
                        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
             (setf temp2 zero)
             (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                     (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                        (* temp1
                           (coerce (realpart
                            (f2cl-lib:fref a-%data%
                                           (j j)
                                           ((1 lda) (1 *))
                                           a-%offset%)) 'double-float))))
             (setf ix jx)
             (setf iy jy)
             (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                              (f2cl-lib:int-add i 1))
                           ((> i n) nil)
               (tagbody
                 (setf ix (f2cl-lib:int-add ix incx))
                 (setf iy (f2cl-lib:int-add iy incy))
                 (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                         (+
                          (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                          (* temp1
                             (f2cl-lib:fref a-%data%
                                            (i j)
                                            ((1 lda) (1 *))
                                            a-%offset%))))
                 (setf temp2
                         (+ temp2
                            (*
                             (f2cl-lib:dconjg
                              (f2cl-lib:fref a-%data%
                                             (i j)
                                             ((1 lda) (1 *))
```

```
                                                a-%offset%))
                                      (f2cl-lib:fref x-%data%
                                                     (ix)
                                                     ((1 *))
                                                     x-%offset%))))))
                        (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                   (* alpha temp2)))
                        (setf jx (f2cl-lib:int-add jx incx))
                        (setf jy (f2cl-lib:int-add jy incy))))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil nil nil))))))
```

5.23 zher2 BLAS

⟨*zher2.input*⟩≡

```
)set break resume
)sys rm -f zher2.output
)spool zher2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zher2.help*⟩≡

```
==========================================================================
zher2 examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

 NAME
      ZHER2 - perform the hermitian rank 2 operation   A :=
      alpha*x*conjg( y' ) + conjg( alpha )*y*conjg( x' ) + A,

 SYNOPSIS
      SUBROUTINE ZHER2 ( UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA
                       )

          COMPLEX*16   ALPHA

          INTEGER      INCX, INCY, LDA, N

          CHARACTER*1  UPLO

          COMPLEX*16   A( LDA, * ), X( * ), Y( * )

 PURPOSE
      ZHER2  performs the hermitian rank 2 operation

      where alpha is a scalar, x and y are n element vectors and A
      is an n by n hermitian matrix.

 PARAMETERS
      UPLO   - CHARACTER*1.
             On entry, UPLO specifies whether the upper or lower
             triangular part of the array A is to be referenced as
             follows:

             UPLO = 'U' or 'u'   Only the upper triangular part of
             A is to be referenced.

             UPLO = 'L' or 'l'   Only the lower triangular part of
             A is to be referenced.

             Unchanged on exit.

      N      - INTEGER.
```

```
          On entry, N specifies the order of the matrix A.  N
          must be at least zero.  Unchanged on exit.

ALPHA  - COMPLEX*16      .
          On entry, ALPHA specifies the scalar alpha.
          Unchanged on exit.

X      - COMPLEX*16       array of dimension at least
          ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
          incremented array X must contain the n element vector
          x.  Unchanged on exit.

INCX   - INTEGER.
          On entry, INCX specifies the increment for the

          elements of X. INCX must not be zero.  Unchanged on
          exit.

Y      - COMPLEX*16       array of dimension at least
          ( 1 + ( n - 1 )*abs( INCY ) ).  Before entry, the
          incremented array Y must contain the n element vector
          y.  Unchanged on exit.

INCY   - INTEGER.
          On entry, INCY specifies the increment for the ele-
          ments of Y. INCY must not be zero.  Unchanged on
          exit.

A      - COMPLEX*16       array of DIMENSION ( LDA, n ).
          Before entry with  UPLO = 'U' or 'u', the leading n
          by n upper triangular part of the array A must con-
          tain the upper triangular part of the hermitian
          matrix and the strictly lower triangular part of A is
          not referenced. On exit, the upper triangular part of
          the array A is overwritten by the upper triangular
          part of the updated matrix.  Before entry with UPLO =
          'L' or 'l', the leading n by n lower triangular part
          of the array A must contain the lower triangular part
          of the hermitian matrix and the strictly upper tri-
          angular part of A is not referenced. On exit, the
          lower triangular part of the array A is overwritten
          by the lower triangular part of the updated matrix.
          Note that the imaginary parts of the diagonal ele-
          ments need not be set, they are assumed to be zero,
          and on exit they are set to zero.
```

```
LDA    - INTEGER.
       On entry, LDA specifies the first dimension of A as
       declared in the calling (sub) program. LDA must be at
       least max( 1, n ).  Unchanged on exit.
```

⟨*BLAS 2 zher2*⟩≡

```
(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun zher2 (uplo n alpha x incx y incy a lda)
    (declare (type (simple-array (complex double-float) (*)) a y x)
             (type (complex double-float) alpha)
             (type fixnum lda incy incx n)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (x (complex double-float) x-%data% x-%offset%)
         (y (complex double-float) y-%data% y-%offset%)
         (a (complex double-float) a-%data% a-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (kx 0) (ky 0)
             (temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)))
        (declare (type fixnum i info ix iy j jx jy kx ky)
                 (type (complex double-float) temp1 temp2))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((= incx 0)
           (setf info 5))
          ((= incy 0)
           (setf info 7))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info 9)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZHER2" info)
           (go end_label)))
        (if (or (= n 0) (= alpha zero)) (go end_label))
        (cond
          ((or (/= incx 1) (/= incy 1))
           (cond
             ((> incx 0)
              (setf kx 1))
             (t
              (setf kx
                      (f2cl-lib:int-sub 1
                                        (f2cl-lib:int-mul
                                         (f2cl-lib:int-sub n 1)
```

```
                                        incx)))))
                (cond
                  ((> incy 0)
                   (setf ky 1))
                  (t
                   (setf ky
                         (f2cl-lib:int-sub 1
                                           (f2cl-lib:int-mul
                                            (f2cl-lib:int-sub n 1)
                                            incy)))))
                (setf jx kx)
                (setf jy ky)))
           (cond
             ((char-equal uplo #\U)
              (cond
                ((and (= incx 1) (= incy 1))
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (cond
                       ((or (/= (f2cl-lib:fref x (j) ((1 *))) zero)
                            (/= (f2cl-lib:fref y (j) ((1 *))) zero))
                        (setf temp1
                                (* alpha
                                   (f2cl-lib:dconjg
                                    (f2cl-lib:fref y-%data%
                                                   (j)
                                                   ((1 *))
                                                   y-%offset%))))
                        (setf temp2
                                (coerce
                                 (f2cl-lib:dconjg
                                  (* alpha
                                     (f2cl-lib:fref x-%data%
                                                    (j)
                                                    ((1 *))
                                                    x-%offset%)))
                                 '(complex double-float)))
                        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                      ((> i
                                          (f2cl-lib:int-add j
                                                            (f2cl-lib:int-sub 1)))
                                       nil)
                          (tagbody
                            (setf (f2cl-lib:fref a-%data%
                                                 (i j)
```

```
                                    ((1 lda) (1 *))
                                    a-%offset%)
                                (+
                                 (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%)
                                 (*
                                  (f2cl-lib:fref x-%data%
                                                 (i)
                                                 ((1 *))
                                                 x-%offset%)
                                  temp1)
                                 (*
                                  (f2cl-lib:fref y-%data%
                                                 (i)
                                                 ((1 *))
                                                 y-%offset%)
                                  temp2)))))
                          (setf (f2cl-lib:fref a-%data%
                                               (j j)
                                               ((1 lda) (1 *))
                                               a-%offset%)
                                (coerce
                                 (+
                                  (coerce (realpart
                                   (f2cl-lib:fref a-%data%
                                                  (j j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%)) 'double-float)
                                  (coerce (realpart
                                   (+
                                    (*
                                     (f2cl-lib:fref x-%data%
                                                    (j)
                                                    ((1 *))
                                                    x-%offset%)
                                     temp1)
                                    (*
                                     (f2cl-lib:fref y-%data%
                                                    (j)
                                                    ((1 *))
                                                    y-%offset%)
                                     temp2))) 'double-float))
                                 '(complex double-float))))
                          (t
```

```
                (setf (f2cl-lib:fref a-%data%
                                     (j j)
                                     ((1 lda) (1 *))
                                     a-%offset%)
                      (coerce
                       (coerce (realpart
                        (f2cl-lib:fref a-%data%
                                       (j j)
                                       ((1 lda) (1 *))
                                       a-%offset%)) 'double-float)
                       '(complex double-float))))))))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (cond
                 ((or (/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                      (/= (f2cl-lib:fref y (jy) ((1 *))) zero))
                  (setf temp1
                        (* alpha
                           (f2cl-lib:dconjg
                            (f2cl-lib:fref y-%data%
                                           (jy)
                                           ((1 *))
                                           y-%offset%))))
                  (setf temp2
                        (coerce
                         (f2cl-lib:dconjg
                          (* alpha
                             (f2cl-lib:fref x-%data%
                                            (jx)
                                            ((1 *))
                                            x-%offset%)))
                         '(complex double-float)))
                  (setf ix kx)
                  (setf iy ky)
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i
                                    (f2cl-lib:int-add j
                                                      (f2cl-lib:int-sub 1)))
                                 nil)
                    (tagbody
                      (setf (f2cl-lib:fref a-%data%
                                           (i j)
                                           ((1 lda) (1 *))
                                           a-%offset%)
```

```
                        (+
                         (f2cl-lib:fref a-%data%
                                        (i j)
                                        ((1 lda) (1 *))
                                        a-%offset%)
                         (*
                          (f2cl-lib:fref x-%data%
                                         (ix)
                                         ((1 *))
                                         x-%offset%)
                          temp1)
                         (*
                          (f2cl-lib:fref y-%data%
                                         (iy)
                                         ((1 *))
                                         y-%offset%)
                          temp2)))
                    (setf ix (f2cl-lib:int-add ix incx))
                    (setf iy (f2cl-lib:int-add iy incy))))
                (setf (f2cl-lib:fref a-%data%
                                     (j j)
                                     ((1 lda) (1 *))
                                     a-%offset%)
                      (coerce
                       (+
                        (coerce (realpart
                         (f2cl-lib:fref a-%data%
                                        (j j)
                                        ((1 lda) (1 *))
                                        a-%offset%)) 'double-float)
                        (coerce (realpart
                         (+
                          (*
                           (f2cl-lib:fref x-%data%
                                          (jx)
                                          ((1 *))
                                          x-%offset%)
                           temp1)
                          (*
                           (f2cl-lib:fref y-%data%
                                          (jy)
                                          ((1 *))
                                          y-%offset%)
                           temp2))) 'double-float))
                        '(complex double-float))))
                (t
```

```
                (setf (f2cl-lib:fref a-%data%
                                     (j j)
                                     ((1 lda) (1 *))
                                     a-%offset%)
                      (coerce
                       (coerce (realpart
                        (f2cl-lib:fref a-%data%
                                       (j j)
                                       ((1 lda) (1 *))
                                       a-%offset%)) 'double-float)
                       '(complex double-float)))))
              (setf jx (f2cl-lib:int-add jx incx))
              (setf jy (f2cl-lib:int-add jy incy)))))))
      (t
       (cond
         ((and (= incx 1) (= incy 1))
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                        ((> j n) nil)
            (tagbody
              (cond
                ((or (/= (f2cl-lib:fref x (j) ((1 *))) zero)
                     (/= (f2cl-lib:fref y (j) ((1 *))) zero))
                 (setf temp1
                       (* alpha
                          (f2cl-lib:dconjg
                           (f2cl-lib:fref y-%data%
                                          (j)
                                          ((1 *))
                                          y-%offset%))))
                 (setf temp2
                       (coerce
                        (f2cl-lib:dconjg
                         (* alpha
                            (f2cl-lib:fref x-%data%
                                           (j)
                                           ((1 *))
                                           x-%offset%)))
                        '(complex double-float)))
                 (setf (f2cl-lib:fref a-%data%
                                      (j j)
                                      ((1 lda) (1 *))
                                      a-%offset%)
                       (coerce
                        (+
                         (coerce (realpart
                          (f2cl-lib:fref a-%data%
```

```
                                        (j j)
                                        ((1 lda) (1 *))
                                        a-%offset%)) 'double-float)
                        (coerce (realpart
                                 (+
                                  (*
                                   (f2cl-lib:fref x-%data%
                                                  (j)
                                                  ((1 *))
                                                  x-%offset%)
                                   temp1)
                                  (*
                                   (f2cl-lib:fref y-%data%
                                                  (j)
                                                  ((1 *))
                                                  y-%offset%)
                                   temp2))) 'double-float))
                                '(complex double-float)))
                  (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                   (f2cl-lib:int-add i 1))
                                ((> i n) nil)
                    (tagbody
                      (setf (f2cl-lib:fref a-%data%
                                           (i j)
                                           ((1 lda) (1 *))
                                           a-%offset%)
                              (+
                               (f2cl-lib:fref a-%data%
                                              (i j)
                                              ((1 lda) (1 *))
                                              a-%offset%)
                               (*
                                (f2cl-lib:fref x-%data%
                                               (i)
                                               ((1 *))
                                               x-%offset%)
                                temp1)
                               (*
                                (f2cl-lib:fref y-%data%
                                               (i)
                                               ((1 *))
                                               y-%offset%)
                                temp2))))))
                 (t
                  (setf (f2cl-lib:fref a-%data%
                                       (j j)
```

```
                                    ((1 lda) (1 *))
                                    a-%offset%)
                            (coerce
                             (coerce (realpart
                              (f2cl-lib:fref a-%data%
                                             (j j)
                                             ((1 lda) (1 *))
                                             a-%offset%)) 'double-float)
                             '(complex double-float))))))))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (cond
                 ((or (/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                      (/= (f2cl-lib:fref y (jy) ((1 *))) zero))
                  (setf temp1
                          (* alpha
                             (f2cl-lib:dconjg
                              (f2cl-lib:fref y-%data%
                                             (jy)
                                             ((1 *))
                                             y-%offset%))))
                  (setf temp2
                          (coerce
                           (f2cl-lib:dconjg
                            (* alpha
                               (f2cl-lib:fref x-%data%
                                              (jx)
                                              ((1 *))
                                              x-%offset%)))
                           '(complex double-float)))
                  (setf (f2cl-lib:fref a-%data%
                                       (j j)
                                       ((1 lda) (1 *))
                                       a-%offset%)
                          (coerce
                           (+
                            (coerce (realpart
                             (f2cl-lib:fref a-%data%
                                            (j j)
                                            ((1 lda) (1 *))
                                            a-%offset%)) 'double-float)
                            (coerce (realpart
                             (+
                              (*
```

```
                (f2cl-lib:fref x-%data%
                               (jx)
                               ((1 *))
                               x-%offset%)
                temp1)
               (*
                (f2cl-lib:fref y-%data%
                               (jy)
                               ((1 *))
                               y-%offset%)
                temp2))) 'double-float))
          '(complex double-float)))
 (setf ix jx)
 (setf iy jy)
 (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                  (f2cl-lib:int-add i 1))
                ((> i n) nil)
   (tagbody
     (setf ix (f2cl-lib:int-add ix incx))
     (setf iy (f2cl-lib:int-add iy incy))
     (setf (f2cl-lib:fref a-%data%
                          (i j)
                          ((1 lda) (1 *))
                          a-%offset%)
             (+
              (f2cl-lib:fref a-%data%
                             (i j)
                             ((1 lda) (1 *))
                             a-%offset%)
              (*
               (f2cl-lib:fref x-%data%
                              (ix)
                              ((1 *))
                              x-%offset%)
               temp1)
              (*
               (f2cl-lib:fref y-%data%
                              (iy)
                              ((1 *))
                              y-%offset%)
               temp2))))))
(t
 (setf (f2cl-lib:fref a-%data%
                      (j j)
                      ((1 lda) (1 *))
                      a-%offset%)
```

```
                                (coerce
                                 (coerce (realpart
                                  (f2cl-lib:fref a-%data%
                                                 (j j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%)) 'double-float)
                                 '(complex double-float)))))
                    (setf jx (f2cl-lib:int-add jx incx))
                    (setf jy (f2cl-lib:int-add jy incy))))))))
   end_label
          (return (values nil nil nil nil nil nil nil nil nil))))))
```

5.24 zher BLAS

⟨*zher.input*⟩≡

```
  )set break resume
  )sys rm -f zher.output
  )spool zher.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*zher.help*⟩≡

```
  ======================================================================
  zher examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        ZHER - perform the hermitian rank 1 operation   A :=
        alpha*x*conjg( x' ) + A,

   SYNOPSIS
        SUBROUTINE ZHER ( UPLO, N, ALPHA, X, INCX, A, LDA )

            DOUBLE      PRECISION ALPHA

            INTEGER     INCX, LDA, N

            CHARACTER*1 UPLO

            COMPLEX*16  A( LDA, * ), X( * )

   PURPOSE
        ZHER   performs the hermitian rank 1 operation

        where alpha is a real scalar, x is an n element vector and A
        is an n by n hermitian matrix.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On entry, UPLO specifies whether the upper or lower
                 triangular part of the array A is to be referenced as
                 follows:

                 UPLO = 'U' or 'u'   Only the upper triangular part of
                 A is to be referenced.

                 UPLO = 'L' or 'l'   Only the lower triangular part of
                 A is to be referenced.

                 Unchanged on exit.

        N      - INTEGER.
                 On entry, N specifies the order of the matrix A.  N
```

```
       must be at least zero.  Unchanged on exit.

ALPHA  - DOUBLE PRECISION.
       On entry, ALPHA specifies the scalar alpha.
       Unchanged on exit.

X      - COMPLEX*16       array of dimension at least
       ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
       incremented array X must contain the n element vector
       x.  Unchanged on exit.

INCX   - INTEGER.
       On entry, INCX specifies the increment for the ele-
       ments of X. INCX must not be zero.  Unchanged on

       exit.

A      - COMPLEX*16       array of DIMENSION ( LDA, n ).
       Before entry with  UPLO = 'U' or 'u', the leading n
       by n upper triangular part of the array A must con-
       tain the upper triangular part of the hermitian
       matrix and the strictly lower triangular part of A is
       not referenced. On exit, the upper triangular part of
       the array A is overwritten by the upper triangular
       part of the updated matrix.  Before entry with UPLO =
       'L' or 'l', the leading n by n lower triangular part
       of the array A must contain the lower triangular part
       of the hermitian matrix and the strictly upper tri-
       angular part of A is not referenced. On exit, the
       lower triangular part of the array A is overwritten
       by the lower triangular part of the updated matrix.
       Note that the imaginary parts of the diagonal ele-
       ments need not be set, they are assumed to be zero,
       and on exit they are set to zero.

LDA    - INTEGER.
       On entry, LDA specifies the first dimension of A as
       declared in the calling (sub) program. LDA must be at
       least max( 1, n ).  Unchanged on exit.
```

⟨*BLAS 2 zher*⟩≡

```
(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun zher (uplo n alpha x incx a lda)
    (declare (type (simple-array (complex double-float) (*)) a x)
             (type (double-float) alpha)
             (type fixnum lda incx n)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (x (complex double-float) x-%data% x-%offset%)
         (a (complex double-float) a-%data% a-%offset%))
      (prog ((i 0) (info 0) (ix 0) (j 0) (jx 0) (kx 0) (temp #C(0.0 0.0)))
        (declare (type fixnum i info ix j jx kx)
                 (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((= incx 0)
           (setf info 5))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info 7)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZHER" info)
           (go end_label)))
        (if (or (= n 0) (= alpha (coerce (realpart zero) 'double-float)))
           (go end_label))
        (cond
          ((<= incx 0)
           (setf kx
                   (f2cl-lib:int-sub 1
                                     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                       incx))))
          ((/= incx 1)
           (setf kx 1)))
        (cond
          ((char-equal uplo #\U)
           (cond
             ((= incx 1)
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
```

```
            ((> j n) nil)
          (tagbody
            (cond
              ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
               (setf temp
                       (coerce
                        (* alpha
                           (f2cl-lib:dconjg
                            (f2cl-lib:fref x-%data%
                                           (j)
                                           ((1 *))
                                           x-%offset%)))
                        '(complex double-float)))
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i
                                 (f2cl-lib:int-add j
                                                   (f2cl-lib:int-sub 1)))
                              nil)
                 (tagbody
                   (setf (f2cl-lib:fref a-%data%
                                        (i j)
                                        ((1 lda) (1 *))
                                        a-%offset%)
                         (+
                          (f2cl-lib:fref a-%data%
                                         (i j)
                                         ((1 lda) (1 *))
                                         a-%offset%)
                          (*
                           (f2cl-lib:fref x-%data%
                                          (i)
                                          ((1 *))
                                          x-%offset%)
                           temp)))))
               (setf (f2cl-lib:fref a-%data%
                                    (j j)
                                    ((1 lda) (1 *))
                                    a-%offset%)
                       (coerce
                        (+
                         (coerce (realpart
                          (f2cl-lib:fref a-%data%
                                         (j j)
                                         ((1 lda) (1 *))
                                         a-%offset%)) 'double-float)
                         (coerce (realpart
```

```
                        (*
                         (f2cl-lib:fref x-%data%
                                        (j)
                                        ((1 *))
                                        x-%offset%)
                         temp)) 'double-float))
                       '(complex double-float))))
              (t
               (setf (f2cl-lib:fref a-%data%
                                    (j j)
                                    ((1 lda) (1 *))
                                    a-%offset%)
                     (coerce
                      (coerce (realpart
                               (f2cl-lib:fref a-%data%
                                              (j j)
                                              ((1 lda) (1 *))
                                              a-%offset%)) 'double-float)
                      '(complex double-float))))))))
    (t
     (setf jx kx)
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   ((> j n) nil)
       (tagbody
         (cond
           ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
            (setf temp
                    (coerce
                     (* alpha
                        (f2cl-lib:dconjg
                         (f2cl-lib:fref x-%data%
                                        (jx)
                                        ((1 *))
                                        x-%offset%)))
                     '(complex double-float)))
            (setf ix kx)
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                          ((> i
                              (f2cl-lib:int-add j
                                                (f2cl-lib:int-sub 1)))
                           nil)
              (tagbody
                (setf (f2cl-lib:fref a-%data%
                                     (i j)
                                     ((1 lda) (1 *))
                                     a-%offset%)
```

```
                                    (+
                                     (f2cl-lib:fref a-%data%
                                                    (i j)
                                                    ((1 lda) (1 *))
                                                    a-%offset%)
                                     (*
                                      (f2cl-lib:fref x-%data%
                                                     (ix)
                                                     ((1 *))
                                                     x-%offset%)
                                      temp)))
                                (setf ix (f2cl-lib:int-add ix incx))))
                           (setf (f2cl-lib:fref a-%data%
                                                (j j)
                                                ((1 lda) (1 *))
                                                a-%offset%)
                                 (coerce
                                  (+
                                   (coerce (realpart
                                    (f2cl-lib:fref a-%data%
                                                   (j j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%)) 'double-float)
                                   (coerce (realpart
                                    (*
                                     (f2cl-lib:fref x-%data%
                                                    (jx)
                                                    ((1 *))
                                                    x-%offset%)
                                     temp)) 'double-float))
                                  '(complex double-float))))
                          (t
                           (setf (f2cl-lib:fref a-%data%
                                                (j j)
                                                ((1 lda) (1 *))
                                                a-%offset%)
                                 (coerce
                                  (coerce (realpart
                                   (f2cl-lib:fref a-%data%
                                                  (j j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%)) 'double-float)
                                  '(complex double-float)))))
                        (setf jx (f2cl-lib:int-add jx incx)))))))
                (t
                 (cond
```

```
((= incx 1)
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j n) nil)
   (tagbody
     (cond
       ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
        (setf temp
                (coerce
                 (* alpha
                    (f2cl-lib:dconjg
                     (f2cl-lib:fref x-%data%
                                    (j)
                                    ((1 *))
                                    x-%offset%)))
                 '(complex double-float)))
        (setf (f2cl-lib:fref a-%data%
                             (j j)
                             ((1 lda) (1 *))
                             a-%offset%)
                (coerce
                 (+
                  (coerce (realpart
                   (f2cl-lib:fref a-%data%
                                  (j j)
                                  ((1 lda) (1 *))
                                  a-%offset%)) 'double-float)
                  (coerce (realpart
                   (* temp
                      (f2cl-lib:fref x-%data%
                                     (j)
                                     ((1 *))
                                     x-%offset%))) 'double-float))
                 '(complex double-float)))
        (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                         (f2cl-lib:int-add i 1))
                      ((> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref a-%data%
                                 (i j)
                                 ((1 lda) (1 *))
                                 a-%offset%)
                    (+
                     (f2cl-lib:fref a-%data%
                                    (i j)
                                    ((1 lda) (1 *))
                                    a-%offset%)
```

```
                                (*
                                 (f2cl-lib:fref x-%data%
                                                (i)
                                                ((1 *))
                                                x-%offset%)
                                 temp))))))
                        (t
                         (setf (f2cl-lib:fref a-%data%
                                              (j j)
                                              ((1 lda) (1 *))
                                              a-%offset%)
                                (coerce
                                 (coerce (realpart
                                  (f2cl-lib:fref a-%data%
                                                 (j j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%)) 'double-float)
                                 '(complex double-float)))))))
(t
 (setf jx kx)
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j n) nil)
   (tagbody
     (cond
       ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
        (setf temp
                (coerce
                 (* alpha
                    (f2cl-lib:dconjg
                     (f2cl-lib:fref x-%data%
                                    (jx)
                                    ((1 *))
                                    x-%offset%)))
                 '(complex double-float)))
        (setf (f2cl-lib:fref a-%data%
                             (j j)
                             ((1 lda) (1 *))
                             a-%offset%)
                (coerce
                 (+
                  (coerce (realpart
                   (f2cl-lib:fref a-%data%
                                  (j j)
                                  ((1 lda) (1 *))
                                  a-%offset%)) 'double-float)
                  (coerce (realpart
```

```
                                (* temp
                                   (f2cl-lib:fref x-%data%
                                                  (jx)
                                                  ((1 *))
                                                  x-%offset%))) 'double-float))
                             '(complex double-float)))
                      (setf ix jx)
                      (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                       (f2cl-lib:int-add i 1))
                                    ((> i n) nil)
                        (tagbody
                          (setf ix (f2cl-lib:int-add ix incx))
                          (setf (f2cl-lib:fref a-%data%
                                               (i j)
                                               ((1 lda) (1 *))
                                               a-%offset%)
                                (+
                                 (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%)
                                 (*
                                  (f2cl-lib:fref x-%data%
                                                 (ix)
                                                 ((1 *))
                                                 x-%offset%)
                                  temp))))))
                     (t
                      (setf (f2cl-lib:fref a-%data%
                                           (j j)
                                           ((1 lda) (1 *))
                                           a-%offset%)
                            (coerce
                             (coerce (realpart
                                      (f2cl-lib:fref a-%data%
                                                     (j j)
                                                     ((1 lda) (1 *))
                                                     a-%offset%)) 'double-float)
                             '(complex double-float)))))
                    (setf jx (f2cl-lib:int-add jx incx))))))))
  end_label
       (return (values nil nil nil nil nil nil nil))))))
```

5.25 zhpmv BLAS

⟨*zhpmv.input*⟩≡

```
)set break resume
)sys rm -f zhpmv.output
)spool zhpmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zhpmv.help*⟩≡

```
  ======================================================================
  zhpmv examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        ZHPMV - perform the matrix-vector operation   y := alpha*A*x
        + beta*y,

   SYNOPSIS
        SUBROUTINE ZHPMV ( UPLO, N, ALPHA, AP, X, INCX, BETA, Y,
                           INCY )

            COMPLEX*16   ALPHA, BETA

            INTEGER      INCX, INCY, N

            CHARACTER*1  UPLO

            COMPLEX*16   AP( * ), X( * ), Y( * )

   PURPOSE
        ZHPMV  performs the matrix-vector operation

        where alpha and beta are scalars, x and y are n element vec-
        tors and A is an n by n hermitian matrix, supplied in packed
        form.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On entry, UPLO specifies whether the upper or lower
                 triangular part of the matrix A is supplied in the
                 packed array AP as follows:

                 UPLO = 'U' or 'u'   The upper triangular part of A is
                 supplied in AP.

                 UPLO = 'L' or 'l'   The lower triangular part of A is
                 supplied in AP.

                 Unchanged on exit.
```

```
N      - INTEGER.
       On entry, N specifies the order of the matrix A.  N
       must be at least zero.  Unchanged on exit.

ALPHA  - COMPLEX*16      .
       On entry, ALPHA specifies the scalar alpha.
       Unchanged on exit.

AP     - COMPLEX*16       array of DIMENSION at least
       ( ( n*( n + 1 ) )/2 ).  Before entry with UPLO = 'U'
       or 'u', the array AP must contain the upper triangu-
       lar part of the hermitian matrix packed sequentially,
       column by column, so that AP( 1 ) contains a( 1, 1 ),
       AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 )

       respectively, and so on.  Before entry with UPLO =
       'L' or 'l', the array AP must contain the lower tri-
       angular part of the hermitian matrix packed sequen-
       tially, column by column, so that AP( 1 ) contains a(
       1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1 ) and a(
       3, 1 ) respectively, and so on.  Note that the ima-
       ginary parts of the diagonal elements need not be set
       and are assumed to be zero.  Unchanged on exit.

X      - COMPLEX*16       array of dimension at least
       ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
       incremented array X must contain the n element vector
       x.  Unchanged on exit.

INCX   - INTEGER.
       On entry, INCX specifies the increment for the ele-
       ments of X. INCX must not be zero.  Unchanged on
       exit.

BETA   - COMPLEX*16      .
       On entry, BETA specifies the scalar beta. When BETA
       is supplied as zero then Y need not be set on input.
       Unchanged on exit.

Y      - COMPLEX*16       array of dimension at least
       ( 1 + ( n - 1 )*abs( INCY ) ).  Before entry, the
       incremented array Y must contain the n element vector
       y. On exit, Y is overwritten by the updated vector y.

INCY   - INTEGER.
       On entry, INCY specifies the increment for the ele-
```

```
ments of Y. INCY must not be zero.  Unchanged on
exit.
```

⟨*BLAS 2 zhpmv*⟩≡

```
(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
 (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zhpmv (uplo n alpha ap x incx beta y incy)
    (declare (type (simple-array (complex double-float) (*)) y x ap)
             (type (complex double-float) beta alpha)
             (type fixnum incy incx n)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (ap (complex double-float) ap-%data% ap-%offset%)
         (x (complex double-float) x-%data% x-%offset%)
         (y (complex double-float) y-%data% y-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (k 0) (kk 0)
             (kx 0) (ky 0) (temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)))
        (declare (type fixnum i info ix iy j jx jy k kk kx ky)
                 (type (complex double-float) temp1 temp2))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((= incx 0)
           (setf info 6))
          ((= incy 0)
           (setf info 9)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZHPMV" info)
           (go end_label)))
        (if (or (= n 0) (and (= alpha zero) (= beta one))) (go end_label))
        (cond
          ((> incx 0)
           (setf kx 1))
          (t
           (setf kx
                   (f2cl-lib:int-sub 1
                                     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                       incx)))))
        (cond
          ((> incy 0)
           (setf ky 1))
          (t
```

```
            (setf ky
                  (f2cl-lib:int-sub 1
                                    (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                      incy)))))
          (cond
            ((/= beta one)
             (cond
               ((= incy 1)
                (cond
                  ((= beta zero)
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                               zero))))
                  (t
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                               (* beta
                                  (f2cl-lib:fref y-%data%
                                                 (i)
                                                 ((1 *))
                                                 y-%offset%)))))))))
               (t
                (setf iy ky)
                (cond
                  ((= beta zero)
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                               zero)
                       (setf iy (f2cl-lib:int-add iy incy)))))
                  (t
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                               (* beta
                                  (f2cl-lib:fref y-%data%
                                                 (iy)
                                                 ((1 *))
                                                 y-%offset%)))
                       (setf iy (f2cl-lib:int-add iy incy)))))))))))
```

```
(if (= alpha zero) (go end_label))
(setf kk 1)
(cond
  ((char-equal uplo #\U)
   (cond
     ((and (= incx 1) (= incy 1))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (setf temp1
                  (* alpha
                     (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
          (setf temp2 zero)
          (setf k kk)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i
                            (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                         nil)
            (tagbody
              (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                      (+
                       (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                       (* temp1
                          (f2cl-lib:fref ap-%data%
                                         (k)
                                         ((1 *))
                                         ap-%offset%))))
              (setf temp2
                      (+ temp2
                         (*
                          (f2cl-lib:dconjg
                           (f2cl-lib:fref ap-%data%
                                          (k)
                                          ((1 *))
                                          ap-%offset%))
                          (f2cl-lib:fref x-%data%
                                         (i)
                                         ((1 *))
                                         x-%offset%))))
              (setf k (f2cl-lib:int-add k 1))))
          (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                  (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                     (* temp1
                        (coerce (realpart
                         (f2cl-lib:fref ap-%data%
                                        ((f2cl-lib:int-sub
```

```
                                        (f2cl-lib:int-add kk j)
                                        1))
                                      ((1 *))
                                      ap-%offset%)) 'double-float))
                  (* alpha temp2)))
      (setf kk (f2cl-lib:int-add kk j)))))
 (t
  (setf jx kx)
  (setf jy ky)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                ((> j n) nil)
    (tagbody
      (setf temp1
              (* alpha
                 (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
      (setf temp2 zero)
      (setf ix kx)
      (setf iy ky)
      (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
                    ((> k
                        (f2cl-lib:int-add kk
                                          j
                                          (f2cl-lib:int-sub 2)))
                     nil)
        (tagbody
          (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                  (+
                   (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                   (* temp1
                      (f2cl-lib:fref ap-%data%
                                     (k)
                                     ((1 *))
                                     ap-%offset%))))
          (setf temp2
                  (+ temp2
                     (*
                      (f2cl-lib:dconjg
                       (f2cl-lib:fref ap-%data%
                                      (k)
                                      ((1 *))
                                      ap-%offset%))
                      (f2cl-lib:fref x-%data%
                                     (ix)
                                     ((1 *))
                                     x-%offset%))))
          (setf ix (f2cl-lib:int-add ix incx))
```

```
                    (setf iy (f2cl-lib:int-add iy incy))))
                (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                        (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                           (* temp1
                              (coerce (realpart
                               (f2cl-lib:fref ap-%data%
                                              ((f2cl-lib:int-sub
                                                (f2cl-lib:int-add kk j)
                                                1))
                                              ((1 *))
                                              ap-%offset%)) 'double-float))
                           (* alpha temp2)))
                (setf jx (f2cl-lib:int-add jx incx))
                (setf jy (f2cl-lib:int-add jy incy))
                (setf kk (f2cl-lib:int-add kk j)))))))
          (t
           (cond
             ((and (= incx 1) (= incy 1))
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (setf temp1
                          (* alpha
                             (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
                  (setf temp2 zero)
                  (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                          (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                             (* temp1
                                (coerce (realpart
                                 (f2cl-lib:fref ap-%data%
                                                (kk)
                                                ((1 *))
                                                ap-%offset%)) 'double-float))))
                  (setf k (f2cl-lib:int-add kk 1))
                  (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                   (f2cl-lib:int-add i 1))
                                ((> i n) nil)
                    (tagbody
                      (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                              (+
                               (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                               (* temp1
                                  (f2cl-lib:fref ap-%data%
                                                 (k)
                                                 ((1 *))
                                                 ap-%offset%))))
```

```
                (setf temp2
                       (+ temp2
                          (*
                           (f2cl-lib:dconjg
                            (f2cl-lib:fref ap-%data%
                                           (k)
                                           ((1 *))
                                           ap-%offset%))
                           (f2cl-lib:fref x-%data%
                                          (i)
                                          ((1 *))
                                          x-%offset%))))
                (setf k (f2cl-lib:int-add k 1))))
            (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                  (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
                     (* alpha temp2)))
            (setf kk
                  (f2cl-lib:int-add kk
                                    (f2cl-lib:int-add
                                     (f2cl-lib:int-sub n j)
                                     1))))))
          (t
           (setf jx kx)
           (setf jy ky)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf temp1
                     (* alpha
                        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
               (setf temp2 zero)
               (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                     (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                        (* temp1
                           (coerce (realpart
                                    (f2cl-lib:fref ap-%data%
                                                   (kk)
                                                   ((1 *))
                                                   ap-%offset%)) 'double-float))))
               (setf ix jx)
               (setf iy jy)
               (f2cl-lib:fdo (k (f2cl-lib:int-add kk 1)
                                (f2cl-lib:int-add k 1))
                             ((> k
                                 (f2cl-lib:int-add kk
                                                   n
```

```
                                                (f2cl-lib:int-sub j)))
                                 nil)
                           (tagbody
                             (setf ix (f2cl-lib:int-add ix incx))
                             (setf iy (f2cl-lib:int-add iy incy))
                             (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                                     (+
                                      (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                                      (* temp1
                                         (f2cl-lib:fref ap-%data%
                                                        (k)
                                                        ((1 *))
                                                        ap-%offset%))))
                             (setf temp2
                                     (+ temp2
                                        (*
                                         (f2cl-lib:dconjg
                                          (f2cl-lib:fref ap-%data%
                                                         (k)
                                                         ((1 *))
                                                         ap-%offset%))
                                         (f2cl-lib:fref x-%data%
                                                        (ix)
                                                        ((1 *))
                                                        x-%offset%))))))
                           (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                   (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                                      (* alpha temp2)))
                           (setf jx (f2cl-lib:int-add jx incx))
                           (setf jy (f2cl-lib:int-add jy incy))
                           (setf kk
                                   (f2cl-lib:int-add kk
                                                     (f2cl-lib:int-add
                                                      (f2cl-lib:int-sub n j)
                                                      1)))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil nil))))))
```

5.26 zhpr2 BLAS

⟨*zhpr2.input*⟩≡

```
)set break resume
)sys rm -f zhpr2.output
)spool zhpr2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zhpr2.help*⟩≡

```
 ==========================================================================
 zhpr2 examples
 ==========================================================================

 ==========================================================================
 Man Page Details
 ==========================================================================

  NAME
       ZHPR2 - perform the hermitian rank 2 operation   A :=
       alpha*x*conjg( y' ) + conjg( alpha )*y*conjg( x' ) + A,

  SYNOPSIS
       SUBROUTINE ZHPR2 ( UPLO, N, ALPHA, X, INCX, Y, INCY, AP )

           COMPLEX*16   ALPHA

           INTEGER      INCX, INCY, N

           CHARACTER*1  UPLO

           COMPLEX*16   AP( * ), X( * ), Y( * )

  PURPOSE
       ZHPR2  performs the hermitian rank 2 operation

       where alpha is a scalar, x and y are n element vectors and A
       is an n by n hermitian matrix, supplied in packed form.

  PARAMETERS
       UPLO   - CHARACTER*1.
              On entry, UPLO specifies whether the upper or lower
              triangular part of the matrix A is supplied in the
              packed array AP as follows:

              UPLO = 'U' or 'u'   The upper triangular part of A is
              supplied in AP.

              UPLO = 'L' or 'l'   The lower triangular part of A is
              supplied in AP.

              Unchanged on exit.

       N      - INTEGER.
              On entry, N specifies the order of the matrix A.  N
```

must be at least zero. Unchanged on exit.

ALPHA - COMPLEX*16 .
On entry, ALPHA specifies the scalar alpha.
Unchanged on exit.

X - COMPLEX*16 array of dimension at least
(1 + (n - 1)*abs(INCX)). Before entry, the
incremented array X must contain the n element vector
x. Unchanged on exit.

INCX - INTEGER.
On entry, INCX specifies the increment for the ele-
ments of X. INCX must not be zero. Unchanged on

exit.

Y - COMPLEX*16 array of dimension at least
(1 + (n - 1)*abs(INCY)). Before entry, the
incremented array Y must contain the n element vector
y. Unchanged on exit.

INCY - INTEGER.
On entry, INCY specifies the increment for the ele-
ments of Y. INCY must not be zero. Unchanged on
exit.

AP - COMPLEX*16 array of DIMENSION at least
((n*(n + 1))/2). Before entry with UPLO = 'U'
or 'u', the array AP must contain the upper triangu-
lar part of the hermitian matrix packed sequentially,
column by column, so that AP(1) contains a(1, 1),
AP(2) and AP(3) contain a(1, 2) and a(2, 2)
respectively, and so on. On exit, the array AP is
overwritten by the upper triangular part of the
updated matrix. Before entry with UPLO = 'L' or 'l',
the array AP must contain the lower triangular part
of the hermitian matrix packed sequentially, column
by column, so that AP(1) contains a(1, 1), AP(2
) and AP(3) contain a(2, 1) and a(3, 1) respec-
tively, and so on. On exit, the array AP is overwrit-
ten by the lower triangular part of the updated
matrix. Note that the imaginary parts of the diago-
nal elements need not be set, they are assumed to be
zero, and on exit they are set to zero.

```
            Level 2 Blas routine.

            -- Written on 22-October-1986.  Jack Dongarra,
            Argonne National Lab.  Jeremy Du Croz, Nag Central
            Office.  Sven Hammarling, Nag Central Office.
            Richard Hanson, Sandia National Labs.
NAME

SYNOPSIS

rou-
tine
zrotg(ca,cb,c,s)
sub-
ble
    dou-                    complex
                            ca,cb,s
ble
    dou-                    precision
                            c
ble
    dou-                    precision
                            norm,scale
ble
    dou-                    complex
                            alpha
    if                      (cdabs(ca)
                            .ne.
                            0.0d0)
                            go
                            to
                            10
    c                       =
                            0.0d0
    s                       =
                            (1.0d0,0.0d0)
    ca                      =
                            cb
    go                      to
                            20
    10                      con-
                            tinue
    scale                   =
                            cdabs(ca)
                            +
                            cdabs(cb)
```

```
      norm                    =
                              scale*dsqrt((cdabs(ca/dcmplx(scale,0.0d0)))**2
                              +
      *                       (cdabs(cb/dcmplx(scale,0.0d0)))**2)
      alpha                   =
                              ca
                              /cdabs(ca)
      c                       =
                              cdabs(ca)
                              /
                              norm
      s                       =
                              alpha
                              *
                              dconjg(cb)
                              /
                              norm
      ca                      =
                              alpha
                              *
                              norm
      20                      continue
      return
      end
PUR-
POSE
NAME

SYNOPSIS

rou-
tine
zscal(n,za,zx,incx)
sub-
      c                          scales
                                 a
                                 vec-
                                 tor
                                 by
                                 a
                                 con-
                                 stant.
      c                          jack
                                 dongarra,
                                 3/11/78.
      c                          modified
```

```
                              to
                              correct
                              prob-
                              lem
                              with
                              nega-
                              tive
                              incre-
                              ment,
                              8/21/90.
   ble
       dou-                   complex
                              za,zx(1)
       integer                i,incx,ix,n
       if(n.le.0)return
       if(incx.eq.1)go        to
                              20
       c                      code
                              for
                              incre-
                              ment
                              not
                              equal
                              to
                              1
       ix                     =
                              1
       if(incx.lt.0)ix        =
                              (-
                              n+1)*incx
                              +
                              1
       do                     10
                              i
                              =
                              1,n
       zx(ix)                 =
                              za*zx(ix)
       ix                     =
                              ix
                              +
                              incx
       10                     con-
                              tinue
       return
       c                      code
```

```
                                for
                                incre-
                                ment
                                equal
                                to
                                1
      20                        do
                                30
                                i
                                =
                                1,n
      zx(i)                     =
                                za*zx(i)
      30                        con-
                                tinue
      return
      end
  PUR-
  POSE
```

⟨*BLAS 2 zhpr2*⟩≡

```
(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun zhpr2 (uplo n alpha x incx y incy ap)
    (declare (type (simple-array (complex double-float) (*)) ap y x)
             (type (complex double-float) alpha)
             (type fixnum incy incx n)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (x (complex double-float) x-%data% x-%offset%)
         (y (complex double-float) y-%data% y-%offset%)
         (ap (complex double-float) ap-%data% ap-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (k 0) (kk 0)
             (kx 0) (ky 0) (temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)))
        (declare (type fixnum i info ix iy j jx jy k kk kx ky)
                 (type (complex double-float) temp1 temp2))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((= incx 0)
           (setf info 5))
          ((= incy 0)
           (setf info 7)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZHPR2" info)
           (go end_label)))
        (if (or (= n 0) (= alpha zero)) (go end_label))
        (cond
          ((or (/= incx 1) (/= incy 1))
           (cond
             ((> incx 0)
              (setf kx 1))
             (t
              (setf kx
                      (f2cl-lib:int-sub 1
                                        (f2cl-lib:int-mul
                                         (f2cl-lib:int-sub n 1)
                                         incx)))))
           (cond
```

```
          ((> incy 0)
           (setf ky 1))
          (t
           (setf ky
                 (f2cl-lib:int-sub 1
                                   (f2cl-lib:int-mul
                                    (f2cl-lib:int-sub n 1)
                                    incy)))))
         (setf jx kx)
         (setf jy ky)))
      (setf kk 1)
      (cond
        ((char-equal uplo #\U)
         (cond
           ((and (= incx 1) (= incy 1))
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                          ((> j n) nil)
              (tagbody
                (cond
                  ((or (/= (f2cl-lib:fref x (j) ((1 *))) zero)
                       (/= (f2cl-lib:fref y (j) ((1 *))) zero))
                   (setf temp1
                           (* alpha
                              (f2cl-lib:dconjg
                               (f2cl-lib:fref y-%data%
                                              (j)
                                              ((1 *))
                                              y-%offset%))))
                   (setf temp2
                           (coerce
                            (f2cl-lib:dconjg
                             (* alpha
                                (f2cl-lib:fref x-%data%
                                               (j)
                                               ((1 *))
                                               x-%offset%)))
                            '(complex double-float)))
                   (setf k kk)
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i
                                     (f2cl-lib:int-add j
                                                       (f2cl-lib:int-sub 1)))
                                  nil)
                     (tagbody
                       (setf (f2cl-lib:fref ap-%data%
                                            (k)
```

```
                                    ((1 *))
                                    ap-%offset%)
                                (+
                                 (f2cl-lib:fref ap-%data%
                                                (k)
                                                ((1 *))
                                                ap-%offset%)
                                 (*
                                  (f2cl-lib:fref x-%data%
                                                 (i)
                                                 ((1 *))
                                                 x-%offset%)
                                  temp1)
                                 (*
                                  (f2cl-lib:fref y-%data%
                                                 (i)
                                                 ((1 *))
                                                 y-%offset%)
                                  temp2)))
                            (setf k (f2cl-lib:int-add k 1))))
                        (setf (f2cl-lib:fref ap-%data%
                                             ((f2cl-lib:int-sub
                                               (f2cl-lib:int-add kk j)
                                               1))
                                             ((1 *))
                                             ap-%offset%)
                              (coerce
                               (+
                                (coerce (realpart
                                 (f2cl-lib:fref ap-%data%
                                                ((f2cl-lib:int-sub
                                                  (f2cl-lib:int-add kk j)
                                                  1))
                                                ((1 *))
                                                ap-%offset%)) 'double-float)
                                (coerce (realpart
                                 (+
                                  (*
                                   (f2cl-lib:fref x-%data%
                                                  (j)
                                                  ((1 *))
                                                  x-%offset%)
                                   temp1)
                                  (*
                                   (f2cl-lib:fref y-%data%
                                                  (j)
```

```
                                        ((1 *))
                                        y-%offset%)
                              temp2))) 'double-float))
                         '(complex double-float))))
                (t
                 (setf (f2cl-lib:fref ap-%data%
                                      ((f2cl-lib:int-sub
                                        (f2cl-lib:int-add kk j)
                                        1))
                                      ((1 *))
                                      ap-%offset%)
                       (coerce
                        (coerce (realpart
                                 (f2cl-lib:fref ap-%data%
                                                ((f2cl-lib:int-sub
                                                  (f2cl-lib:int-add kk j)
                                                  1))
                                                ((1 *))
                                                ap-%offset%)) 'double-float)
                        '(complex double-float)))))
            (setf kk (f2cl-lib:int-add kk j)))))
        (t
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (cond
               ((or (/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                    (/= (f2cl-lib:fref y (jy) ((1 *))) zero))
                (setf temp1
                        (* alpha
                           (f2cl-lib:dconjg
                            (f2cl-lib:fref y-%data%
                                           (jy)
                                           ((1 *))
                                           y-%offset%))))
                (setf temp2
                        (coerce
                         (f2cl-lib:dconjg
                          (* alpha
                             (f2cl-lib:fref x-%data%
                                            (jx)
                                            ((1 *))
                                            x-%offset%)))
                         '(complex double-float)))
                (setf ix kx)
                (setf iy ky)
```

```
(f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
              ((> k
                  (f2cl-lib:int-add kk
                                    j
                                    (f2cl-lib:int-sub 2)))
               nil)
  (tagbody
    (setf (f2cl-lib:fref ap-%data%
                         (k)
                         ((1 *))
                         ap-%offset%)
          (+
           (f2cl-lib:fref ap-%data%
                          (k)
                          ((1 *))
                          ap-%offset%)
           (*
            (f2cl-lib:fref x-%data%
                           (ix)
                           ((1 *))
                           x-%offset%)
            temp1)
           (*
            (f2cl-lib:fref y-%data%
                           (iy)
                           ((1 *))
                           y-%offset%)
            temp2)))
    (setf ix (f2cl-lib:int-add ix incx))
    (setf iy (f2cl-lib:int-add iy incy))))
(setf (f2cl-lib:fref ap-%data%
                     ((f2cl-lib:int-sub
                       (f2cl-lib:int-add kk j)
                       1))
                     ((1 *))
                     ap-%offset%)
      (coerce
       (+
        (coerce (realpart
         (f2cl-lib:fref ap-%data%
                        ((f2cl-lib:int-sub
                          (f2cl-lib:int-add kk j)
                          1))
                        ((1 *))
                        ap-%offset%)) 'double-float)
        (coerce (realpart
```

```
                              (+
                               (*
                                (f2cl-lib:fref x-%data%
                                               (jx)
                                               ((1 *))
                                               x-%offset%)
                                temp1)
                               (*
                                (f2cl-lib:fref y-%data%
                                               (jy)
                                               ((1 *))
                                               y-%offset%)
                                temp2))) 'double-float))
                            '(complex double-float))))
                     (t
                      (setf (f2cl-lib:fref ap-%data%
                                           ((f2cl-lib:int-sub
                                             (f2cl-lib:int-add kk j)
                                             1))
                                           ((1 *))
                                           ap-%offset%)
                            (coerce
                             (coerce (realpart
                              (f2cl-lib:fref ap-%data%
                                             ((f2cl-lib:int-sub
                                               (f2cl-lib:int-add kk j)
                                               1))
                                             ((1 *))
                                             ap-%offset%)) 'double-float)
                             '(complex double-float)))))
                     (setf jx (f2cl-lib:int-add jx incx))
                     (setf jy (f2cl-lib:int-add jy incy))
                     (setf kk (f2cl-lib:int-add kk j)))))))
           (t
            (cond
              ((and (= incx 1) (= incy 1))
               (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                             ((> j n) nil)
                 (tagbody
                   (cond
                     ((or (/= (f2cl-lib:fref x (j) ((1 *))) zero)
                          (/= (f2cl-lib:fref y (j) ((1 *))) zero))
                      (setf temp1
                            (* alpha
                               (f2cl-lib:dconjg
                                (f2cl-lib:fref y-%data%
```

```
                              (j)
                              ((1 *))
                              y-%offset%))))
(setf temp2
        (coerce
         (f2cl-lib:dconjg
          (* alpha
             (f2cl-lib:fref x-%data%
                            (j)
                            ((1 *))
                            x-%offset%)))
         '(complex double-float)))
(setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
        (coerce
         (+
          (coerce (realpart
           (f2cl-lib:fref ap-%data%
                          (kk)
                          ((1 *))
                          ap-%offset%)) 'double-float)
          (coerce (realpart
           (+
            (*
             (f2cl-lib:fref x-%data%
                            (j)
                            ((1 *))
                            x-%offset%)
             temp1)
            (*
             (f2cl-lib:fref y-%data%
                            (j)
                            ((1 *))
                            y-%offset%)
             temp2))) 'double-float))
         '(complex double-float)))
(setf k (f2cl-lib:int-add kk 1))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                 (f2cl-lib:int-add i 1))
              ((> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref ap-%data%
                         (k)
                         ((1 *))
                         ap-%offset%)
            (+
             (f2cl-lib:fref ap-%data%
```

```
                                        (k)
                                        ((1 *))
                                        ap-%offset%)
                                (*
                                 (f2cl-lib:fref x-%data%
                                                (i)
                                                ((1 *))
                                                x-%offset%)
                                 temp1)
                                (*
                                 (f2cl-lib:fref y-%data%
                                                (i)
                                                ((1 *))
                                                y-%offset%)
                                 temp2)))
                        (setf k (f2cl-lib:int-add k 1)))))
                  (t
                   (setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
                           (coerce
                            (coerce (realpart
                             (f2cl-lib:fref ap-%data%
                                            (kk)
                                            ((1 *))
                                            ap-%offset%)) 'double-float)
                            '(complex double-float)))))
               (setf kk
                       (f2cl-lib:int-add
                        (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
                        1)))))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (cond
                 ((or (/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                      (/= (f2cl-lib:fref y (jy) ((1 *))) zero))
                  (setf temp1
                          (* alpha
                             (f2cl-lib:dconjg
                              (f2cl-lib:fref y-%data%
                                             (jy)
                                             ((1 *))
                                             y-%offset%))))
                  (setf temp2
                          (coerce
                           (f2cl-lib:dconjg
```

```
                (* alpha
                   (f2cl-lib:fref x-%data%
                                  (jx)
                                  ((1 *))
                                  x-%offset%)))
                '(complex double-float)))
          (setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
                (coerce
                 (+
                  (coerce (realpart
                   (f2cl-lib:fref ap-%data%
                                  (kk)
                                  ((1 *))
                                  ap-%offset%)) 'double-float)
                  (coerce (realpart
                   (+
                    (*
                     (f2cl-lib:fref x-%data%
                                    (jx)
                                    ((1 *))
                                    x-%offset%)
                     temp1)
                    (*
                     (f2cl-lib:fref y-%data%
                                    (jy)
                                    ((1 *))
                                    y-%offset%)
                     temp2))) 'double-float))
                '(complex double-float)))
          (setf ix jx)
          (setf iy jy)
          (f2cl-lib:fdo (k (f2cl-lib:int-add kk 1)
                         (f2cl-lib:int-add k 1))
                        ((> k
                            (f2cl-lib:int-add kk
                                              n
                                              (f2cl-lib:int-sub j)))
                         nil)
            (tagbody
              (setf ix (f2cl-lib:int-add ix incx))
              (setf iy (f2cl-lib:int-add iy incy))
              (setf (f2cl-lib:fref ap-%data%
                                   (k)
                                   ((1 *))
                                   ap-%offset%)
                    (+
```

```
                                    (f2cl-lib:fref ap-%data%
                                                   (k)
                                                   ((1 *))
                                                   ap-%offset%)
                                    (*
                                     (f2cl-lib:fref x-%data%
                                                    (ix)
                                                    ((1 *))
                                                    x-%offset%)
                                     temp1)
                                    (*
                                     (f2cl-lib:fref y-%data%
                                                    (iy)
                                                    ((1 *))
                                                    y-%offset%)
                                     temp2))))))
                          (t
                           (setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
                                   (coerce
                                    (coerce (realpart
                                     (f2cl-lib:fref ap-%data%
                                                    (kk)
                                                    ((1 *))
                                                    ap-%offset%)) 'double-float)
                                    '(complex double-float)))))
                        (setf jx (f2cl-lib:int-add jx incx))
                        (setf jy (f2cl-lib:int-add jy incy))
                        (setf kk
                                (f2cl-lib:int-add
                                 (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
                                 1))))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil))))))
```

5.27 zhpr BLAS

⟨*zhpr.input*⟩≡

```
)set break resume
)sys rm -f zhpr.output
)spool zhpr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zhpr.help*⟩≡

```
  ======================================================================
  zhpr examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        ZHPR - perform the hermitian rank 1 operation   A :=
        alpha*x*conjg( x' ) + A,

   SYNOPSIS
        SUBROUTINE ZHPR ( UPLO, N, ALPHA, X, INCX, AP )

            DOUBLE      PRECISION ALPHA

            INTEGER     INCX, N

            CHARACTER*1 UPLO

            COMPLEX*16  AP( * ), X( * )

   PURPOSE
        ZHPR    performs the hermitian rank 1 operation

        where alpha is a real scalar, x is an n element vector and A
        is an n by n hermitian matrix, supplied in packed form.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On entry, UPLO specifies whether the upper or lower
                 triangular part of the matrix A is supplied in the
                 packed array AP as follows:

                 UPLO = 'U' or 'u'   The upper triangular part of A is
                 supplied in AP.

                 UPLO = 'L' or 'l'   The lower triangular part of A is
                 supplied in AP.

                 Unchanged on exit.

        N      - INTEGER.
                 On entry, N specifies the order of the matrix A.  N
```

```
       must be at least zero.  Unchanged on exit.

ALPHA  - DOUBLE PRECISION.
       On entry, ALPHA specifies the scalar alpha.
       Unchanged on exit.

X      - COMPLEX*16       array of dimension at least
       ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
       incremented array X must contain the n element vector
       x.  Unchanged on exit.

INCX   - INTEGER.
       On entry, INCX specifies the increment for the ele-
       ments of X. INCX must not be zero.  Unchanged on

       exit.

AP     - COMPLEX*16       array of DIMENSION at least
       ( ( n*( n + 1 ) )/2 ).  Before entry with  UPLO = 'U'
       or 'u', the array AP must contain the upper triangu-
       lar part of the hermitian matrix packed sequentially,
       column by column, so that AP( 1 ) contains a( 1, 1 ),
       AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 )
       respectively, and so on. On exit, the array AP is
       overwritten by the upper triangular part of the
       updated matrix.  Before entry with UPLO = 'L' or 'l',
       the array AP must contain the lower triangular part
       of the hermitian matrix packed sequentially, column
       by column, so that AP( 1 ) contains a( 1, 1 ), AP( 2
       ) and AP( 3 ) contain a( 2, 1 ) and a( 3, 1 ) respec-
       tively, and so on. On exit, the array AP is overwrit-
       ten by the lower triangular part of the updated
       matrix.  Note that the imaginary parts of the diago-
       nal elements need not be set, they are assumed to be
       zero, and on exit they are set to zero.
```

⟨*BLAS 2 zhpr*⟩≡

```
(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun zhpr (uplo n alpha x incx ap)
    (declare (type (simple-array (complex double-float) (*)) ap x)
             (type (double-float) alpha)
             (type fixnum incx n)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (x (complex double-float) x-%data% x-%offset%)
         (ap (complex double-float) ap-%data% ap-%offset%))
      (prog ((i 0) (info 0) (ix 0) (j 0) (jx 0) (k 0) (kk 0) (kx 0)
             (temp #C(0.0 0.0)))
        (declare (type fixnum i info ix j jx k kk kx)
                 (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((< n 0)
           (setf info 2))
          ((= incx 0)
           (setf info 5)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZHPR" info)
           (go end_label)))
        (if (or (= n 0) (= alpha (coerce (realpart zero) 'double-float)))
           (go end_label))
        (cond
          ((<= incx 0)
           (setf kx
                   (f2cl-lib:int-sub 1
                                     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                       incx))))
          ((/= incx 1)
           (setf kx 1)))
        (setf kk 1)
        (cond
          ((char-equal uplo #\U)
           (cond
             ((= incx 1)
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
```

```
            ((> j n) nil)
          (tagbody
            (cond
              ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
               (setf temp
                       (coerce
                        (* alpha
                           (f2cl-lib:dconjg
                            (f2cl-lib:fref x-%data%
                                           (j)
                                           ((1 *))
                                           x-%offset%)))
                        '(complex double-float)))
               (setf k kk)
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i
                                 (f2cl-lib:int-add j
                                                   (f2cl-lib:int-sub 1)))
                              nil)
                 (tagbody
                   (setf (f2cl-lib:fref ap-%data%
                                        (k)
                                        ((1 *))
                                        ap-%offset%)
                           (+
                            (f2cl-lib:fref ap-%data%
                                           (k)
                                           ((1 *))
                                           ap-%offset%)
                            (*
                             (f2cl-lib:fref x-%data%
                                            (i)
                                            ((1 *))
                                            x-%offset%)
                             temp)))
                   (setf k (f2cl-lib:int-add k 1))))
               (setf (f2cl-lib:fref ap-%data%
                                    ((f2cl-lib:int-sub
                                      (f2cl-lib:int-add kk j)
                                      1))
                                    ((1 *))
                                    ap-%offset%)
                       (coerce
                        (+
                         (coerce (realpart
                          (f2cl-lib:fref ap-%data%
```

```
                                        ((f2cl-lib:int-sub
                                          (f2cl-lib:int-add kk j)
                                          1))
                                        ((1 *))
                                        ap-%offset%)) 'double-float)
                          (coerce (realpart
                           (*
                            (f2cl-lib:fref x-%data%
                                           (j)
                                           ((1 *))
                                           x-%offset%)
                            temp)) 'double-float))
                         '(complex double-float))))
                (t
                 (setf (f2cl-lib:fref ap-%data%
                                      ((f2cl-lib:int-sub
                                        (f2cl-lib:int-add kk j)
                                        1))
                                      ((1 *))
                                      ap-%offset%)
                       (coerce
                        (coerce (realpart
                         (f2cl-lib:fref ap-%data%
                                        ((f2cl-lib:int-sub
                                          (f2cl-lib:int-add kk j)
                                          1))
                                        ((1 *))
                                        ap-%offset%)) 'double-float)
                        '(complex double-float)))))
            (setf kk (f2cl-lib:int-add kk j)))))
        (t
         (setf jx kx)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (cond
               ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                (setf temp
                        (coerce
                         (* alpha
                            (f2cl-lib:dconjg
                             (f2cl-lib:fref x-%data%
                                            (jx)
                                            ((1 *))
                                            x-%offset%)))
                         '(complex double-float)))
```

```
(setf ix kx)
(f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
              ((> k
                  (f2cl-lib:int-add kk
                                    j
                                    (f2cl-lib:int-sub 2)))
               nil)
  (tagbody
    (setf (f2cl-lib:fref ap-%data%
                         (k)
                         ((1 *))
                         ap-%offset%)
          (+
           (f2cl-lib:fref ap-%data%
                          (k)
                          ((1 *))
                          ap-%offset%)
           (*
            (f2cl-lib:fref x-%data%
                           (ix)
                           ((1 *))
                           x-%offset%)
            temp)))
    (setf ix (f2cl-lib:int-add ix incx))))
(setf (f2cl-lib:fref ap-%data%
                     ((f2cl-lib:int-sub
                       (f2cl-lib:int-add kk j)
                       1))
                     ((1 *))
                     ap-%offset%)
      (coerce
       (+
        (coerce (realpart
         (f2cl-lib:fref ap-%data%
                        ((f2cl-lib:int-sub
                          (f2cl-lib:int-add kk j)
                          1))
                        ((1 *))
                        ap-%offset%)) 'double-float)
        (coerce (realpart
         (*
          (f2cl-lib:fref x-%data%
                         (jx)
                         ((1 *))
                         x-%offset%)
          temp)) 'double-float))
```

```
                        '(complex double-float))))
                (t
                 (setf (f2cl-lib:fref ap-%data%
                                      ((f2cl-lib:int-sub
                                        (f2cl-lib:int-add kk j)
                                        1))
                                      ((1 *))
                                      ap-%offset%)
                       (coerce
                        (coerce (realpart
                         (f2cl-lib:fref ap-%data%
                                        ((f2cl-lib:int-sub
                                          (f2cl-lib:int-add kk j)
                                          1))
                                        ((1 *))
                                        ap-%offset%)) 'double-float)
                        '(complex double-float)))))
          (setf jx (f2cl-lib:int-add jx incx))
          (setf kk (f2cl-lib:int-add kk j)))))))
(t
 (cond
   ((= incx 1)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  ((> j n) nil)
      (tagbody
        (cond
          ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
           (setf temp
                   (coerce
                    (* alpha
                       (f2cl-lib:dconjg
                        (f2cl-lib:fref x-%data%
                                       (j)
                                       ((1 *))
                                       x-%offset%)))
                    '(complex double-float)))
           (setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
                   (coerce
                    (+
                     (coerce (realpart
                      (f2cl-lib:fref ap-%data%
                                     (kk)
                                     ((1 *))
                                     ap-%offset%)) 'double-float)
                     (coerce (realpart
                      (* temp
```

```
                                (f2cl-lib:fref x-%data%
                                               (j)
                                               ((1 *))
                                               x-%offset%))) 'double-float))
                         '(complex double-float)))
                  (setf k (f2cl-lib:int-add kk 1))
                  (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                   (f2cl-lib:int-add i 1))
                                ((> i n) nil)
                    (tagbody
                      (setf (f2cl-lib:fref ap-%data%
                                           (k)
                                           ((1 *))
                                           ap-%offset%)
                            (+
                             (f2cl-lib:fref ap-%data%
                                            (k)
                                            ((1 *))
                                            ap-%offset%)
                             (*
                              (f2cl-lib:fref x-%data%
                                             (i)
                                             ((1 *))
                                             x-%offset%)
                              temp)))
                      (setf k (f2cl-lib:int-add k 1)))))
                 (t
                  (setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
                        (coerce
                         (coerce (realpart
                                  (f2cl-lib:fref ap-%data%
                                                 (kk)
                                                 ((1 *))
                                                 ap-%offset%)) 'double-float)
                         '(complex double-float)))))
               (setf kk
                     (f2cl-lib:int-add
                      (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
                      1)))))
          (t
           (setf jx kx)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (cond
                 ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
```

```
(setf temp
        (coerce
         (* alpha
            (f2cl-lib:dconjg
             (f2cl-lib:fref x-%data%
                            (jx)
                            ((1 *))
                            x-%offset%)))
         '(complex double-float)))
(setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
        (coerce
         (+
          (coerce (realpart
           (f2cl-lib:fref ap-%data%
                          (kk)
                          ((1 *))
                          ap-%offset%)) 'double-float)
          (coerce (realpart
           (* temp
              (f2cl-lib:fref x-%data%
                             (jx)
                             ((1 *))
                             x-%offset%))) 'double-float))
         '(complex double-float)))
(setf ix jx)
(f2cl-lib:fdo (k (f2cl-lib:int-add kk 1)
                 (f2cl-lib:int-add k 1))
              ((> k
                  (f2cl-lib:int-add kk
                                    n
                                    (f2cl-lib:int-sub j)))
               nil)
  (tagbody
    (setf ix (f2cl-lib:int-add ix incx))
    (setf (f2cl-lib:fref ap-%data%
                         (k)
                         ((1 *))
                         ap-%offset%)
          (+
           (f2cl-lib:fref ap-%data%
                          (k)
                          ((1 *))
                          ap-%offset%)
           (*
            (f2cl-lib:fref x-%data%
                           (ix)
```

```
                                                ((1 *))
                                                x-%offset%)
                                      temp))))))
                        (t
                         (setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
                                 (coerce
                                  (coerce (realpart
                                   (f2cl-lib:fref ap-%data%
                                                  (kk)
                                                  ((1 *))
                                                  ap-%offset%)) 'double-float)
                                  '(complex double-float)))))
                       (setf jx (f2cl-lib:int-add jx incx))
                       (setf kk
                               (f2cl-lib:int-add
                                (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
                                1))))))))
   end_label
          (return (values nil nil nil nil nil nil))))))
```

5.28 ztbmv BLAS

⟨*ztbmv.input*⟩≡

```
  )set break resume
  )sys rm -f ztbmv.output
  )spool ztbmv.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*ztbmv.help*⟩≡

```
  ======================================================================
  ztbmv examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        ZTBMV - perform one of the matrix-vector operations   x :=
        A*x, or x := A'*x, or x := conjg( A' )*x,

   SYNOPSIS
        SUBROUTINE ZTBMV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX
                         )

            INTEGER      INCX, K, LDA, N

            CHARACTER*1  DIAG, TRANS, UPLO

            COMPLEX*16   A( LDA, * ), X( * )

   PURPOSE
        ZTBMV  performs one of the matrix-vector operations

        where x is an n element vector and  A is an n by n unit, or
        non-unit, upper or lower triangular band matrix, with ( k +
        1 ) diagonals.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On entry, UPLO specifies whether the matrix is an
                 upper or lower triangular matrix as follows:

                 UPLO = 'U' or 'u'   A is an upper triangular matrix.

                 UPLO = 'L' or 'l'   A is a lower triangular matrix.

                 Unchanged on exit.

        TRANS  - CHARACTER*1.
                 On entry, TRANS specifies the operation to be per-
                 formed as follows:

                 TRANS = 'N' or 'n'   x := A*x.
```

```
        TRANS = 'T' or 't'   x := A'*x.

        TRANS = 'C' or 'c'   x := conjg( A' )*x.

        Unchanged on exit.

DIAG    - CHARACTER*1.
        On entry, DIAG specifies whether or not A is unit
        triangular as follows:

        DIAG = 'U' or 'u'   A is assumed to be unit triangu-
        lar.

        DIAG = 'N' or 'n'   A is not assumed to be unit tri-
        angular.

        Unchanged on exit.

N       - INTEGER.
        On entry, N specifies the order of the matrix A.  N
        must be at least zero.  Unchanged on exit.

K       - INTEGER.
        On entry with UPLO = 'U' or 'u', K specifies the
        number of super-diagonals of the matrix A.  On entry
        with UPLO = 'L' or 'l', K specifies the number of
        sub-diagonals of the matrix A.  K must satisfy  0
        .le. K.  Unchanged on exit.

A       - COMPLEX*16       array of DIMENSION ( LDA, n ).
        Before entry with UPLO = 'U' or 'u', the leading ( k
        + 1 ) by n part of the array A must contain the upper
        triangular band part of the matrix of coefficients,
        supplied column by column, with the leading diagonal
        of the matrix in row ( k + 1 ) of the array, the
        first super-diagonal starting at position 2 in row k,
        and so on. The top left k by k triangle of the array
        A is not referenced.  The following program segment
        will transfer an upper triangular band matrix from
        conventional full matrix storage to band storage:

        DO 20, J = 1, N M = K + 1 - J DO 10, I = MAX( 1, J -
        K ), J A( M + I, J ) = matrix( I, J ) 10    CONTINUE
        20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

DO 20, J = 1, N M = 1 - J DO 10, I = J, MIN(N, J + K) A(M + I, J) = matrix(I, J) 10 CONTINUE 20 CONTINUE

Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity. Unchanged on exit.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1). Unchanged on exit.

X - COMPLEX*16 array of dimension at least (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

⟨*BLAS 2 ztbmv*⟩≡

```
(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun ztbmv (uplo trans diag n k a lda x incx)
    (declare (type (simple-array (complex double-float) (*)) x a)
             (type fixnum incx lda k n)
             (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (diag character diag-%data% diag-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (x (complex double-float) x-%data% x-%offset%))
      (prog ((noconj nil) (nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0)
             (kplus1 0) (kx 0) (l 0) (temp #C(0.0 0.0)))
        (declare (type (member t nil) noconj nounit)
                 (type fixnum i info ix j jx kplus1 kx l)
                 (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 2))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((< k 0)
           (setf info 5))
          ((< lda (f2cl-lib:int-add k 1))
           (setf info 7))
          ((= incx 0)
           (setf info 9)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZTBMV" info)
           (go end_label)))
        (if (= n 0) (go end_label))
        (setf noconj (char-equal trans #\T))
        (setf nounit (char-equal diag #\N))
        (cond
```

```
          ((<= incx 0)
           (setf kx
                 (f2cl-lib:int-sub 1
                                   (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                     incx))))
          ((/= incx 1)
           (setf kx 1)))
        (cond
          ((char-equal trans #\N)
           (cond
             ((char-equal uplo #\U)
              (setf kplus1 (f2cl-lib:int-add k 1))
              (cond
                ((= incx 1)
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (cond
                       ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                        (setf temp
                                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                        (setf l (f2cl-lib:int-sub kplus1 j))
                        (f2cl-lib:fdo (i
                                       (max (the fixnum 1)
                                            (the fixnum
                                                 (f2cl-lib:int-add j
                                                                   (f2cl-lib:int-sub
                                                                    k))))
                                       (f2cl-lib:int-add i 1))
                                      ((> i
                                          (f2cl-lib:int-add j
                                                            (f2cl-lib:int-sub
                                                             1)))
                                       nil)
                          (tagbody
                            (setf (f2cl-lib:fref x-%data%
                                                 (i)
                                                 ((1 *))
                                                 x-%offset%)
                                  (+
                                   (f2cl-lib:fref x-%data%
                                                  (i)
                                                  ((1 *))
                                                  x-%offset%)
                                   (* temp
                                      (f2cl-lib:fref a-%data%
```

```
                                          ((f2cl-lib:int-add l i)
                                           j)
                                          ((1 lda) (1 *))
                                          a-%offset%))))))
                (if nounit
                    (setf (f2cl-lib:fref x-%data%
                                         (j)
                                         ((1 *))
                                         x-%offset%)
                          (*
                           (f2cl-lib:fref x-%data%
                                          (j)
                                          ((1 *))
                                          x-%offset%)
                           (f2cl-lib:fref a-%data%
                                          (kplus1 j)
                                          ((1 lda) (1 *))
                                          a-%offset%)))))))))
          (t
           (setf jx kx)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (cond
                 ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                  (setf temp
                          (f2cl-lib:fref x-%data%
                                         (jx)
                                         ((1 *))
                                         x-%offset%))
                  (setf ix kx)
                  (setf l (f2cl-lib:int-sub kplus1 j))
                  (f2cl-lib:fdo (i
                                 (max (the fixnum 1)
                                      (the fixnum
                                           (f2cl-lib:int-add j
                                                             (f2cl-lib:int-sub
                                                              k))))
                                 (f2cl-lib:int-add i 1))
                                ((> i
                                    (f2cl-lib:int-add j
                                                      (f2cl-lib:int-sub
                                                       1)))
                                 nil)
                    (tagbody
                      (setf (f2cl-lib:fref x-%data%
```

```
                                        (ix)
                                        ((1 *))
                                        x-%offset%)
                                (+
                                 (f2cl-lib:fref x-%data%
                                                (ix)
                                                ((1 *))
                                                x-%offset%)
                                 (* temp
                                    (f2cl-lib:fref a-%data%
                                                   ((f2cl-lib:int-add l i)
                                                    j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%))))
                            (setf ix (f2cl-lib:int-add ix incx))))
                        (if nounit
                            (setf (f2cl-lib:fref x-%data%
                                                 (jx)
                                                 ((1 *))
                                                 x-%offset%)
                                  (+
                                   (f2cl-lib:fref x-%data%
                                                  (jx)
                                                  ((1 *))
                                                  x-%offset%)
                                   (f2cl-lib:fref a-%data%
                                                  (kplus1 j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))))))
                        (setf jx (f2cl-lib:int-add jx incx))
                        (if (> j k) (setf kx (f2cl-lib:int-add kx incx))))))))
                (t
                 (cond
                   ((= incx 1)
                    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                  ((> j 1) nil)
                      (tagbody
                        (cond
                          ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                           (setf temp
                                   (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                           (setf l (f2cl-lib:int-sub 1 j))
                           (f2cl-lib:fdo (i
                                          (min (the fixnum n)
                                               (the fixnum
                                                    (f2cl-lib:int-add j k)))
```

```
                                  (f2cl-lib:int-add i
                                                   (f2cl-lib:int-sub 1)))
                                 ((> i (f2cl-lib:int-add j 1)) nil)
                           (tagbody
                             (setf (f2cl-lib:fref x-%data%
                                                  (i)
                                                  ((1 *))
                                                  x-%offset%)
                                   (+
                                    (f2cl-lib:fref x-%data%
                                                   (i)
                                                   ((1 *))
                                                   x-%offset%)
                                    (* temp
                                       (f2cl-lib:fref a-%data%
                                                      ((f2cl-lib:int-add l i)
                                                       j)
                                                      ((1 lda) (1 *))
                                                      a-%offset%))))))
                       (if nounit
                           (setf (f2cl-lib:fref x-%data%
                                                (j)
                                                ((1 *))
                                                x-%offset%)
                                 (*
                                  (f2cl-lib:fref x-%data%
                                                 (j)
                                                 ((1 *))
                                                 x-%offset%)
                                  (f2cl-lib:fref a-%data%
                                                 (1 j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%)))))))))
              (t
               (setf kx
                     (f2cl-lib:int-add kx
                                       (f2cl-lib:int-mul
                                        (f2cl-lib:int-sub n 1)
                                        incx)))
               (setf jx kx)
               (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                             ((> j 1) nil)
                 (tagbody
                   (cond
                     ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                      (setf temp
```

```
                (f2cl-lib:fref x-%data%
                               (jx)
                               ((1 *))
                               x-%offset%))
          (setf ix kx)
          (setf l (f2cl-lib:int-sub 1 j))
          (f2cl-lib:fdo (i
                         (min (the fixnum n)
                              (the fixnum
                                   (f2cl-lib:int-add j k)))
                         (f2cl-lib:int-add i
                                           (f2cl-lib:int-sub 1)))
                        ((> i (f2cl-lib:int-add j 1)) nil)
            (tagbody
              (setf (f2cl-lib:fref x-%data%
                                   (ix)
                                   ((1 *))
                                   x-%offset%)
                    (+
                     (f2cl-lib:fref x-%data%
                                    (ix)
                                    ((1 *))
                                    x-%offset%)
                     (* temp
                        (f2cl-lib:fref a-%data%
                                       ((f2cl-lib:int-add l i)
                                        j)
                                       ((1 lda) (1 *))
                                       a-%offset%))))
              (setf ix (f2cl-lib:int-sub ix incx))))
          (if nounit
              (setf (f2cl-lib:fref x-%data%
                                   (jx)
                                   ((1 *))
                                   x-%offset%)
                    (*
                     (f2cl-lib:fref x-%data%
                                    (jx)
                                    ((1 *))
                                    x-%offset%)
                     (f2cl-lib:fref a-%data%
                                    (1 j)
                                    ((1 lda) (1 *))
                                    a-%offset%))))))
    (setf jx (f2cl-lib:int-sub jx incx))
    (if (>= (f2cl-lib:int-sub n j) k)
```

```
                (setf kx (f2cl-lib:int-sub kx incx))))))))))
           (t
            (cond
              ((char-equal uplo #\U)
               (setf kplus1 (f2cl-lib:int-add k 1))
               (cond
                 ((= incx 1)
                  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                ((> j 1) nil)
                    (tagbody
                      (setf temp
                              (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                      (setf l (f2cl-lib:int-sub kplus1 j))
                      (cond
                        (noconj
                         (if nounit
                             (setf temp
                                     (* temp
                                        (f2cl-lib:fref a-%data%
                                                       (kplus1 j)
                                                       ((1 lda) (1 *))
                                                       a-%offset%))))
                         (f2cl-lib:fdo (i
                                        (f2cl-lib:int-add j
                                                          (f2cl-lib:int-sub 1))
                                        (f2cl-lib:int-add i
                                                          (f2cl-lib:int-sub 1)))
                                       ((> i
                                           (max (the fixnum 1)
                                                (the fixnum
                                                     (f2cl-lib:int-add j
                                                                       (f2cl-lib:int-s
                                                                        k)))))
                                        nil)
                           (tagbody
                             (setf temp
                                     (+ temp
                                        (*
                                         (f2cl-lib:fref a-%data%
                                                        ((f2cl-lib:int-add l i)
                                                         j)
                                                        ((1 lda) (1 *))
                                                        a-%offset%)
                                         (f2cl-lib:fref x-%data%
                                                        (i)
                                                        ((1 *))
```

```
                                                x-%offset%)))))))
                        (t
                         (if nounit
                             (setf temp
                                     (* temp
                                        (f2cl-lib:dconjg
                                         (f2cl-lib:fref a-%data%
                                                        (kplus1 j)
                                                        ((1 lda) (1 *))
                                                        a-%offset%)))))
                         (f2cl-lib:fdo (i
                                        (f2cl-lib:int-add j
                                                          (f2cl-lib:int-sub 1))
                                        (f2cl-lib:int-add i
                                                          (f2cl-lib:int-sub 1)))
                                       ((> i
                                           (max (the fixnum 1)
                                                (the fixnum
                                                     (f2cl-lib:int-add j
                                                                       (f2cl-lib:int-sub
                                                                        k)))))
                                        nil)
                           (tagbody
                             (setf temp
                                     (+ temp
                                        (*
                                         (f2cl-lib:dconjg
                                          (f2cl-lib:fref a-%data%
                                                         ((f2cl-lib:int-add l i)
                                                          j)
                                                         ((1 lda) (1 *))
                                                         a-%offset%))
                                         (f2cl-lib:fref x-%data%
                                                        (i)
                                                        ((1 *))
                                                        x-%offset%))))))))
                     (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                             temp))))
                    (t
                     (setf kx
                             (f2cl-lib:int-add kx
                                               (f2cl-lib:int-mul
                                                (f2cl-lib:int-sub n 1)
                                                incx)))
                     (setf jx kx)
                     (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
```

```
           ((> j 1) nil)
(tagbody
  (setf temp
         (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
  (setf kx (f2cl-lib:int-sub kx incx))
  (setf ix kx)
  (setf l (f2cl-lib:int-sub kplus1 j))
  (cond
    (noconj
     (if nounit
         (setf temp
                 (* temp
                    (f2cl-lib:fref a-%data%
                                   (kplus1 j)
                                   ((1 lda) (1 *))
                                   a-%offset%))))
     (f2cl-lib:fdo (i
                    (f2cl-lib:int-add j
                                      (f2cl-lib:int-sub 1))
                    (f2cl-lib:int-add i
                                      (f2cl-lib:int-sub 1)))
                   ((> i
                       (max (the fixnum 1)
                            (the fixnum
                                 (f2cl-lib:int-add j
                                                  (f2cl-lib:int-s
                                                   k)))))
                    nil)
       (tagbody
         (setf temp
                 (+ temp
                    (*
                     (f2cl-lib:fref a-%data%
                                    ((f2cl-lib:int-add l i)
                                     j)
                                    ((1 lda) (1 *))
                                    a-%offset%)
                     (f2cl-lib:fref x-%data%
                                    (ix)
                                    ((1 *))
                                    x-%offset%))))
         (setf ix (f2cl-lib:int-sub ix incx)))))
    (t
     (if nounit
         (setf temp
                 (* temp
```

```
                                    (f2cl-lib:dconjg
                                     (f2cl-lib:fref a-%data%
                                                    (kplus1 j)
                                                    ((1 lda) (1 *))
                                                    a-%offset%)))))
                          (f2cl-lib:fdo (i
                                         (f2cl-lib:int-add j
                                                           (f2cl-lib:int-sub 1))
                                         (f2cl-lib:int-add i
                                                           (f2cl-lib:int-sub 1)))
                                        ((> i
                                            (max (the fixnum 1)
                                                 (the fixnum
                                                      (f2cl-lib:int-add j
                                                                        (f2cl-lib:int-sub
                                                                         k)))))
                                         nil)
                            (tagbody
                              (setf temp
                                      (+ temp
                                         (*
                                          (f2cl-lib:dconjg
                                           (f2cl-lib:fref a-%data%
                                                          ((f2cl-lib:int-add l i)
                                                           j)
                                                          ((1 lda) (1 *))
                                                          a-%offset%))
                                          (f2cl-lib:fref x-%data%
                                                         (ix)
                                                         ((1 *))
                                                         x-%offset%))))
                                    (setf ix (f2cl-lib:int-sub ix incx))))))
                        (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                                temp)
                        (setf jx (f2cl-lib:int-sub jx incx)))))))
                (t
                 (cond
                   ((= incx 1)
                    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                  ((> j n) nil)
                      (tagbody
                        (setf temp
                                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                        (setf l (f2cl-lib:int-sub 1 j))
                        (cond
                          (noconj
```

```
 (if nounit
     (setf temp
             (* temp
                (f2cl-lib:fref a-%data%
                               (1 j)
                               ((1 lda) (1 *))
                               a-%offset%))))
 (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                  (f2cl-lib:int-add i 1))
               ((> i
                   (min (the fixnum n)
                        (the fixnum
                             (f2cl-lib:int-add j k))))
                nil)
   (tagbody
     (setf temp
             (+ temp
                (*
                 (f2cl-lib:fref a-%data%
                                ((f2cl-lib:int-add l i)
                                 j)
                                ((1 lda) (1 *))
                                a-%offset%)
                 (f2cl-lib:fref x-%data%
                                (i)
                                ((1 *))
                                x-%offset%)))))))
(t
 (if nounit
     (setf temp
             (* temp
                (f2cl-lib:dconjg
                 (f2cl-lib:fref a-%data%
                                (1 j)
                                ((1 lda) (1 *))
                                a-%offset%)))))
 (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                  (f2cl-lib:int-add i 1))
               ((> i
                   (min (the fixnum n)
                        (the fixnum
                             (f2cl-lib:int-add j k))))
                nil)
   (tagbody
     (setf temp
             (+ temp
```

```
                                (*
                                 (f2cl-lib:dconjg
                                  (f2cl-lib:fref a-%data%
                                                 ((f2cl-lib:int-add l i)
                                                  j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))
                                 (f2cl-lib:fref x-%data%
                                                (i)
                                                ((1 *))
                                                x-%offset%))))))))
            (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                  temp))))
          (t
           (setf jx kx)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf temp
                     (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
               (setf kx (f2cl-lib:int-add kx incx))
               (setf ix kx)
               (setf l (f2cl-lib:int-sub 1 j))
               (cond
                 (noconj
                  (if nounit
                      (setf temp
                            (* temp
                               (f2cl-lib:fref a-%data%
                                              (1 j)
                                              ((1 lda) (1 *))
                                              a-%offset%))))
                  (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                   (f2cl-lib:int-add i 1))
                                ((> i
                                    (min (the fixnum n)
                                         (the fixnum
                                              (f2cl-lib:int-add j k))))
                                 nil)
                    (tagbody
                      (setf temp
                            (+ temp
                               (*
                                (f2cl-lib:fref a-%data%
                                               ((f2cl-lib:int-add l i)
                                                j)
```

```
                                                ((1 lda) (1 *))
                                                a-%offset%)
                                           (f2cl-lib:fref x-%data%
                                                (ix)
                                                ((1 *))
                                                x-%offset%))))
                                    (setf ix (f2cl-lib:int-add ix incx)))))
                                (t
                                 (if nounit
                                     (setf temp
                                             (* temp
                                                (f2cl-lib:dconjg
                                                 (f2cl-lib:fref a-%data%
                                                      (1 j)
                                                      ((1 lda) (1 *))
                                                      a-%offset%)))))
                                 (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                                (f2cl-lib:int-add i 1))
                                               ((> i
                                                   (min (the fixnum n)
                                                        (the fixnum
                                                             (f2cl-lib:int-add j k))))
                                                nil)
                                   (tagbody
                                     (setf temp
                                             (+ temp
                                                (*
                                                 (f2cl-lib:dconjg
                                                  (f2cl-lib:fref a-%data%
                                                       ((f2cl-lib:int-add l i)
                                                        j)
                                                       ((1 lda) (1 *))
                                                       a-%offset%))
                                                 (f2cl-lib:fref x-%data%
                                                      (ix)
                                                      ((1 *))
                                                      x-%offset%))))
                                     (setf ix (f2cl-lib:int-add ix incx))))))
                               (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                                     temp)
                               (setf jx (f2cl-lib:int-add jx incx)))))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil nil))))))
```

5.29 ztbsv BLAS

⟨*ztbsv.input*⟩≡

```
)set break resume
)sys rm -f ztbsv.output
)spool ztbsv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*ztbsv.help*⟩≡

```
  ======================================================================
  ztbsv examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        ZTBSV - solve one of the systems of equations   A*x = b, or
        A'*x = b, or conjg( A' )*x = b,

   SYNOPSIS
        SUBROUTINE ZTBSV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX
                         )

            INTEGER      INCX, K, LDA, N

            CHARACTER*1  DIAG, TRANS, UPLO

            COMPLEX*16   A( LDA, * ), X( * )

   PURPOSE
        ZTBSV  solves one of the systems of equations

        where b and x are n element vectors and A is an n by n unit,
        or non-unit, upper or lower triangular band matrix, with ( k
        + 1 ) diagonals.

        No test for singularity or near-singularity is included in
        this routine. Such tests must be performed before calling
        this routine.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On entry, UPLO specifies whether the matrix is an
                 upper or lower triangular matrix as follows:

                 UPLO = 'U' or 'u'   A is an upper triangular matrix.

                 UPLO = 'L' or 'l'   A is a lower triangular matrix.

                 Unchanged on exit.

        TRANS  - CHARACTER*1.
```

```
          On entry, TRANS specifies the equations to be solved
          as follows:

          TRANS = 'N' or 'n'   A*x = b.

          TRANS = 'T' or 't'   A'*x = b.

          TRANS = 'C' or 'c'   conjg( A' )*x = b.

          Unchanged on exit.

DIAG      - CHARACTER*1.
          On entry, DIAG specifies whether or not A is unit
          triangular as follows:

          DIAG = 'U' or 'u'   A is assumed to be unit triangu-
          lar.

          DIAG = 'N' or 'n'   A is not assumed to be unit tri-
          angular.

          Unchanged on exit.

N         - INTEGER.
          On entry, N specifies the order of the matrix A.  N
          must be at least zero.  Unchanged on exit.

K         - INTEGER.
          On entry with UPLO = 'U' or 'u', K specifies the
          number of super-diagonals of the matrix A.  On entry
          with UPLO = 'L' or 'l', K specifies the number of
          sub-diagonals of the matrix A.  K must satisfy  0
          .le. K.  Unchanged on exit.

A         - COMPLEX*16       array of DIMENSION ( LDA, n ).
          Before entry with UPLO = 'U' or 'u', the leading ( k
          + 1 ) by n part of the array A must contain the upper
          triangular band part of the matrix of coefficients,
          supplied column by column, with the leading diagonal
          of the matrix in row ( k + 1 ) of the array, the
          first super-diagonal starting at position 2 in row k,
          and so on. The top left k by k triangle of the array
          A is not referenced.  The following program segment
          will transfer an upper triangular band matrix from
          conventional full matrix storage to band storage:
```

DO 20, J = 1, N M = K + 1 - J DO 10, I = MAX(1, J - K), J A(M + I, J) = matrix(I, J) 10 CONTINUE 20 CONTINUE

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

DO 20, J = 1, N M = 1 - J DO 10, I = J, MIN(N, J + K) A(M + I, J) = matrix(I, J) 10 CONTINUE 20 CONTINUE

Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the

matrix are not referenced, but are assumed to be unity. Unchanged on exit.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1). Unchanged on exit.

X - COMPLEX*16 array of dimension at least (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

⟨*BLAS 2 ztbsv*⟩≡

```
(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun ztbsv (uplo trans diag n k a lda x incx)
    (declare (type (simple-array (complex double-float) (*)) x a)
             (type fixnum incx lda k n)
             (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (diag character diag-%data% diag-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (x (complex double-float) x-%data% x-%offset%))
      (prog ((noconj nil) (nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0)
             (kplus1 0) (kx 0) (l 0) (temp #C(0.0 0.0)))
        (declare (type (member t nil) noconj nounit)
                 (type fixnum i info ix j jx kplus1 kx l)
                 (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 2))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((< k 0)
           (setf info 5))
          ((< lda (f2cl-lib:int-add k 1))
           (setf info 7))
          ((= incx 0)
           (setf info 9)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZTBSV" info)
           (go end_label)))
        (if (= n 0) (go end_label))
        (setf noconj (char-equal trans #\T))
        (setf nounit (char-equal diag #\N))
        (cond
```

```
  ((<= incx 0)
   (setf kx
           (f2cl-lib:int-sub 1
                             (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                               incx))))
  ((/= incx 1)
   (setf kx 1)))
(cond
  ((char-equal trans #\N)
   (cond
     ((char-equal uplo #\U)
      (setf kplus1 (f2cl-lib:int-add k 1))
      (cond
        ((= incx 1)
         (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                       ((> j 1) nil)
           (tagbody
             (cond
               ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                (setf l (f2cl-lib:int-sub kplus1 j))
                (if nounit
                    (setf (f2cl-lib:fref x-%data%
                                         (j)
                                         ((1 *))
                                         x-%offset%)
                                  (/
                                   (f2cl-lib:fref x-%data%
                                                  (j)
                                                  ((1 *))
                                                  x-%offset%)
                                   (f2cl-lib:fref a-%data%
                                                  (kplus1 j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))))
                (setf temp
                        (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                (f2cl-lib:fdo (i
                               (f2cl-lib:int-add j
                                                 (f2cl-lib:int-sub 1))
                               (f2cl-lib:int-add i
                                                 (f2cl-lib:int-sub 1)))
                              ((> i
                                  (max (the fixnum 1)
                                       (the fixnum
                                            (f2cl-lib:int-add j
                                                              (f2cl-lib:int-s
```

```
                                                                  k)))))
                                 nil)
                           (tagbody
                             (setf (f2cl-lib:fref x-%data%
                                                  (i)
                                                  ((1 *))
                                                  x-%offset%)
                                   (-
                                    (f2cl-lib:fref x-%data%
                                                   (i)
                                                   ((1 *))
                                                   x-%offset%)
                                    (* temp
                                       (f2cl-lib:fref a-%data%
                                                      ((f2cl-lib:int-add l i)
                                                       j)
                                                      ((1 lda) (1 *))
                                                      a-%offset%)))))))))
                (t
                 (setf kx
                       (f2cl-lib:int-add kx
                                         (f2cl-lib:int-mul
                                          (f2cl-lib:int-sub n 1)
                                          incx)))
                 (setf jx kx)
                 (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                               ((> j 1) nil)
                   (tagbody
                     (setf kx (f2cl-lib:int-sub kx incx))
                     (cond
                       ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                        (setf ix kx)
                        (setf l (f2cl-lib:int-sub kplus1 j))
                        (if nounit
                            (setf (f2cl-lib:fref x-%data%
                                                 (jx)
                                                 ((1 *))
                                                 x-%offset%)
                                  (/
                                   (f2cl-lib:fref x-%data%
                                                  (jx)
                                                  ((1 *))
                                                  x-%offset%)
                                   (f2cl-lib:fref a-%data%
                                                  (kplus1 j)
                                                  ((1 lda) (1 *))
```

```
                                        a-%offset%))))
                    (setf temp
                            (f2cl-lib:fref x-%data%
                                           (jx)
                                           ((1 *))
                                           x-%offset%))
                    (f2cl-lib:fdo (i
                                   (f2cl-lib:int-add j
                                                     (f2cl-lib:int-sub 1))
                                   (f2cl-lib:int-add i
                                                     (f2cl-lib:int-sub 1)))
                                  ((> i
                                      (max (the fixnum 1)
                                           (the fixnum
                                                (f2cl-lib:int-add j
                                                                  (f2cl-lib:int-s
                                                                   k)))))
                                   nil)
                      (tagbody
                        (setf (f2cl-lib:fref x-%data%
                                             (ix)
                                             ((1 *))
                                             x-%offset%)
                              (-
                               (f2cl-lib:fref x-%data%
                                              (ix)
                                              ((1 *))
                                              x-%offset%)
                               (* temp
                                  (f2cl-lib:fref a-%data%
                                                 ((f2cl-lib:int-add l i)
                                                  j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))))
                        (setf ix (f2cl-lib:int-sub ix incx))))))
                (setf jx (f2cl-lib:int-sub jx incx)))))))
        (t
         (cond
           ((= incx 1)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                          ((> j n) nil)
              (tagbody
                (cond
                  ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                   (setf l (f2cl-lib:int-sub 1 j))
                   (if nounit
```

```
                    (setf (f2cl-lib:fref x-%data%
                                         (j)
                                         ((1 *))
                                         x-%offset%)
                          (/
                           (f2cl-lib:fref x-%data%
                                          (j)
                                          ((1 *))
                                          x-%offset%)
                           (f2cl-lib:fref a-%data%
                                          (1 j)
                                          ((1 lda) (1 *))
                                          a-%offset%))))
                  (setf temp
                        (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                  (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                   (f2cl-lib:int-add i 1))
                                ((> i
                                    (min (the fixnum n)
                                         (the fixnum
                                              (f2cl-lib:int-add j k))))
                                 nil)
                    (tagbody
                      (setf (f2cl-lib:fref x-%data%
                                           (i)
                                           ((1 *))
                                           x-%offset%)
                            (-
                             (f2cl-lib:fref x-%data%
                                            (i)
                                            ((1 *))
                                            x-%offset%)
                             (* temp
                                (f2cl-lib:fref a-%data%
                                               ((f2cl-lib:int-add l i)
                                                j)
                                               ((1 lda) (1 *))
                                               a-%offset%)))))))))
          (t
           (setf jx kx)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf kx (f2cl-lib:int-add kx incx))
               (cond
                 ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
```

```
(setf ix kx)
(setf l (f2cl-lib:int-sub 1 j))
(if nounit
    (setf (f2cl-lib:fref x-%data%
                         (jx)
                         ((1 *))
                         x-%offset%)
          (/
           (f2cl-lib:fref x-%data%
                          (jx)
                          ((1 *))
                          x-%offset%)
           (f2cl-lib:fref a-%data%
                          (1 j)
                          ((1 lda) (1 *))
                          a-%offset%))))
(setf temp
      (f2cl-lib:fref x-%data%
                     (jx)
                     ((1 *))
                     x-%offset%))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                 (f2cl-lib:int-add i 1))
              ((> i
                  (min (the fixnum n)
                       (the fixnum
                            (f2cl-lib:int-add j k))))
               nil)
  (tagbody
    (setf (f2cl-lib:fref x-%data%
                         (ix)
                         ((1 *))
                         x-%offset%)
          (-
           (f2cl-lib:fref x-%data%
                          (ix)
                          ((1 *))
                          x-%offset%)
           (* temp
              (f2cl-lib:fref a-%data%
                             ((f2cl-lib:int-add l i)
                              j)
                             ((1 lda) (1 *))
                             a-%offset%))))
    (setf ix (f2cl-lib:int-add ix incx))))))
(setf jx (f2cl-lib:int-add jx incx)))))))))
```

```
(t
 (cond
   ((char-equal uplo #\U)
    (setf kplus1 (f2cl-lib:int-add k 1))
    (cond
      ((= incx 1)
       (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                     ((> j n) nil)
         (tagbody
           (setf temp
                   (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
           (setf l (f2cl-lib:int-sub kplus1 j))
           (cond
             (noconj
              (f2cl-lib:fdo (i
                             (max (the fixnum 1)
                                  (the fixnum
                                       (f2cl-lib:int-add j
                                                         (f2cl-lib:int-sub
                                                          k))))
                             (f2cl-lib:int-add i 1))
                            ((> i
                                (f2cl-lib:int-add j
                                                  (f2cl-lib:int-sub
                                                   1)))
                             nil)
                (tagbody
                  (setf temp
                          (- temp
                             (*
                              (f2cl-lib:fref a-%data%
                                             ((f2cl-lib:int-add l i)
                                              j)
                                             ((1 lda) (1 *))
                                             a-%offset%)
                              (f2cl-lib:fref x-%data%
                                             (i)
                                             ((1 *))
                                             x-%offset%))))))
                (if nounit
                    (setf temp
                            (/ temp
                               (f2cl-lib:fref a-%data%
                                              (kplus1 j)
                                              ((1 lda) (1 *))
                                              a-%offset%)))))
```

```
                (t
                 (f2cl-lib:fdo (i
                                (max (the fixnum 1)
                                     (the fixnum
                                          (f2cl-lib:int-add j
                                                            (f2cl-lib:int-sub
                                                             k))))
                                (f2cl-lib:int-add i 1))
                               ((> i
                                   (f2cl-lib:int-add j
                                                     (f2cl-lib:int-sub
                                                      1)))
                                nil)
                   (tagbody
                     (setf temp
                             (- temp
                                (*
                                 (f2cl-lib:dconjg
                                  (f2cl-lib:fref a-%data%
                                                 ((f2cl-lib:int-add l i)
                                                  j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))
                                 (f2cl-lib:fref x-%data%
                                                (i)
                                                ((1 *))
                                                x-%offset%))))))
                 (if nounit
                     (setf temp
                             (/ temp
                                (f2cl-lib:dconjg
                                 (f2cl-lib:fref a-%data%
                                                (kplus1 j)
                                                ((1 lda) (1 *))
                                                a-%offset%)))))))
             (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                    temp))))
        (t
         (setf jx kx)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (setf temp
                    (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
             (setf ix kx)
             (setf l (f2cl-lib:int-sub kplus1 j))
```

```
(cond
  (noconj
   (f2cl-lib:fdo (i
                  (max (the fixnum 1)
                       (the fixnum
                            (f2cl-lib:int-add j
                                              (f2cl-lib:int-sub
                                               k))))
                  (f2cl-lib:int-add i 1))
                 ((> i
                     (f2cl-lib:int-add j
                                       (f2cl-lib:int-sub
                                        1)))
                  nil)
     (tagbody
       (setf temp
             (- temp
                (*
                 (f2cl-lib:fref a-%data%
                                ((f2cl-lib:int-add l i)
                                 j)
                                ((1 lda) (1 *))
                                a-%offset%)
                 (f2cl-lib:fref x-%data%
                                (ix)
                                ((1 *))
                                x-%offset%))))
       (setf ix (f2cl-lib:int-add ix incx))))
   (if nounit
       (setf temp
             (/ temp
                (f2cl-lib:fref a-%data%
                               (kplus1 j)
                               ((1 lda) (1 *))
                               a-%offset%)))))
  (t
   (f2cl-lib:fdo (i
                  (max (the fixnum 1)
                       (the fixnum
                            (f2cl-lib:int-add j
                                              (f2cl-lib:int-sub
                                               k))))
                  (f2cl-lib:int-add i 1))
                 ((> i
                     (f2cl-lib:int-add j
                                       (f2cl-lib:int-sub
```

```
                                                    1)))
                                  nil)
                            (tagbody
                              (setf temp
                                    (- temp
                                       (*
                                        (f2cl-lib:dconjg
                                         (f2cl-lib:fref a-%data%
                                                        ((f2cl-lib:int-add l i)
                                                         j)
                                                        ((1 lda) (1 *))
                                                        a-%offset%))
                                         (f2cl-lib:fref x-%data%
                                                        (ix)
                                                        ((1 *))
                                                        x-%offset%))))
                              (setf ix (f2cl-lib:int-add ix incx))))
                           (if nounit
                               (setf temp
                                     (/ temp
                                        (f2cl-lib:dconjg
                                         (f2cl-lib:fref a-%data%
                                                        (kplus1 j)
                                                        ((1 lda) (1 *))
                                                        a-%offset%)))))))
                       (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                             temp)
                       (setf jx (f2cl-lib:int-add jx incx))
                       (if (> j k) (setf kx (f2cl-lib:int-add kx incx)))))))
              (t
               (cond
                 ((= incx 1)
                  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                ((> j 1) nil)
                    (tagbody
                      (setf temp
                            (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                      (setf l (f2cl-lib:int-sub 1 j))
                      (cond
                        (noconj
                         (f2cl-lib:fdo (i
                                        (min (the fixnum n)
                                             (the fixnum
                                                  (f2cl-lib:int-add j k)))
                                        (f2cl-lib:int-add i
                                                          (f2cl-lib:int-sub 1)))
```

```
                ((> i (f2cl-lib:int-add j 1)) nil)
  (tagbody
    (setf temp
            (- temp
               (*
                (f2cl-lib:fref a-%data%
                               ((f2cl-lib:int-add l i)
                                j)
                               ((1 lda) (1 *))
                               a-%offset%)
                (f2cl-lib:fref x-%data%
                               (i)
                               ((1 *))
                               x-%offset%))))))
 (if nounit
     (setf temp
             (/ temp
                (f2cl-lib:fref a-%data%
                               (1 j)
                               ((1 lda) (1 *))
                               a-%offset%)))))
(t
 (f2cl-lib:fdo (i
                (min (the fixnum n)
                     (the fixnum
                          (f2cl-lib:int-add j k)))
                (f2cl-lib:int-add i
                                  (f2cl-lib:int-sub 1)))
               ((> i (f2cl-lib:int-add j 1)) nil)
  (tagbody
    (setf temp
            (- temp
               (*
                (f2cl-lib:dconjg
                 (f2cl-lib:fref a-%data%
                                ((f2cl-lib:int-add l i)
                                 j)
                                ((1 lda) (1 *))
                                a-%offset%))
                (f2cl-lib:fref x-%data%
                               (i)
                               ((1 *))
                               x-%offset%))))))
 (if nounit
     (setf temp
             (/ temp
```

```
                          (f2cl-lib:dconjg
                           (f2cl-lib:fref a-%data%
                                          (1 j)
                                          ((1 lda) (1 *))
                                          a-%offset%)))))))
      (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
            temp))))
(t
 (setf kx
       (f2cl-lib:int-add kx
                         (f2cl-lib:int-mul
                          (f2cl-lib:int-sub n 1)
                          incx)))
 (setf jx kx)
 (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
               ((> j 1) nil)
   (tagbody
     (setf temp
           (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
     (setf ix kx)
     (setf l (f2cl-lib:int-sub 1 j))
     (cond
       (noconj
        (f2cl-lib:fdo (i
                       (min (the fixnum n)
                            (the fixnum
                                 (f2cl-lib:int-add j k)))
                       (f2cl-lib:int-add i
                                         (f2cl-lib:int-sub 1)))
                      ((> i (f2cl-lib:int-add j 1)) nil)
          (tagbody
            (setf temp
                  (- temp
                     (*
                      (f2cl-lib:fref a-%data%
                                     ((f2cl-lib:int-add l i)
                                      j)
                                     ((1 lda) (1 *))
                                     a-%offset%)
                      (f2cl-lib:fref x-%data%
                                     (ix)
                                     ((1 *))
                                     x-%offset%))))
            (setf ix (f2cl-lib:int-sub ix incx))))
        (if nounit
            (setf temp
```

```
                                    (/ temp
                                       (f2cl-lib:fref a-%data%
                                                      (1 j)
                                                      ((1 lda) (1 *))
                                                      a-%offset%)))))
                             (t
                              (f2cl-lib:fdo (i
                                             (min (the fixnum n)
                                                  (the fixnum
                                                       (f2cl-lib:int-add j k)))
                                             (f2cl-lib:int-add i
                                                               (f2cl-lib:int-sub 1)))
                                            ((> i (f2cl-lib:int-add j 1)) nil)
                                (tagbody
                                  (setf temp
                                          (- temp
                                             (*
                                              (f2cl-lib:dconjg
                                               (f2cl-lib:fref a-%data%
                                                              ((f2cl-lib:int-add l i)
                                                               j)
                                                              ((1 lda) (1 *))
                                                              a-%offset%))
                                              (f2cl-lib:fref x-%data%
                                                             (ix)
                                                             ((1 *))
                                                             x-%offset%))))
                                      (setf ix (f2cl-lib:int-sub ix incx))))
                               (if nounit
                                   (setf temp
                                           (/ temp
                                              (f2cl-lib:dconjg
                                               (f2cl-lib:fref a-%data%
                                                              (1 j)
                                                              ((1 lda) (1 *))
                                                              a-%offset%)))))))
                           (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                                 temp)
                           (setf jx (f2cl-lib:int-sub jx incx))
                           (if (>= (f2cl-lib:int-sub n j) k)
                               (setf kx (f2cl-lib:int-sub kx incx)))))))))
  end_label
       (return (values nil nil nil nil nil nil nil nil nil))))))
```

5.30 ztpmv BLAS

⟨*ztpmv.input*⟩≡

```
)set break resume
)sys rm -f ztpmv.output
)spool ztpmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*ztpmv.help*⟩≡

```
  ======================================================================
  ztpmv examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        ZTPMV - perform one of the matrix-vector operations   x :=
        A*x, or x := A'*x, or x := conjg( A' )*x,

   SYNOPSIS
        SUBROUTINE ZTPMV ( UPLO, TRANS, DIAG, N, AP, X, INCX )

            INTEGER      INCX, N

            CHARACTER*1  DIAG, TRANS, UPLO

            COMPLEX*16   AP( * ), X( * )

   PURPOSE
        ZTPMV  performs one of the matrix-vector operations

        where x is an n element vector and  A is an n by n unit, or
        non-unit, upper or lower triangular matrix, supplied in
        packed form.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On entry, UPLO specifies whether the matrix is an
                 upper or lower triangular matrix as follows:

                 UPLO = 'U' or 'u'   A is an upper triangular matrix.

                 UPLO = 'L' or 'l'   A is a lower triangular matrix.

                 Unchanged on exit.

        TRANS  - CHARACTER*1.
                 On entry, TRANS specifies the operation to be per-
                 formed as follows:

                 TRANS = 'N' or 'n'   x := A*x.
```

```
         TRANS = 'T' or 't'   x := A'*x.

         TRANS = 'C' or 'c'   x := conjg( A' )*x.

         Unchanged on exit.

DIAG   - CHARACTER*1.
         On entry, DIAG specifies whether or not A is unit
         triangular as follows:

         DIAG = 'U' or 'u'   A is assumed to be unit triangu-
         lar.

         DIAG = 'N' or 'n'   A is not assumed to be unit

         triangular.

         Unchanged on exit.

N      - INTEGER.
         On entry, N specifies the order of the matrix A.  N
         must be at least zero.  Unchanged on exit.

AP     - COMPLEX*16       array of DIMENSION at least
         ( ( n*( n + 1 ) )/2 ).  Before entry with  UPLO = 'U'
         or 'u', the array AP must contain the upper triangu-
         lar matrix packed sequentially, column by column, so
         that AP( 1 ) contains a( 1, 1 ), AP( 2 ) and AP( 3 )
         contain a( 1, 2 ) and a( 2, 2 ) respectively, and so
         on.  Before entry with UPLO = 'L' or 'l', the array
         AP must contain the lower triangular matrix packed
         sequentially, column by column, so that AP( 1 ) con-
         tains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1
         ) and a( 3, 1 ) respectively, and so on.  Note that
         when  DIAG = 'U' or 'u', the diagonal elements of A
         are not referenced, but are assumed to be unity.
         Unchanged on exit.

X      - COMPLEX*16       array of dimension at least
         ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
         incremented array X must contain the n element vector
         x. On exit, X is overwritten with the transformed
         vector x.

INCX   - INTEGER.
         On entry, INCX specifies the increment for the ele-
```

```
ments of X. INCX must not be zero.  Unchanged on
exit.
```

⟨*BLAS 2 ztpmv*⟩≡

```
(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun ztpmv (uplo trans diag n ap x incx)
    (declare (type (simple-array (complex double-float) (*)) x ap)
             (type fixnum incx n)
             (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (diag character diag-%data% diag-%offset%)
         (ap (complex double-float) ap-%data% ap-%offset%)
         (x (complex double-float) x-%data% x-%offset%))
      (prog ((noconj nil) (nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0) (k 0)
             (kk 0) (kx 0) (temp #C(0.0 0.0)))
        (declare (type (member t nil) noconj nounit)
                 (type fixnum i info ix j jx k kk kx)
                 (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 2))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((= incx 0)
           (setf info 7)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZTPMV" info)
           (go end_label)))
        (if (= n 0) (go end_label))
        (setf noconj (char-equal trans #\T))
        (setf nounit (char-equal diag #\N))
        (cond
          ((<= incx 0)
           (setf kx
                   (f2cl-lib:int-sub 1
                                     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
```

```
                                                    incx))))
          ((/= incx 1)
           (setf kx 1)))
        (cond
          ((char-equal trans #\N)
           (cond
             ((char-equal uplo #\U)
              (setf kk 1)
              (cond
                ((= incx 1)
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (cond
                       ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                        (setf temp
                                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                        (setf k kk)
                        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                      ((> i
                                          (f2cl-lib:int-add j
                                                            (f2cl-lib:int-sub
                                                             1)))
                                       nil)
                          (tagbody
                            (setf (f2cl-lib:fref x-%data%
                                                 (i)
                                                 ((1 *))
                                                 x-%offset%)
                                  (+
                                   (f2cl-lib:fref x-%data%
                                                  (i)
                                                  ((1 *))
                                                  x-%offset%)
                                   (* temp
                                      (f2cl-lib:fref ap-%data%
                                                     (k)
                                                     ((1 *))
                                                     ap-%offset%))))
                            (setf k (f2cl-lib:int-add k 1))))
                        (if nounit
                            (setf (f2cl-lib:fref x-%data%
                                                 (j)
                                                 ((1 *))
                                                 x-%offset%)
                                  (*
```

```
                                (f2cl-lib:fref x-%data%
                                               (j)
                                               ((1 *))
                                               x-%offset%)
                                (f2cl-lib:fref ap-%data%
                                               ((f2cl-lib:int-sub
                                                 (f2cl-lib:int-add kk j)
                                                 1))
                                               ((1 *))
                                               ap-%offset%))))))
                    (setf kk (f2cl-lib:int-add kk j)))))
               (t
                (setf jx kx)
                (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                              ((> j n) nil)
                  (tagbody
                    (cond
                      ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                       (setf temp
                               (f2cl-lib:fref x-%data%
                                              (jx)
                                              ((1 *))
                                              x-%offset%))
                       (setf ix kx)
                       (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
                                     ((> k
                                         (f2cl-lib:int-add kk
                                                           j
                                                           (f2cl-lib:int-sub
                                                            2)))
                                      nil)
                         (tagbody
                           (setf (f2cl-lib:fref x-%data%
                                                (ix)
                                                ((1 *))
                                                x-%offset%)
                                 (+
                                  (f2cl-lib:fref x-%data%
                                                 (ix)
                                                 ((1 *))
                                                 x-%offset%)
                                  (* temp
                                     (f2cl-lib:fref ap-%data%
                                                    (k)
                                                    ((1 *))
                                                    ap-%offset%))))
```

```
                    (setf ix (f2cl-lib:int-add ix incx))))
                (if nounit
                    (setf (f2cl-lib:fref x-%data%
                                         (jx)
                                         ((1 *))
                                         x-%offset%)
                          (*
                           (f2cl-lib:fref x-%data%
                                          (jx)
                                          ((1 *))
                                          x-%offset%)
                           (f2cl-lib:fref ap-%data%
                                          ((f2cl-lib:int-sub
                                            (f2cl-lib:int-add kk j)
                                            1))
                                          ((1 *))
                                          ap-%offset%))))))
            (setf jx (f2cl-lib:int-add jx incx))
            (setf kk (f2cl-lib:int-add kk j)))))))
(t
 (setf kk (the fixnum (truncate (* n (+ n 1)) 2)))
 (cond
   ((= incx 1)
    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                  ((> j 1) nil)
      (tagbody
        (cond
          ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
           (setf temp
                   (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
           (setf k kk)
           (f2cl-lib:fdo (i n
                          (f2cl-lib:int-add i
                                            (f2cl-lib:int-sub 1)))
                         ((> i (f2cl-lib:int-add j 1)) nil)
             (tagbody
               (setf (f2cl-lib:fref x-%data%
                                    (i)
                                    ((1 *))
                                    x-%offset%)
                     (+
                      (f2cl-lib:fref x-%data%
                                     (i)
                                     ((1 *))
                                     x-%offset%)
                      (* temp
```

```
                                   (f2cl-lib:fref ap-%data%
                                                  (k)
                                                  ((1 *))
                                                  ap-%offset%))))
                           (setf k (f2cl-lib:int-sub k 1))))
                       (if nounit
                           (setf (f2cl-lib:fref x-%data%
                                                (j)
                                                ((1 *))
                                                x-%offset%)
                                 (*
                                  (f2cl-lib:fref x-%data%
                                                 (j)
                                                 ((1 *))
                                                 x-%offset%)
                                  (f2cl-lib:fref ap-%data%
                                                 ((f2cl-lib:int-add
                                                   (f2cl-lib:int-sub kk n)
                                                   j))
                                                 ((1 *))
                                                 ap-%offset%))))))
                   (setf kk
                         (f2cl-lib:int-sub kk
                                           (f2cl-lib:int-add
                                            (f2cl-lib:int-sub n j)
                                            1))))))
              (t
               (setf kx
                     (f2cl-lib:int-add kx
                                       (f2cl-lib:int-mul
                                        (f2cl-lib:int-sub n 1)
                                        incx)))
               (setf jx kx)
               (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                             ((> j 1) nil)
                 (tagbody
                   (cond
                     ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                      (setf temp
                            (f2cl-lib:fref x-%data%
                                           (jx)
                                           ((1 *))
                                           x-%offset%))
                      (setf ix kx)
                      (f2cl-lib:fdo (k kk
                                     (f2cl-lib:int-add k
```

```
                                                (f2cl-lib:int-sub 1)))
                                 ((> k
                                     (f2cl-lib:int-add kk
                                                       (f2cl-lib:int-sub
                                                        (f2cl-lib:int-add
                                                         n
                                                         (f2cl-lib:int-sub
                                                          (f2cl-lib:int-add
                                                           j
                                                           1))))))
                                  nil)
                           (tagbody
                             (setf (f2cl-lib:fref x-%data%
                                                  (ix)
                                                  ((1 *))
                                                  x-%offset%)
                                   (+
                                    (f2cl-lib:fref x-%data%
                                                   (ix)
                                                   ((1 *))
                                                   x-%offset%)
                                    (* temp
                                       (f2cl-lib:fref ap-%data%
                                                      (k)
                                                      ((1 *))
                                                      ap-%offset%))))
                             (setf ix (f2cl-lib:int-sub ix incx))))
                          (if nounit
                              (setf (f2cl-lib:fref x-%data%
                                                   (jx)
                                                   ((1 *))
                                                   x-%offset%)
                                    (*
                                     (f2cl-lib:fref x-%data%
                                                    (jx)
                                                    ((1 *))
                                                    x-%offset%)
                                     (f2cl-lib:fref ap-%data%
                                                    ((f2cl-lib:int-add
                                                      (f2cl-lib:int-sub kk n)
                                                      j))
                                                    ((1 *))
                                                    ap-%offset%))))))
                      (setf jx (f2cl-lib:int-sub jx incx))
                      (setf kk
                            (f2cl-lib:int-sub kk
```

```
                                          (f2cl-lib:int-add
                                           (f2cl-lib:int-sub n j)
                                           1))))))))))
                (t
                 (cond
                   ((char-equal uplo #\U)
                    (setf kk (the fixnum (truncate (* n (+ n 1)) 2)))
                    (cond
                      ((= incx 1)
                       (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                     ((> j 1) nil)
                         (tagbody
                           (setf temp
                                   (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                           (setf k (f2cl-lib:int-sub kk 1))
                           (cond
                             (noconj
                              (if nounit
                                  (setf temp
                                          (* temp
                                             (f2cl-lib:fref ap-%data%
                                                            (kk)
                                                            ((1 *))
                                                            ap-%offset%))))
                              (f2cl-lib:fdo (i
                                             (f2cl-lib:int-add j
                                                               (f2cl-lib:int-sub 1))
                                             (f2cl-lib:int-add i
                                                               (f2cl-lib:int-sub 1)))
                                            ((> i 1) nil)
                                (tagbody
                                  (setf temp
                                          (+ temp
                                             (*
                                              (f2cl-lib:fref ap-%data%
                                                             (k)
                                                             ((1 *))
                                                             ap-%offset%)
                                              (f2cl-lib:fref x-%data%
                                                             (i)
                                                             ((1 *))
                                                             x-%offset%))))
                                  (setf k (f2cl-lib:int-sub k 1)))))
                             (t
                              (if nounit
                                  (setf temp
```

```
                              (* temp
                                 (f2cl-lib:dconjg
                                  (f2cl-lib:fref ap-%data%
                                                 (kk)
                                                 ((1 *))
                                                 ap-%offset%)))))
                         (f2cl-lib:fdo (i
                                        (f2cl-lib:int-add j
                                                          (f2cl-lib:int-sub 1))
                                        (f2cl-lib:int-add i
                                                          (f2cl-lib:int-sub 1)))
                                       ((> i 1) nil)
                           (tagbody
                             (setf temp
                                     (+ temp
                                        (*
                                         (f2cl-lib:dconjg
                                          (f2cl-lib:fref ap-%data%
                                                         (k)
                                                         ((1 *))
                                                         ap-%offset%))
                                         (f2cl-lib:fref x-%data%
                                                        (i)
                                                        ((1 *))
                                                        x-%offset%))))
                             (setf k (f2cl-lib:int-sub k 1))))))
                     (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                             temp)
                     (setf kk (f2cl-lib:int-sub kk j)))))
                (t
                 (setf jx
                         (f2cl-lib:int-add kx
                                           (f2cl-lib:int-mul
                                            (f2cl-lib:int-sub n 1)
                                            incx)))
                 (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                               ((> j 1) nil)
                   (tagbody
                     (setf temp
                             (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
                     (setf ix jx)
                     (cond
                       (noconj
                        (if nounit
                            (setf temp
                                    (* temp
```

```
                (f2cl-lib:fref ap-%data%
                               (kk)
                               ((1 *))
                               ap-%offset%))))
 (f2cl-lib:fdo (k
                (f2cl-lib:int-add kk
                                  (f2cl-lib:int-sub 1))
                (f2cl-lib:int-add k
                                  (f2cl-lib:int-sub 1)))
               ((> k
                   (f2cl-lib:int-add kk
                                     (f2cl-lib:int-sub
                                      j)
                                     1))
                nil)
   (tagbody
     (setf ix (f2cl-lib:int-sub ix incx))
     (setf temp
             (+ temp
                (*
                 (f2cl-lib:fref ap-%data%
                                (k)
                                ((1 *))
                                ap-%offset%)
                 (f2cl-lib:fref x-%data%
                                (ix)
                                ((1 *))
                                x-%offset%)))))))
(t
 (if nounit
     (setf temp
             (* temp
                (f2cl-lib:dconjg
                 (f2cl-lib:fref ap-%data%
                                (kk)
                                ((1 *))
                                ap-%offset%)))))
 (f2cl-lib:fdo (k
                (f2cl-lib:int-add kk
                                  (f2cl-lib:int-sub 1))
                (f2cl-lib:int-add k
                                  (f2cl-lib:int-sub 1)))
               ((> k
                   (f2cl-lib:int-add kk
                                     (f2cl-lib:int-sub
                                      j)
```

```
                                                    1))
                                 nil)
                          (tagbody
                            (setf ix (f2cl-lib:int-sub ix incx))
                            (setf temp
                                  (+ temp
                                     (*
                                      (f2cl-lib:dconjg
                                       (f2cl-lib:fref ap-%data%
                                                      (k)
                                                      ((1 *))
                                                      ap-%offset%))
                                       (f2cl-lib:fref x-%data%
                                                      (ix)
                                                      ((1 *))
                                                      x-%offset%))))))))
                  (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                          temp)
                  (setf jx (f2cl-lib:int-sub jx incx))
                  (setf kk (f2cl-lib:int-sub kk j)))))))
          (t
           (setf kk 1)
           (cond
             ((= incx 1)
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (setf temp
                          (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                  (setf k (f2cl-lib:int-add kk 1))
                  (cond
                    (noconj
                     (if nounit
                         (setf temp
                                 (* temp
                                    (f2cl-lib:fref ap-%data%
                                                   (kk)
                                                   ((1 *))
                                                   ap-%offset%))))
                     (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                      (f2cl-lib:int-add i 1))
                                   ((> i n) nil)
                       (tagbody
                         (setf temp
                                 (+ temp
                                    (*
```

```
                                (f2cl-lib:fref ap-%data%
                                               (k)
                                               ((1 *))
                                               ap-%offset%)
                                (f2cl-lib:fref x-%data%
                                               (i)
                                               ((1 *))
                                               x-%offset%))))
                        (setf k (f2cl-lib:int-add k 1)))))
                   (t
                    (if nounit
                        (setf temp
                                (* temp
                                   (f2cl-lib:dconjg
                                    (f2cl-lib:fref ap-%data%
                                                   (kk)
                                                   ((1 *))
                                                   ap-%offset%)))))
                    (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                   (f2cl-lib:int-add i 1))
                                  ((> i n) nil)
                      (tagbody
                        (setf temp
                                (+ temp
                                   (*
                                    (f2cl-lib:dconjg
                                     (f2cl-lib:fref ap-%data%
                                                    (k)
                                                    ((1 *))
                                                    ap-%offset%))
                                    (f2cl-lib:fref x-%data%
                                                   (i)
                                                   ((1 *))
                                                   x-%offset%))))
                        (setf k (f2cl-lib:int-add k 1))))))
                   (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                         temp)
                   (setf kk
                         (f2cl-lib:int-add kk
                                          (f2cl-lib:int-add
                                           (f2cl-lib:int-sub n j)
                                           1))))))
              (t
               (setf jx kx)
               (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                             ((> j n) nil)
```

```
(tagbody
  (setf temp
         (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
  (setf ix jx)
  (cond
    (noconj
     (if nounit
         (setf temp
                 (* temp
                    (f2cl-lib:fref ap-%data%
                                   (kk)
                                   ((1 *))
                                   ap-%offset%))))
     (f2cl-lib:fdo (k (f2cl-lib:int-add kk 1)
                    (f2cl-lib:int-add k 1))
                   ((> k
                       (f2cl-lib:int-add kk
                                         n
                                         (f2cl-lib:int-sub
                                          j)))
                    nil)
       (tagbody
         (setf ix (f2cl-lib:int-add ix incx))
         (setf temp
                 (+ temp
                    (*
                     (f2cl-lib:fref ap-%data%
                                    (k)
                                    ((1 *))
                                    ap-%offset%)
                     (f2cl-lib:fref x-%data%
                                    (ix)
                                    ((1 *))
                                    x-%offset%)))))))
    (t
     (if nounit
         (setf temp
                 (* temp
                    (f2cl-lib:dconjg
                     (f2cl-lib:fref ap-%data%
                                    (kk)
                                    ((1 *))
                                    ap-%offset%)))))
     (f2cl-lib:fdo (k (f2cl-lib:int-add kk 1)
                    (f2cl-lib:int-add k 1))
                   ((> k
```

```
                                        (f2cl-lib:int-add kk
                                                          n
                                                          (f2cl-lib:int-sub
                                                           j)))
                                    nil)
                                (tagbody
                                  (setf ix (f2cl-lib:int-add ix incx))
                                  (setf temp
                                          (+ temp
                                             (*
                                              (f2cl-lib:dconjg
                                               (f2cl-lib:fref ap-%data%
                                                              (k)
                                                              ((1 *))
                                                              ap-%offset%))
                                              (f2cl-lib:fref x-%data%
                                                             (ix)
                                                             ((1 *))
                                                             x-%offset%))))))))
                        (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                                temp)
                        (setf jx (f2cl-lib:int-add jx incx))
                        (setf kk
                                (f2cl-lib:int-add kk
                                                  (f2cl-lib:int-add
                                                   (f2cl-lib:int-sub n j)
                                                   1)))))))))
   end_label
          (return (values nil nil nil nil nil nil nil))))))
```

5.31 ztpsv BLAS

⟨*ztpsv.input*⟩≡

```
  )set break resume
  )sys rm -f ztpsv.output
  )spool ztpsv.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*ztpsv.help*⟩≡

```
  ==========================================================================
  ztpsv examples
  ==========================================================================

  ==========================================================================
  Man Page Details
  ==========================================================================

   NAME
        ZTPSV - solve one of the systems of equations   A*x = b, or
        A'*x = b, or conjg( A' )*x = b,

   SYNOPSIS
        SUBROUTINE ZTPSV ( UPLO, TRANS, DIAG, N, AP, X, INCX )

            INTEGER      INCX, N

            CHARACTER*1  DIAG, TRANS, UPLO

            COMPLEX*16   AP( * ), X( * )

   PURPOSE
        ZTPSV  solves one of the systems of equations

        where b and x are n element vectors and A is an n by n unit,
        or non-unit, upper or lower triangular matrix, supplied in
        packed form.

        No test for singularity or near-singularity is included in
        this routine. Such tests must be performed before calling
        this routine.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On entry, UPLO specifies whether the matrix is an
                 upper or lower triangular matrix as follows:

                 UPLO = 'U' or 'u'   A is an upper triangular matrix.

                 UPLO = 'L' or 'l'   A is a lower triangular matrix.

                 Unchanged on exit.

        TRANS  - CHARACTER*1.
                 On entry, TRANS specifies the equations to be solved
```

```
          as follows:

          TRANS = 'N' or 'n'   A*x = b.

          TRANS = 'T' or 't'   A'*x = b.

          TRANS = 'C' or 'c'   conjg( A' )*x = b.

          Unchanged on exit.

DIAG   - CHARACTER*1.
          On entry, DIAG specifies whether or not A is unit
          triangular as follows:

          DIAG = 'U' or 'u'   A is assumed to be unit triangu-
          lar.

          DIAG = 'N' or 'n'   A is not assumed to be unit tri-
          angular.

          Unchanged on exit.

N      - INTEGER.
          On entry, N specifies the order of the matrix A.  N
          must be at least zero.  Unchanged on exit.

AP     - COMPLEX*16       array of DIMENSION at least
          ( ( n*( n + 1 ) )/2 ).  Before entry with  UPLO = 'U'
          or 'u', the array AP must contain the upper triangu-
          lar matrix packed sequentially, column by column, so
          that AP( 1 ) contains a( 1, 1 ), AP( 2 ) and AP( 3 )
          contain a( 1, 2 ) and a( 2, 2 ) respectively, and so
          on.  Before entry with UPLO = 'L' or 'l', the array
          AP must contain the lower triangular matrix packed
          sequentially, column by column, so that AP( 1 ) con-
          tains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1
          ) and a( 3, 1 ) respectively, and so on.  Note that
          when  DIAG = 'U' or 'u', the diagonal elements of A
          are not referenced, but are assumed to be unity.
          Unchanged on exit.

X      - COMPLEX*16       array of dimension at least
          ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
          incremented array X must contain the n element
          right-hand side vector b. On exit, X is overwritten
          with the solution vector x.
```

```
INCX   - INTEGER.
       On entry, INCX specifies the increment for the ele-
       ments of X. INCX must not be zero.  Unchanged on
       exit.
```

⟨*BLAS 2 ztpsv*⟩≡

```
(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun ztpsv (uplo trans diag n ap x incx)
    (declare (type (simple-array (complex double-float) (*)) x ap)
             (type fixnum incx n)
             (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (diag character diag-%data% diag-%offset%)
         (ap (complex double-float) ap-%data% ap-%offset%)
         (x (complex double-float) x-%data% x-%offset%))
      (prog ((noconj nil) (nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0) (k 0)
             (kk 0) (kx 0) (temp #C(0.0 0.0)))
        (declare (type (member t nil) noconj nounit)
                 (type fixnum i info ix j jx k kk kx)
                 (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 2))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((= incx 0)
           (setf info 7)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZTPSV" info)
           (go end_label)))
        (if (= n 0) (go end_label))
        (setf noconj (char-equal trans #\T))
        (setf nounit (char-equal diag #\N))
        (cond
          ((<= incx 0)
           (setf kx
                   (f2cl-lib:int-sub 1
                                     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
```

```
                                                    incx))))
          ((/= incx 1)
           (setf kx 1)))
        (cond
          ((char-equal trans #\N)
           (cond
             ((char-equal uplo #\U)
              (setf kk (the fixnum (truncate (* n (+ n 1)) 2)))
              (cond
                ((= incx 1)
                 (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                               ((> j 1) nil)
                   (tagbody
                     (cond
                       ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                        (if nounit
                            (setf (f2cl-lib:fref x-%data%
                                                 (j)
                                                 ((1 *))
                                                 x-%offset%)
                                  (/
                                   (f2cl-lib:fref x-%data%
                                                  (j)
                                                  ((1 *))
                                                  x-%offset%)
                                   (f2cl-lib:fref ap-%data%
                                                  (kk)
                                                  ((1 *))
                                                  ap-%offset%))))
                        (setf temp
                                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                        (setf k (f2cl-lib:int-sub kk 1))
                        (f2cl-lib:fdo (i
                                       (f2cl-lib:int-add j
                                                         (f2cl-lib:int-sub 1))
                                       (f2cl-lib:int-add i
                                                         (f2cl-lib:int-sub 1)))
                                      ((> i 1) nil)
                          (tagbody
                            (setf (f2cl-lib:fref x-%data%
                                                 (i)
                                                 ((1 *))
                                                 x-%offset%)
                                  (-
                                   (f2cl-lib:fref x-%data%
                                                  (i)
```

```
                                        ((1 *))
                                        x-%offset%)
                                (* temp
                                   (f2cl-lib:fref ap-%data%
                                                  (k)
                                                  ((1 *))
                                                  ap-%offset%))))
                        (setf k (f2cl-lib:int-sub k 1))))))
                    (setf kk (f2cl-lib:int-sub kk j)))))
               (t
                (setf jx
                      (f2cl-lib:int-add kx
                                        (f2cl-lib:int-mul
                                         (f2cl-lib:int-sub n 1)
                                         incx)))
                (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                              ((> j 1) nil)
                  (tagbody
                    (cond
                      ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                       (if nounit
                           (setf (f2cl-lib:fref x-%data%
                                                (jx)
                                                ((1 *))
                                                x-%offset%)
                                 (/
                                  (f2cl-lib:fref x-%data%
                                                 (jx)
                                                 ((1 *))
                                                 x-%offset%)
                                  (f2cl-lib:fref ap-%data%
                                                 (kk)
                                                 ((1 *))
                                                 ap-%offset%))))
                       (setf temp
                             (f2cl-lib:fref x-%data%
                                            (jx)
                                            ((1 *))
                                            x-%offset%))
                       (setf ix jx)
                       (f2cl-lib:fdo (k
                                      (f2cl-lib:int-add kk
                                                        (f2cl-lib:int-sub 1))
                                      (f2cl-lib:int-add k
                                                        (f2cl-lib:int-sub 1)))
                                     ((> k
```

```
                                        (f2cl-lib:int-add kk
                                                          (f2cl-lib:int-sub
                                                           j)
                                                          1))
                                    nil)
                            (tagbody
                              (setf ix (f2cl-lib:int-sub ix incx))
                              (setf (f2cl-lib:fref x-%data%
                                                   (ix)
                                                   ((1 *))
                                                   x-%offset%)
                                    (-
                                     (f2cl-lib:fref x-%data%
                                                    (ix)
                                                    ((1 *))
                                                    x-%offset%)
                                     (* temp
                                        (f2cl-lib:fref ap-%data%
                                                       (k)
                                                       ((1 *))
                                                       ap-%offset%))))))))
                    (setf jx (f2cl-lib:int-sub jx incx))
                    (setf kk (f2cl-lib:int-sub kk j)))))))
          (t
           (setf kk 1)
           (cond
             ((= incx 1)
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (cond
                    ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                     (if nounit
                         (setf (f2cl-lib:fref x-%data%
                                              (j)
                                              ((1 *))
                                              x-%offset%)
                               (/
                                (f2cl-lib:fref x-%data%
                                               (j)
                                               ((1 *))
                                               x-%offset%)
                                (f2cl-lib:fref ap-%data%
                                               (kk)
                                               ((1 *))
                                               ap-%offset%))))))))
```

```
                (setf temp
                        (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                (setf k (f2cl-lib:int-add kk 1))
                (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                 (f2cl-lib:int-add i 1))
                              ((> i n) nil)
                  (tagbody
                    (setf (f2cl-lib:fref x-%data%
                                         (i)
                                         ((1 *))
                                         x-%offset%)
                                 (-
                                  (f2cl-lib:fref x-%data%
                                                 (i)
                                                 ((1 *))
                                                 x-%offset%)
                                  (* temp
                                     (f2cl-lib:fref ap-%data%
                                                    (k)
                                                    ((1 *))
                                                    ap-%offset%))))
                    (setf k (f2cl-lib:int-add k 1))))))
            (setf kk
                    (f2cl-lib:int-add kk
                                      (f2cl-lib:int-add
                                       (f2cl-lib:int-sub n j)
                                       1))))))
          (t
           (setf jx kx)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (cond
                 ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                  (if nounit
                      (setf (f2cl-lib:fref x-%data%
                                           (jx)
                                           ((1 *))
                                           x-%offset%)
                                   (/
                                    (f2cl-lib:fref x-%data%
                                                   (jx)
                                                   ((1 *))
                                                   x-%offset%)
                                    (f2cl-lib:fref ap-%data%
                                                   (kk)
```

```
                                          ((1 *))
                                          ap-%offset%))))
                    (setf temp
                          (f2cl-lib:fref x-%data%
                                         (jx)
                                         ((1 *))
                                         x-%offset%))
                    (setf ix jx)
                    (f2cl-lib:fdo (k (f2cl-lib:int-add kk 1)
                                     (f2cl-lib:int-add k 1))
                                  ((> k
                                      (f2cl-lib:int-add kk
                                                        n
                                                        (f2cl-lib:int-sub
                                                         j)))
                                   nil)
                      (tagbody
                        (setf ix (f2cl-lib:int-add ix incx))
                        (setf (f2cl-lib:fref x-%data%
                                             (ix)
                                             ((1 *))
                                             x-%offset%)
                              (-
                               (f2cl-lib:fref x-%data%
                                              (ix)
                                              ((1 *))
                                              x-%offset%)
                               (* temp
                                  (f2cl-lib:fref ap-%data%
                                                 (k)
                                                 ((1 *))
                                                 ap-%offset%))))))
                    (setf jx (f2cl-lib:int-add jx incx))
                    (setf kk
                          (f2cl-lib:int-add kk
                                            (f2cl-lib:int-add
                                             (f2cl-lib:int-sub n j)
                                             1)))))))))
            (t
             (cond
               ((char-equal uplo #\U)
                (setf kk 1)
                (cond
                  ((= incx 1)
                   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                 ((> j n) nil)
```

```
(tagbody
  (setf temp
        (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
  (setf k kk)
  (cond
    (noconj
     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                   ((> i
                       (f2cl-lib:int-add j
                                         (f2cl-lib:int-sub
                                          1)))
                    nil)
       (tagbody
         (setf temp
                 (- temp
                    (*
                     (f2cl-lib:fref ap-%data%
                                    (k)
                                    ((1 *))
                                    ap-%offset%)
                     (f2cl-lib:fref x-%data%
                                    (i)
                                    ((1 *))
                                    x-%offset%))))
         (setf k (f2cl-lib:int-add k 1))))
     (if nounit
         (setf temp
                 (/ temp
                    (f2cl-lib:fref ap-%data%
                                   ((f2cl-lib:int-sub
                                     (f2cl-lib:int-add kk j)
                                     1))
                                   ((1 *))
                                   ap-%offset%)))))
    (t
     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                   ((> i
                       (f2cl-lib:int-add j
                                         (f2cl-lib:int-sub
                                          1)))
                    nil)
       (tagbody
         (setf temp
                 (- temp
                    (*
                     (f2cl-lib:dconjg
```

```
                                    (f2cl-lib:fref ap-%data%
                                                   (k)
                                                   ((1 *))
                                                   ap-%offset%))
                                   (f2cl-lib:fref x-%data%
                                                  (i)
                                                  ((1 *))
                                                  x-%offset%))))
                        (setf k (f2cl-lib:int-add k 1))))
                    (if nounit
                        (setf temp
                                (/ temp
                                   (f2cl-lib:dconjg
                                    (f2cl-lib:fref ap-%data%
                                                   ((f2cl-lib:int-sub
                                                     (f2cl-lib:int-add kk
                                                                       j)
                                                     1))
                                                   ((1 *))
                                                   ap-%offset%)))))))
                    (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                            temp)
                    (setf kk (f2cl-lib:int-add kk j)))))
                (t
                 (setf jx kx)
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (setf temp
                             (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
                     (setf ix kx)
                     (cond
                       (noconj
                        (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
                                      ((> k
                                          (f2cl-lib:int-add kk
                                                            j
                                                            (f2cl-lib:int-sub
                                                             2)))
                                       nil)
                          (tagbody
                            (setf temp
                                    (- temp
                                       (*
                                        (f2cl-lib:fref ap-%data%
                                                       (k)
```

```
                                    ((1 *))
                                    ap-%offset%)
                       (f2cl-lib:fref x-%data%
                                    (ix)
                                    ((1 *))
                                    x-%offset%))))
          (setf ix (f2cl-lib:int-add ix incx))))
       (if nounit
          (setf temp
                  (/ temp
                     (f2cl-lib:fref ap-%data%
                                   ((f2cl-lib:int-sub
                                     (f2cl-lib:int-add kk j)
                                     1))
                                   ((1 *))
                                   ap-%offset%)))))
      (t
       (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
                     ((> k
                         (f2cl-lib:int-add kk
                                           j
                                           (f2cl-lib:int-sub
                                            2)))
                      nil)
         (tagbody
          (setf temp
                  (- temp
                     (*
                      (f2cl-lib:dconjg
                       (f2cl-lib:fref ap-%data%
                                      (k)
                                      ((1 *))
                                      ap-%offset%))
                      (f2cl-lib:fref x-%data%
                                     (ix)
                                     ((1 *))
                                     x-%offset%))))
          (setf ix (f2cl-lib:int-add ix incx))))
       (if nounit
          (setf temp
                  (/ temp
                     (f2cl-lib:dconjg
                      (f2cl-lib:fref ap-%data%
                                     ((f2cl-lib:int-sub
                                       (f2cl-lib:int-add kk
                                                         j)
```

```
                                                  1))
                                                ((1 *))
                                                ap-%offset%)))))))
                (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                        temp)
                (setf jx (f2cl-lib:int-add jx incx))
                (setf kk (f2cl-lib:int-add kk j)))))))
          (t
           (setf kk (the fixnum (truncate (* n (+ n 1)) 2)))
           (cond
             ((= incx 1)
              (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                            ((> j 1) nil)
                (tagbody
                  (setf temp
                          (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                  (setf k kk)
                  (cond
                    (noconj
                     (f2cl-lib:fdo (i n
                                    (f2cl-lib:int-add i
                                                      (f2cl-lib:int-sub 1)))
                                   ((> i (f2cl-lib:int-add j 1)) nil)
                       (tagbody
                         (setf temp
                                 (- temp
                                    (*
                                     (f2cl-lib:fref ap-%data%
                                                    (k)
                                                    ((1 *))
                                                    ap-%offset%)
                                     (f2cl-lib:fref x-%data%
                                                    (i)
                                                    ((1 *))
                                                    x-%offset%))))
                         (setf k (f2cl-lib:int-sub k 1))))
                     (if nounit
                         (setf temp
                                 (/ temp
                                    (f2cl-lib:fref ap-%data%
                                                   ((f2cl-lib:int-add
                                                     (f2cl-lib:int-sub kk n)
                                                     j))
                                                   ((1 *))
                                                   ap-%offset%)))))
                    (t
```

```
                (f2cl-lib:fdo (i n
                               (f2cl-lib:int-add i
                                                 (f2cl-lib:int-sub 1)))
                              ((> i (f2cl-lib:int-add j 1)) nil)
                  (tagbody
                    (setf temp
                            (- temp
                               (*
                                (f2cl-lib:dconjg
                                 (f2cl-lib:fref ap-%data%
                                                (k)
                                                ((1 *))
                                                ap-%offset%))
                                (f2cl-lib:fref x-%data%
                                               (i)
                                               ((1 *))
                                               x-%offset%))))
                    (setf k (f2cl-lib:int-sub k 1))))
                (if nounit
                    (setf temp
                            (/ temp
                               (f2cl-lib:dconjg
                                (f2cl-lib:fref ap-%data%
                                               ((f2cl-lib:int-add
                                                 (f2cl-lib:int-sub kk
                                                                   n)
                                                 j))
                                               ((1 *))
                                               ap-%offset%)))))))
            (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                    temp)
            (setf kk
                    (f2cl-lib:int-sub kk
                                      (f2cl-lib:int-add
                                       (f2cl-lib:int-sub n j)
                                       1))))))
          (t
           (setf kx
                   (f2cl-lib:int-add kx
                                     (f2cl-lib:int-mul
                                      (f2cl-lib:int-sub n 1)
                                      incx)))
           (setf jx kx)
           (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                         ((> j 1) nil)
             (tagbody
```

```
(setf temp
        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
(setf ix kx)
(cond
  (noconj
   (f2cl-lib:fdo (k kk
                  (f2cl-lib:int-add k
                                    (f2cl-lib:int-sub 1)))
                 ((> k
                     (f2cl-lib:int-add kk
                                       (f2cl-lib:int-sub
                                        (f2cl-lib:int-add
                                         n
                                         (f2cl-lib:int-sub
                                          (f2cl-lib:int-add
                                           j
                                           1))))))
                  nil)
     (tagbody
       (setf temp
               (- temp
                  (*
                   (f2cl-lib:fref ap-%data%
                                  (k)
                                  ((1 *))
                                  ap-%offset%)
                   (f2cl-lib:fref x-%data%
                                  (ix)
                                  ((1 *))
                                  x-%offset%))))
       (setf ix (f2cl-lib:int-sub ix incx))))
   (if nounit
       (setf temp
               (/ temp
                  (f2cl-lib:fref ap-%data%
                                 ((f2cl-lib:int-add
                                   (f2cl-lib:int-sub kk n)
                                   j))
                                 ((1 *))
                                 ap-%offset%)))))
  (t
   (f2cl-lib:fdo (k kk
                  (f2cl-lib:int-add k
                                    (f2cl-lib:int-sub 1)))
                 ((> k
                     (f2cl-lib:int-add kk
```

```
                                                        (f2cl-lib:int-sub
                                                         (f2cl-lib:int-add
                                                          n
                                                          (f2cl-lib:int-sub
                                                           (f2cl-lib:int-add
                                                            j
                                                            1))))))
                                      nil)
                                  (tagbody
                                    (setf temp
                                            (- temp
                                               (*
                                                (f2cl-lib:dconjg
                                                 (f2cl-lib:fref ap-%data%
                                                                (k)
                                                                ((1 *))
                                                                ap-%offset%))
                                                (f2cl-lib:fref x-%data%
                                                               (ix)
                                                               ((1 *))
                                                               x-%offset%))))
                                    (setf ix (f2cl-lib:int-sub ix incx))))
                                (if nounit
                                    (setf temp
                                            (/ temp
                                               (f2cl-lib:dconjg
                                                (f2cl-lib:fref ap-%data%
                                                               ((f2cl-lib:int-add
                                                                 (f2cl-lib:int-sub kk
                                                                                   n)
                                                                 j))
                                                               ((1 *))
                                                               ap-%offset%)))))))
                            (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                                  temp)
                            (setf jx (f2cl-lib:int-sub jx incx))
                            (setf kk
                                  (f2cl-lib:int-sub kk
                                                    (f2cl-lib:int-add
                                                     (f2cl-lib:int-sub n j)
                                                     1))))))))))
  end_label
       (return (values nil nil nil nil nil nil nil))))))
```

5.32 ztrmv BLAS

⟨*ztrmv.input*⟩≡

```
)set break resume
)sys rm -f ztrmv.output
)spool ztrmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*ztrmv.help*⟩≡

```
==========================================================================
ztrmv examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

 NAME
      ZTRMV - perform one of the matrix-vector operations   x :=
      A*x, or x := A'*x, or x := conjg( A' )*x,

 SYNOPSIS
      SUBROUTINE ZTRMV ( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )

          INTEGER      INCX, LDA, N

          CHARACTER*1  DIAG, TRANS, UPLO

          COMPLEX*16   A( LDA, * ), X( * )

 PURPOSE
      ZTRMV  performs one of the matrix-vector operations

      where x is an n element vector and  A is an n by n unit, or
      non-unit, upper or lower triangular matrix.

 PARAMETERS
      UPLO   - CHARACTER*1.
             On entry, UPLO specifies whether the matrix is an
             upper or lower triangular matrix as follows:

             UPLO = 'U' or 'u'   A is an upper triangular matrix.

             UPLO = 'L' or 'l'   A is a lower triangular matrix.

             Unchanged on exit.

      TRANS  - CHARACTER*1.
             On entry, TRANS specifies the operation to be per-
             formed as follows:

             TRANS = 'N' or 'n'   x := A*x.

             TRANS = 'T' or 't'   x := A'*x.
```

```
          TRANS = 'C' or 'c'   x := conjg( A' )*x.

          Unchanged on exit.

DIAG   - CHARACTER*1.
          On entry, DIAG specifies whether or not A is unit
          triangular as follows:

          DIAG = 'U' or 'u'   A is assumed to be unit triangu-
          lar.

          DIAG = 'N' or 'n'   A is not assumed to be unit tri-
          angular.

          Unchanged on exit.

N      - INTEGER.
          On entry, N specifies the order of the matrix A.  N
          must be at least zero.  Unchanged on exit.

A      - COMPLEX*16       array of DIMENSION ( LDA, n ).
          Before entry with  UPLO = 'U' or 'u', the leading n
          by n upper triangular part of the array A must con-
          tain the upper triangular matrix and the strictly
          lower triangular part of A is not referenced.  Before
          entry with UPLO = 'L' or 'l', the leading n by n
          lower triangular part of the array A must contain the
          lower triangular matrix and the strictly upper tri-
          angular part of A is not referenced.  Note that when
          DIAG = 'U' or 'u', the diagonal elements of A are not
          referenced either, but are assumed to be unity.
          Unchanged on exit.

LDA    - INTEGER.
          On entry, LDA specifies the first dimension of A as
          declared in the calling (sub) program. LDA must be at
          least max( 1, n ).  Unchanged on exit.

X      - COMPLEX*16       array of dimension at least
          ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
          incremented array X must contain the n element vector
          x. On exit, X is overwritten with the transformed
          vector x.

INCX   - INTEGER.
```

```
          On entry, INCX specifies the increment for the ele-
          ments of X. INCX must not be zero.  Unchanged on
          exit.
```

⟨*BLAS 2 ztrmv*⟩≡
```
(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun ztrmv (uplo trans diag n a lda x incx)
    (declare (type (simple-array (complex double-float) (*)) x a)
             (type fixnum incx lda n)
             (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (diag character diag-%data% diag-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (x (complex double-float) x-%data% x-%offset%))
      (prog ((noconj nil) (nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0)
             (kx 0) (temp #C(0.0 0.0)))
        (declare (type (member t nil) noconj nounit)
                 (type fixnum i info ix j jx kx)
                 (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 2))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info 6))
          ((= incx 0)
           (setf info 8)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZTRMV" info)
           (go end_label)))
        (if (= n 0) (go end_label))
        (setf noconj (char-equal trans #\T))
        (setf nounit (char-equal diag #\N))
        (cond
          ((<= incx 0)
           (setf kx
```

```
                (f2cl-lib:int-sub 1
                                  (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                    incx))))
          ((/= incx 1)
           (setf kx 1)))
        (cond
          ((char-equal trans #\N)
           (cond
             ((char-equal uplo #\U)
              (cond
                ((= incx 1)
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (cond
                       ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                        (setf temp
                                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                      ((> i
                                          (f2cl-lib:int-add j
                                                            (f2cl-lib:int-sub
                                                             1)))
                                       nil)
                          (tagbody
                            (setf (f2cl-lib:fref x-%data%
                                                 (i)
                                                 ((1 *))
                                                 x-%offset%)
                                  (+
                                   (f2cl-lib:fref x-%data%
                                                  (i)
                                                  ((1 *))
                                                  x-%offset%)
                                   (* temp
                                      (f2cl-lib:fref a-%data%
                                                     (i j)
                                                     ((1 lda) (1 *))
                                                     a-%offset%))))))
                        (if nounit
                            (setf (f2cl-lib:fref x-%data%
                                                 (j)
                                                 ((1 *))
                                                 x-%offset%)
                                  (*
                                   (f2cl-lib:fref x-%data%
```

```
                                        (j)
                                        ((1 *))
                                        x-%offset%)
                              (f2cl-lib:fref a-%data%
                                        (j j)
                                        ((1 lda) (1 *))
                                        a-%offset%)))))))))
          (t
           (setf jx kx)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (cond
                 ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                  (setf temp
                          (f2cl-lib:fref x-%data%
                                        (jx)
                                        ((1 *))
                                        x-%offset%))
                  (setf ix kx)
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i
                                    (f2cl-lib:int-add j
                                                      (f2cl-lib:int-sub
                                                       1)))
                                 nil)
                    (tagbody
                      (setf (f2cl-lib:fref x-%data%
                                           (ix)
                                           ((1 *))
                                           x-%offset%)
                            (+
                             (f2cl-lib:fref x-%data%
                                           (ix)
                                           ((1 *))
                                           x-%offset%)
                             (* temp
                                (f2cl-lib:fref a-%data%
                                               (i j)
                                               ((1 lda) (1 *))
                                               a-%offset%))))
                      (setf ix (f2cl-lib:int-add ix incx))))
                  (if nounit
                      (setf (f2cl-lib:fref x-%data%
                                           (jx)
                                           ((1 *))
```

```
                                        x-%offset%)
                                (*
                                 (f2cl-lib:fref x-%data%
                                                (jx)
                                                ((1 *))
                                                x-%offset%)
                                 (f2cl-lib:fref a-%data%
                                                (j j)
                                                ((1 lda) (1 *))
                                                a-%offset%))))))
                  (setf jx (f2cl-lib:int-add jx incx)))))))
          (t
           (cond
             ((= incx 1)
              (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                            ((> j 1) nil)
                (tagbody
                  (cond
                    ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                     (setf temp
                             (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                     (f2cl-lib:fdo (i n
                                    (f2cl-lib:int-add i
                                                      (f2cl-lib:int-sub 1)))
                                   ((> i (f2cl-lib:int-add j 1)) nil)
                       (tagbody
                         (setf (f2cl-lib:fref x-%data%
                                              (i)
                                              ((1 *))
                                              x-%offset%)
                                (+
                                 (f2cl-lib:fref x-%data%
                                                (i)
                                                ((1 *))
                                                x-%offset%)
                                 (* temp
                                    (f2cl-lib:fref a-%data%
                                                   (i j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%))))))
                     (if nounit
                         (setf (f2cl-lib:fref x-%data%
                                              (j)
                                              ((1 *))
                                              x-%offset%)
                                (*
```

```
                              (f2cl-lib:fref x-%data%
                                             (j)
                                             ((1 *))
                                             x-%offset%)
                              (f2cl-lib:fref a-%data%
                                             (j j)
                                             ((1 lda) (1 *))
                                             a-%offset%)))))))))
                    (t
                     (setf kx
                           (f2cl-lib:int-add kx
                                             (f2cl-lib:int-mul
                                              (f2cl-lib:int-sub n 1)
                                              incx)))
                     (setf jx kx)
                     (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                   ((> j 1) nil)
                       (tagbody
                         (cond
                           ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                            (setf temp
                                  (f2cl-lib:fref x-%data%
                                                 (jx)
                                                 ((1 *))
                                                 x-%offset%))
                            (setf ix kx)
                            (f2cl-lib:fdo (i n
                                           (f2cl-lib:int-add i
                                                             (f2cl-lib:int-sub 1)))
                                          ((> i (f2cl-lib:int-add j 1)) nil)
                              (tagbody
                                (setf (f2cl-lib:fref x-%data%
                                                     (ix)
                                                     ((1 *))
                                                     x-%offset%)
                                      (+
                                       (f2cl-lib:fref x-%data%
                                                      (ix)
                                                      ((1 *))
                                                      x-%offset%)
                                       (* temp
                                          (f2cl-lib:fref a-%data%
                                                         (i j)
                                                         ((1 lda) (1 *))
                                                         a-%offset%))))
                                (setf ix (f2cl-lib:int-sub ix incx))))
```

```
                    (if nounit
                        (setf (f2cl-lib:fref x-%data%
                                             (jx)
                                             ((1 *))
                                             x-%offset%)
                              (*
                               (f2cl-lib:fref x-%data%
                                              (jx)
                                              ((1 *))
                                              x-%offset%)
                               (f2cl-lib:fref a-%data%
                                              (j j)
                                              ((1 lda) (1 *))
                                              a-%offset%))))))
                (setf jx (f2cl-lib:int-sub jx incx))))))))
          (t
           (cond
             ((char-equal uplo #\U)
              (cond
                ((= incx 1)
                 (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                               ((> j 1) nil)
                   (tagbody
                     (setf temp
                             (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                     (cond
                       (noconj
                        (if nounit
                            (setf temp
                                    (* temp
                                       (f2cl-lib:fref a-%data%
                                                      (j j)
                                                      ((1 lda) (1 *))
                                                      a-%offset%))))
                        (f2cl-lib:fdo (i
                                       (f2cl-lib:int-add j
                                                         (f2cl-lib:int-sub 1))
                                       (f2cl-lib:int-add i
                                                         (f2cl-lib:int-sub 1)))
                                      ((> i 1) nil)
                          (tagbody
                            (setf temp
                                    (+ temp
                                       (*
                                        (f2cl-lib:fref a-%data%
                                                       (i j)
```

```
                                          ((1 lda) (1 *))
                                          a-%offset%)
                                (f2cl-lib:fref x-%data%
                                          (i)
                                          ((1 *))
                                          x-%offset%)))))))
                       (t
                        (if nounit
                            (setf temp
                                    (* temp
                                       (f2cl-lib:dconjg
                                        (f2cl-lib:fref a-%data%
                                                  (j j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%)))))
                        (f2cl-lib:fdo (i
                                       (f2cl-lib:int-add j
                                                         (f2cl-lib:int-sub 1))
                                       (f2cl-lib:int-add i
                                                         (f2cl-lib:int-sub 1)))
                                      ((> i 1) nil)
                          (tagbody
                            (setf temp
                                    (+ temp
                                       (*
                                        (f2cl-lib:dconjg
                                         (f2cl-lib:fref a-%data%
                                                   (i j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%))
                                        (f2cl-lib:fref x-%data%
                                                  (i)
                                                  ((1 *))
                                                  x-%offset%)))))))
                        (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                                temp))))
                  (t
                   (setf jx
                           (f2cl-lib:int-add kx
                                             (f2cl-lib:int-mul
                                              (f2cl-lib:int-sub n 1)
                                              incx)))
                   (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                 ((> j 1) nil)
                     (tagbody
                       (setf temp
```

```
       (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
(setf ix jx)
(cond
  (noconj
   (if nounit
       (setf temp
               (* temp
                  (f2cl-lib:fref a-%data%
                                 (j j)
                                 ((1 lda) (1 *))
                                 a-%offset%))))
   (f2cl-lib:fdo (i
                  (f2cl-lib:int-add j
                                    (f2cl-lib:int-sub 1))
                  (f2cl-lib:int-add i
                                    (f2cl-lib:int-sub 1)))
                 ((> i 1) nil)
     (tagbody
       (setf ix (f2cl-lib:int-sub ix incx))
       (setf temp
               (+ temp
                  (*
                   (f2cl-lib:fref a-%data%
                                  (i j)
                                  ((1 lda) (1 *))
                                  a-%offset%)
                   (f2cl-lib:fref x-%data%
                                  (ix)
                                  ((1 *))
                                  x-%offset%)))))))
  (t
   (if nounit
       (setf temp
               (* temp
                  (f2cl-lib:dconjg
                   (f2cl-lib:fref a-%data%
                                  (j j)
                                  ((1 lda) (1 *))
                                  a-%offset%)))))
   (f2cl-lib:fdo (i
                  (f2cl-lib:int-add j
                                    (f2cl-lib:int-sub 1))
                  (f2cl-lib:int-add i
                                    (f2cl-lib:int-sub 1)))
                 ((> i 1) nil)
     (tagbody
```

```
                        (setf ix (f2cl-lib:int-sub ix incx))
                        (setf temp
                                (+ temp
                                   (*
                                    (f2cl-lib:dconjg
                                     (f2cl-lib:fref a-%data%
                                                    (i j)
                                                    ((1 lda) (1 *))
                                                    a-%offset%))
                                     (f2cl-lib:fref x-%data%
                                                    (ix)
                                                    ((1 *))
                                                    x-%offset%))))))))
                (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                        temp)
                (setf jx (f2cl-lib:int-sub jx incx)))))))
          (t
           (cond
             ((= incx 1)
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (setf temp
                          (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                  (cond
                    (noconj
                     (if nounit
                         (setf temp
                                 (* temp
                                    (f2cl-lib:fref a-%data%
                                                   (j j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%))))
                     (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                      (f2cl-lib:int-add i 1))
                                   ((> i n) nil)
                       (tagbody
                         (setf temp
                                 (+ temp
                                    (*
                                     (f2cl-lib:fref a-%data%
                                                    (i j)
                                                    ((1 lda) (1 *))
                                                    a-%offset%)
                                     (f2cl-lib:fref x-%data%
                                                    (i)
```

```
                                          ((1 *))
                                          x-%offset%)))))))
                (t
                 (if nounit
                     (setf temp
                             (* temp
                                (f2cl-lib:dconjg
                                 (f2cl-lib:fref a-%data%
                                                (j j)
                                                ((1 lda) (1 *))
                                                a-%offset%)))))
                 (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                  (f2cl-lib:int-add i 1))
                               ((> i n) nil)
                   (tagbody
                     (setf temp
                             (+ temp
                                (*
                                 (f2cl-lib:dconjg
                                  (f2cl-lib:fref a-%data%
                                                 (i j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))
                                 (f2cl-lib:fref x-%data%
                                                (i)
                                                ((1 *))
                                                x-%offset%)))))))
                 (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                         temp))))
          (t
           (setf jx kx)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf temp
                       (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
               (setf ix jx)
               (cond
                 (noconj
                  (if nounit
                      (setf temp
                              (* temp
                                 (f2cl-lib:fref a-%data%
                                                (j j)
                                                ((1 lda) (1 *))
                                                a-%offset%))))
```

```
                        (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                       (f2cl-lib:int-add i 1))
                                      ((> i n) nil)
                          (tagbody
                            (setf ix (f2cl-lib:int-add ix incx))
                            (setf temp
                                    (+ temp
                                       (*
                                        (f2cl-lib:fref a-%data%
                                                       (i j)
                                                       ((1 lda) (1 *))
                                                       a-%offset%)
                                        (f2cl-lib:fref x-%data%
                                                       (ix)
                                                       ((1 *))
                                                       x-%offset%)))))))
                       (t
                        (if nounit
                            (setf temp
                                    (* temp
                                       (f2cl-lib:dconjg
                                        (f2cl-lib:fref a-%data%
                                                       (j j)
                                                       ((1 lda) (1 *))
                                                       a-%offset%)))))
                        (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                       (f2cl-lib:int-add i 1))
                                      ((> i n) nil)
                          (tagbody
                            (setf ix (f2cl-lib:int-add ix incx))
                            (setf temp
                                    (+ temp
                                       (*
                                        (f2cl-lib:dconjg
                                         (f2cl-lib:fref a-%data%
                                                        (i j)
                                                        ((1 lda) (1 *))
                                                        a-%offset%))
                                        (f2cl-lib:fref x-%data%
                                                       (ix)
                                                       ((1 *))
                                                       x-%offset%)))))))
                       (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                             temp)
                       (setf jx (f2cl-lib:int-add jx incx))))))))))
  end_label
```

```
          (return (values nil nil nil nil nil nil nil nil))))))
```

5.33 ztrsv BLAS

⟨*ztrsv.input*⟩≡

```
  )set break resume
  )sys rm -f ztrsv.output
  )spool ztrsv.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*ztrsv.help*⟩≡

```
  ==========================================================================
  ztrsv examples
  ==========================================================================

  ==========================================================================
  Man Page Details
  ==========================================================================

   NAME
        ZTRSV - solve one of the systems of equations   A*x = b, or
        A'*x = b, or conjg( A' )*x = b,

   SYNOPSIS
        SUBROUTINE ZTRSV ( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )

            INTEGER      INCX, LDA, N

            CHARACTER*1  DIAG, TRANS, UPLO

            COMPLEX*16   A( LDA, * ), X( * )

   PURPOSE
        ZTRSV  solves one of the systems of equations

        where b and x are n element vectors and A is an n by n unit,
        or non-unit, upper or lower triangular matrix.

        No test for singularity or near-singularity is included in
        this routine. Such tests must be performed before calling
        this routine.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On entry, UPLO specifies whether the matrix is an
                 upper or lower triangular matrix as follows:

                 UPLO = 'U' or 'u'   A is an upper triangular matrix.

                 UPLO = 'L' or 'l'   A is a lower triangular matrix.

                 Unchanged on exit.

        TRANS  - CHARACTER*1.
                 On entry, TRANS specifies the equations to be solved
                 as follows:
```

```
          TRANS = 'N' or 'n'   A*x = b.

          TRANS = 'T' or 't'   A'*x = b.

          TRANS = 'C' or 'c'   conjg( A' )*x = b.

          Unchanged on exit.

DIAG   - CHARACTER*1.
          On entry, DIAG specifies whether or not A is unit
          triangular as follows:

          DIAG = 'U' or 'u'   A is assumed to be unit

          triangular.

          DIAG = 'N' or 'n'   A is not assumed to be unit tri-
          angular.

          Unchanged on exit.

N      - INTEGER.
          On entry, N specifies the order of the matrix A.  N
          must be at least zero.  Unchanged on exit.

A      - COMPLEX*16       array of DIMENSION ( LDA, n ).
          Before entry with  UPLO = 'U' or 'u', the leading n
          by n upper triangular part of the array A must con-
          tain the upper triangular matrix and the strictly
          lower triangular part of A is not referenced.  Before
          entry with UPLO = 'L' or 'l', the leading n by n
          lower triangular part of the array A must contain the
          lower triangular matrix and the strictly upper tri-
          angular part of A is not referenced.  Note that when
          DIAG = 'U' or 'u', the diagonal elements of A are not
          referenced either, but are assumed to be unity.
          Unchanged on exit.

LDA    - INTEGER.
          On entry, LDA specifies the first dimension of A as
          declared in the calling (sub) program. LDA must be at
          least max( 1, n ).  Unchanged on exit.

X      - COMPLEX*16       array of dimension at least
          ( 1 + ( n - 1 )*abs( INCX ) ).  Before entry, the
```

```
          incremented array X must contain the n element
          right-hand side vector b. On exit, X is overwritten
          with the solution vector x.

   INCX   - INTEGER.
          On entry, INCX specifies the increment for the ele-
          ments of X. INCX must not be zero.  Unchanged on
          exit.
```

⟨*BLAS 2 ztrsv*⟩≡

```
(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun ztrsv (uplo trans diag n a lda x incx)
    (declare (type (simple-array (complex double-float) (*)) x a)
             (type fixnum incx lda n)
             (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (diag character diag-%data% diag-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (x (complex double-float) x-%data% x-%offset%))
      (prog ((noconj nil) (nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0)
             (kx 0) (temp #C(0.0 0.0)))
        (declare (type (member t nil) noconj nounit)
                 (type fixnum i info ix j jx kx)
                 (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 2))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info 6))
          ((= incx 0)
           (setf info 8)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZTRSV" info)
           (go end_label)))
        (if (= n 0) (go end_label))
        (setf noconj (char-equal trans #\T))
        (setf nounit (char-equal diag #\N))
        (cond
          ((<= incx 0)
           (setf kx
```

```
                (f2cl-lib:int-sub 1
                                  (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                    incx))))
          ((/= incx 1)
           (setf kx 1)))
        (cond
          ((char-equal trans #\N)
           (cond
             ((char-equal uplo #\U)
              (cond
                ((= incx 1)
                 (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                               ((> j 1) nil)
                   (tagbody
                     (cond
                       ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                        (if nounit
                            (setf (f2cl-lib:fref x-%data%
                                                 (j)
                                                 ((1 *))
                                                 x-%offset%)
                                  (/
                                   (f2cl-lib:fref x-%data%
                                                  (j)
                                                  ((1 *))
                                                  x-%offset%)
                                   (f2cl-lib:fref a-%data%
                                                  (j j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))))
                        (setf temp
                                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                        (f2cl-lib:fdo (i
                                       (f2cl-lib:int-add j
                                                         (f2cl-lib:int-sub 1))
                                       (f2cl-lib:int-add i
                                                         (f2cl-lib:int-sub 1)))
                                      ((> i 1) nil)
                          (tagbody
                            (setf (f2cl-lib:fref x-%data%
                                                 (i)
                                                 ((1 *))
                                                 x-%offset%)
                                  (-
                                   (f2cl-lib:fref x-%data%
                                                  (i)
```

```
                                        ((1 *))
                                        x-%offset%)
                                (* temp
                                   (f2cl-lib:fref a-%data%
                                                  (i j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%)))))))))))
                        (t
                         (setf jx
                                (f2cl-lib:int-add kx
                                                  (f2cl-lib:int-mul
                                                   (f2cl-lib:int-sub n 1)
                                                   incx)))
                         (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                       ((> j 1) nil)
                           (tagbody
                             (cond
                               ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                                (if nounit
                                    (setf (f2cl-lib:fref x-%data%
                                                         (jx)
                                                         ((1 *))
                                                         x-%offset%)
                                                  (/
                                                   (f2cl-lib:fref x-%data%
                                                                  (jx)
                                                                  ((1 *))
                                                                  x-%offset%)
                                                   (f2cl-lib:fref a-%data%
                                                                  (j j)
                                                                  ((1 lda) (1 *))
                                                                  a-%offset%))))
                                (setf temp
                                        (f2cl-lib:fref x-%data%
                                                       (jx)
                                                       ((1 *))
                                                       x-%offset%))
                                (setf ix jx)
                                (f2cl-lib:fdo (i
                                               (f2cl-lib:int-add j
                                                                 (f2cl-lib:int-sub 1))
                                               (f2cl-lib:int-add i
                                                                 (f2cl-lib:int-sub 1)))
                                              ((> i 1) nil)
                                  (tagbody
                                    (setf ix (f2cl-lib:int-sub ix incx))
```

```
                    (setf (f2cl-lib:fref x-%data%
                                         (ix)
                                         ((1 *))
                                         x-%offset%)
                          (-
                           (f2cl-lib:fref x-%data%
                                          (ix)
                                          ((1 *))
                                          x-%offset%)
                           (* temp
                              (f2cl-lib:fref a-%data%
                                             (i j)
                                             ((1 lda) (1 *))
                                             a-%offset%)))))))
          (setf jx (f2cl-lib:int-sub jx incx)))))))
(t
 (cond
   ((= incx 1)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  ((> j n) nil)
      (tagbody
        (cond
          ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
           (if nounit
               (setf (f2cl-lib:fref x-%data%
                                    (j)
                                    ((1 *))
                                    x-%offset%)
                     (/
                      (f2cl-lib:fref x-%data%
                                     (j)
                                     ((1 *))
                                     x-%offset%)
                      (f2cl-lib:fref a-%data%
                                     (j j)
                                     ((1 lda) (1 *))
                                     a-%offset%))))
           (setf temp
                   (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
           (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                            (f2cl-lib:int-add i 1))
                         ((> i n) nil)
             (tagbody
               (setf (f2cl-lib:fref x-%data%
                                    (i)
                                    ((1 *))
```

```
                                        x-%offset%)
                                    (-
                                     (f2cl-lib:fref x-%data%
                                                    (i)
                                                    ((1 *))
                                                    x-%offset%)
                                     (* temp
                                        (f2cl-lib:fref a-%data%
                                                       (i j)
                                                       ((1 lda) (1 *))
                                                       a-%offset%)))))))))))
                    (t
                     (setf jx kx)
                     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                   ((> j n) nil)
                       (tagbody
                         (cond
                           ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                            (if nounit
                                (setf (f2cl-lib:fref x-%data%
                                                     (jx)
                                                     ((1 *))
                                                     x-%offset%)
                                      (/
                                       (f2cl-lib:fref x-%data%
                                                      (jx)
                                                      ((1 *))
                                                      x-%offset%)
                                       (f2cl-lib:fref a-%data%
                                                      (j j)
                                                      ((1 lda) (1 *))
                                                      a-%offset%))))
                            (setf temp
                                    (f2cl-lib:fref x-%data%
                                                   (jx)
                                                   ((1 *))
                                                   x-%offset%))
                            (setf ix jx)
                            (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                             (f2cl-lib:int-add i 1))
                                          ((> i n) nil)
                              (tagbody
                                (setf ix (f2cl-lib:int-add ix incx))
                                (setf (f2cl-lib:fref x-%data%
                                                     (ix)
                                                     ((1 *))
```

```
                                        x-%offset%)
                                (-
                                 (f2cl-lib:fref x-%data%
                                                (ix)
                                                ((1 *))
                                                x-%offset%)
                                 (* temp
                                    (f2cl-lib:fref a-%data%
                                                   (i j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%))))))))
                    (setf jx (f2cl-lib:int-add jx incx)))))))))
           (t
            (cond
              ((char-equal uplo #\U)
               (cond
                 ((= incx 1)
                  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                ((> j n) nil)
                    (tagbody
                      (setf temp
                              (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                      (cond
                        (noconj
                         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                       ((> i
                                           (f2cl-lib:int-add j
                                                             (f2cl-lib:int-sub
                                                              1)))
                                        nil)
                           (tagbody
                             (setf temp
                                     (- temp
                                        (*
                                         (f2cl-lib:fref a-%data%
                                                        (i j)
                                                        ((1 lda) (1 *))
                                                        a-%offset%)
                                         (f2cl-lib:fref x-%data%
                                                        (i)
                                                        ((1 *))
                                                        x-%offset%))))))
                         (if nounit
                             (setf temp
                                     (/ temp
                                        (f2cl-lib:fref a-%data%
```

```
                                          (j j)
                                          ((1 lda) (1 *))
                                          a-%offset%)))))
                 (t
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i
                                    (f2cl-lib:int-add j
                                                      (f2cl-lib:int-sub
                                                       1)))
                                 nil)
                    (tagbody
                      (setf temp
                              (- temp
                                 (*
                                  (f2cl-lib:dconjg
                                   (f2cl-lib:fref a-%data%
                                                  (i j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))
                                  (f2cl-lib:fref x-%data%
                                                 (i)
                                                 ((1 *))
                                                 x-%offset%))))))
                  (if nounit
                      (setf temp
                              (/ temp
                                 (f2cl-lib:dconjg
                                  (f2cl-lib:fref a-%data%
                                                 (j j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%)))))))
               (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                     temp))))
          (t
           (setf jx kx)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf ix kx)
               (setf temp
                       (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
               (cond
                 (noconj
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i
                                    (f2cl-lib:int-add j
```

```
                                                (f2cl-lib:int-sub
                                                 1)))
                        nil)
                   (tagbody
                     (setf temp
                             (- temp
                                (*
                                 (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%)
                                 (f2cl-lib:fref x-%data%
                                                (ix)
                                                ((1 *))
                                                x-%offset%))))
                     (setf ix (f2cl-lib:int-add ix incx))))
                  (if nounit
                      (setf temp
                              (/ temp
                                 (f2cl-lib:fref a-%data%
                                                (j j)
                                                ((1 lda) (1 *))
                                                a-%offset%)))))
                 (t
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i
                                    (f2cl-lib:int-add j
                                                      (f2cl-lib:int-sub
                                                       1)))
                                 nil)
                   (tagbody
                     (setf temp
                             (- temp
                                (*
                                 (f2cl-lib:dconjg
                                  (f2cl-lib:fref a-%data%
                                                 (i j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))
                                 (f2cl-lib:fref x-%data%
                                                (ix)
                                                ((1 *))
                                                x-%offset%))))
                     (setf ix (f2cl-lib:int-add ix incx))))
                  (if nounit
                      (setf temp
```

```
                                (/ temp
                                   (f2cl-lib:dconjg
                                    (f2cl-lib:fref a-%data%
                                                   (j j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%)))))))
                        (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                                temp)
                        (setf jx (f2cl-lib:int-add jx incx)))))))
                (t
                 (cond
                   ((= incx 1)
                    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                  ((> j 1) nil)
                      (tagbody
                        (setf temp
                                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                        (cond
                          (noconj
                           (f2cl-lib:fdo (i n
                                          (f2cl-lib:int-add i
                                                            (f2cl-lib:int-sub 1)))
                                         ((> i (f2cl-lib:int-add j 1)) nil)
                             (tagbody
                               (setf temp
                                       (- temp
                                          (*
                                           (f2cl-lib:fref a-%data%
                                                          (i j)
                                                          ((1 lda) (1 *))
                                                          a-%offset%)
                                           (f2cl-lib:fref x-%data%
                                                          (i)
                                                          ((1 *))
                                                          x-%offset%))))))
                           (if nounit
                               (setf temp
                                       (/ temp
                                          (f2cl-lib:fref a-%data%
                                                         (j j)
                                                         ((1 lda) (1 *))
                                                         a-%offset%)))))
                          (t
                           (f2cl-lib:fdo (i n
                                          (f2cl-lib:int-add i
                                                            (f2cl-lib:int-sub 1)))
```

```
                                ((> i (f2cl-lib:int-add j 1)) nil)
                      (tagbody
                        (setf temp
                                (- temp
                                   (*
                                    (f2cl-lib:dconjg
                                     (f2cl-lib:fref a-%data%
                                                    (i j)
                                                    ((1 lda) (1 *))
                                                    a-%offset%))
                                     (f2cl-lib:fref x-%data%
                                                    (i)
                                                    ((1 *))
                                                    x-%offset%))))))
                     (if nounit
                         (setf temp
                                 (/ temp
                                    (f2cl-lib:dconjg
                                     (f2cl-lib:fref a-%data%
                                                    (j j)
                                                    ((1 lda) (1 *))
                                                    a-%offset%)))))))
                 (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                         temp))))
           (t
            (setf kx
                    (f2cl-lib:int-add kx
                                      (f2cl-lib:int-mul
                                       (f2cl-lib:int-sub n 1)
                                       incx)))
            (setf jx kx)
            (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                          ((> j 1) nil)
              (tagbody
                (setf ix kx)
                (setf temp
                        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
                (cond
                  (noconj
                   (f2cl-lib:fdo (i n
                                  (f2cl-lib:int-add i
                                                    (f2cl-lib:int-sub 1)))
                                 ((> i (f2cl-lib:int-add j 1)) nil)
                     (tagbody
                       (setf temp
                               (- temp
```

```
                    (*
                     (f2cl-lib:fref a-%data%
                                    (i j)
                                    ((1 lda) (1 *))
                                    a-%offset%)
                     (f2cl-lib:fref x-%data%
                                    (ix)
                                    ((1 *))
                                    x-%offset%))))
                (setf ix (f2cl-lib:int-sub ix incx))))
           (if nounit
               (setf temp
                       (/ temp
                          (f2cl-lib:fref a-%data%
                                         (j j)
                                         ((1 lda) (1 *))
                                         a-%offset%)))))
          (t
           (f2cl-lib:fdo (i n
                          (f2cl-lib:int-add i
                                            (f2cl-lib:int-sub 1)))
                         ((> i (f2cl-lib:int-add j 1)) nil)
             (tagbody
               (setf temp
                       (- temp
                          (*
                           (f2cl-lib:dconjg
                            (f2cl-lib:fref a-%data%
                                           (i j)
                                           ((1 lda) (1 *))
                                           a-%offset%))
                           (f2cl-lib:fref x-%data%
                                          (ix)
                                          ((1 *))
                                          x-%offset%))))
               (setf ix (f2cl-lib:int-sub ix incx))))
           (if nounit
               (setf temp
                       (/ temp
                          (f2cl-lib:dconjg
                           (f2cl-lib:fref a-%data%
                                          (j j)
                                          ((1 lda) (1 *))
                                          a-%offset%)))))))
    (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
            temp)
```

```
                    (setf jx (f2cl-lib:int-sub jx incx)))))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil))))))
```

Chapter 6

BLAS Level 3

6.1 dgemm BLAS

⟨*dgemm.input*⟩≡

```
)set break resume
)sys rm -f dgemm.output
)spool dgemm.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgemm.help*⟩≡

```
==========================================================================
dgemm examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

 NAME
      DGEMM - perform one of the matrix-matrix operations   C :=
      alpha*op( A )*op( B ) + beta*C,

 SYNOPSIS
      SUBROUTINE DGEMM ( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA,
                         B, LDB, BETA, C, LDC )

          CHARACTER*1  TRANSA, TRANSB

          INTEGER      M, N, K, LDA, LDB, LDC

          DOUBLE       PRECISION ALPHA, BETA

          DOUBLE       PRECISION A( LDA, * ), B( LDB, * ), C( LDC,
                       * )

 PURPOSE
      DGEMM  performs one of the matrix-matrix operations

      where  op( X ) is one of

         op( X ) = X   or   op( X ) = X',

      alpha and beta are scalars, and A, B and C are matrices,
      with op( A ) an m by k matrix,  op( B )  a  k by n matrix
      and  C an m by n matrix.

 PARAMETERS
      TRANSA - CHARACTER*1.  On entry, TRANSA specifies the form
      of op( A ) to be used in the matrix multiplication as fol-
      lows:

      TRANSA = 'N' or 'n',  op( A ) = A.

      TRANSA = 'T' or 't',  op( A ) = A'.
```

```
        TRANSA = 'C' or 'c',  op( A ) = A'.

        Unchanged on exit.

        TRANSB - CHARACTER*1.  On entry, TRANSB specifies the form
        of op( B ) to be used in the matrix multiplication as fol-
        lows:

        TRANSB = 'N' or 'n',  op( B ) = B.

        TRANSB = 'T' or 't',  op( B ) = B'.

        TRANSB = 'C' or 'c',  op( B ) = B'.

        Unchanged on exit.

        M      - INTEGER.
                 On entry,  M  specifies  the number  of rows  of the
                 matrix op( A )  and of the  matrix  C.  M  must  be
                 at least  zero.  Unchanged on exit.

        N      - INTEGER.
                 On entry,  N  specifies the number  of columns of the
                 matrix op( B ) and the number of columns of the
                 matrix C. N must be at least zero.  Unchanged on
                 exit.

        K      - INTEGER.
                 On entry,  K  specifies  the number of columns of the
                 matrix op( A ) and the number of rows of the matrix
                 op( B ). K must be at least  zero.  Unchanged on
                 exit.

        ALPHA  - DOUBLE PRECISION.
                 On entry, ALPHA specifies the scalar alpha.
                 Unchanged on exit.

   ka is
        A      -
                  DOUBLE PRECISION array of DIMENSION ( LDA, ka ), where
                 k  when  TRANSA = 'N' or 'n',  and is  m  otherwise.
                 Before entry with  TRANSA = 'N' or 'n',  the leading
                 m by k part of the array  A  must contain the matrix
                 A,  otherwise the leading  k by m  part of the array
                 A  must contain  the matrix A.  Unchanged on exit.
```

```
        LDA    - INTEGER.
               On entry, LDA specifies the first dimension of A as
               declared in the calling (sub) program. When  TRANSA =
               'N' or 'n' then LDA must be at least  max( 1, m ),
               otherwise  LDA must be at least  max( 1, k ).
               Unchanged on exit.

  kb is
        B      -
               DOUBLE PRECISION array of DIMENSION ( LDB, kb ), where
               n  when  TRANSB = 'N' or 'n',  and is  k  otherwise.
               Before entry with  TRANSB = 'N' or 'n',  the leading
               k by n part of the array  B  must contain the matrix
               B,  otherwise the leading  n by k  part of the array
               B  must contain  the matrix B.  Unchanged on exit.

        LDB    - INTEGER.
               On entry, LDB specifies the first dimension of B as
               declared in the calling (sub) program. When  TRANSB =
               'N' or 'n' then LDB must be at least  max( 1, k ),

               otherwise  LDB must be at least  max( 1, n ).
               Unchanged on exit.

        BETA   - DOUBLE PRECISION.
               On entry,  BETA  specifies the scalar  beta.  When
               BETA  is supplied as zero then C need not be set on
               input.  Unchanged on exit.

        C      - DOUBLE PRECISION array of DIMENSION ( LDC, n ).
               Before entry, the leading  m by n  part of the array
               C must contain the matrix  C,  except when  beta  is
               zero, in which case C need not be set on entry.  On
               exit, the array  C  is overwritten by the  m by n
               matrix ( alpha*op( A )*op( B ) + beta*C ).

        LDC    - INTEGER.
               On entry, LDC specifies the first dimension of C as
               declared in  the  calling  (sub)  program.   LDC
               must  be  at  least max( 1, m ).  Unchanged on exit.
```

⟨*BLAS 3 dgemm*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dgemm (transa transb m n k alpha a lda b ldb$ beta c ldc)
    (declare (type (simple-array double-float (*)) c b a)
             (type (double-float) beta alpha)
             (type fixnum ldc ldb$ lda k n m)
             (type character transb transa))
    (f2cl-lib:with-multi-array-data
        ((transa character transa-%data% transa-%offset%)
         (transb character transb-%data% transb-%offset%)
         (a double-float a-%data% a-%offset%)
         (b double-float b-%data% b-%offset%)
         (c double-float c-%data% c-%offset%))
      (prog ((temp 0.0) (i 0) (info 0) (j 0) (l 0) (ncola 0) (nrowa 0)
             (nrowb 0) (nota nil) (notb nil))
        (declare (type (double-float) temp)
                 (type fixnum i info j l ncola nrowa nrowb)
                 (type (member t nil) nota notb))
        (setf nota (char-equal transa #\N))
        (setf notb (char-equal transb #\N))
        (cond
          (nota
           (setf nrowa m)
           (setf ncola k))
          (t
           (setf nrowa k)
           (setf ncola m)))
        (cond
          (notb
           (setf nrowb k))
          (t
           (setf nrowb n)))
        (setf info 0)
        (cond
          ((and (not nota) (not (char-equal transa #\C)) (not (char-equal transa #\T)))
           (setf info 1))
          ((and (not notb) (not (char-equal transb #\C)) (not (char-equal transb #\T)))
           (setf info 2))
          ((< m 0)
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((< k 0)
           (setf info 5))
```

```
  ((< lda (max (the fixnum 1) (the fixnum nrowa)))
   (setf info 8))
  ((< ldb$
      (max (the fixnum 1) (the fixnum nrowb)))
   (setf info 10))
  ((< ldc (max (the fixnum 1) (the fixnum m)))
   (setf info 13)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DGEMM" info)
   (go end_label)))
(if (or (= m 0) (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
    (go end_label))
(cond
  ((= alpha zero)
   (cond
     ((= beta zero)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                    zero))))))
     (t
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                    (* beta
                       (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)))))))))
```

```
   (go end_label)))
(cond
  (notb
   (cond
     (nota
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (cond
            ((= beta zero)
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i m) nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       zero))))
            ((/= beta one)
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i m) nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       (* beta
                          (f2cl-lib:fref c-%data%
                                         (i j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)))))))
          (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                        ((> l k) nil)
            (tagbody
              (cond
                ((/= (f2cl-lib:fref b (l j) ((1 ldb$) (1 *))) zero)
                 (setf temp
                         (* alpha
                            (f2cl-lib:fref b-%data%
                                           (l j)
                                           ((1 ldb$) (1 *))
                                           b-%offset%)))
                 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                               ((> i m) nil)
                   (tagbody
                     (setf (f2cl-lib:fref c-%data%
```

```
                                        (i j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                                    (+
                                     (f2cl-lib:fref c-%data%
                                                    (i j)
                                                    ((1 ldc) (1 *))
                                                    c-%offset%)
                                     (* temp
                                        (f2cl-lib:fref a-%data%
                                                       (i l)
                                                       ((1 lda) (1 *))
                                                       a-%offset%)))))))))))))
                (t
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i m) nil)
                       (tagbody
                         (setf temp zero)
                         (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                       ((> l k) nil)
                           (tagbody
                             (setf temp
                                     (+ temp
                                        (*
                                         (f2cl-lib:fref a-%data%
                                                        (l i)
                                                        ((1 lda) (1 *))
                                                        a-%offset%)
                                         (f2cl-lib:fref b-%data%
                                                        (l j)
                                                        ((1 ldb$) (1 *))
                                                        b-%offset%))))))
                         (cond
                           ((= beta zero)
                            (setf (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (* alpha temp)))
                           (t
                            (setf (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
```

```
                                        c-%offset%)
                                (+ (* alpha temp)
                                   (* beta
                                      (f2cl-lib:fref c-%data%
                                                     (i j)
                                                     ((1 ldc) (1 *))
                                                     c-%offset%)))))))))))))
          (t
           (cond
             (nota
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (cond
                    ((= beta zero)
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i m) nil)
                       (tagbody
                         (setf (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                               zero))))
                    ((/= beta one)
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i m) nil)
                       (tagbody
                         (setf (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                               (* beta
                                  (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)))))))
                  (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                ((> l k) nil)
                    (tagbody
                      (cond
                        ((/= (f2cl-lib:fref b (j l) ((1 ldb$) (1 *))) zero)
                         (setf temp
                                 (* alpha
                                    (f2cl-lib:fref b-%data%
                                                   (j l)
                                                   ((1 ldb$) (1 *))
```

```
                                        b-%offset%)))
                    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                  ((> i m) nil)
                      (tagbody
                        (setf (f2cl-lib:fref c-%data%
                                             (i j)
                                             ((1 ldc) (1 *))
                                             c-%offset%)
                              (+
                               (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                               (* temp
                                  (f2cl-lib:fref a-%data%
                                                 (i l)
                                                 ((1 lda) (1 *))
                                                 a-%offset%)))))))))
(t
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j n) nil)
   (tagbody
     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                   ((> i m) nil)
       (tagbody
         (setf temp zero)
         (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                       ((> l k) nil)
           (tagbody
             (setf temp
                     (+ temp
                        (*
                         (f2cl-lib:fref a-%data%
                                        (l i)
                                        ((1 lda) (1 *))
                                        a-%offset%)
                         (f2cl-lib:fref b-%data%
                                        (j l)
                                        ((1 ldb$) (1 *))
                                        b-%offset%))))))
         (cond
           ((= beta zero)
            (setf (f2cl-lib:fref c-%data%
                                 (i j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
```

```
                                (* alpha temp)))
                           (t
                            (setf (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (+ (* alpha temp)
                                     (* beta
                                        (f2cl-lib:fref c-%data%
                                                       (i j)
                                                       ((1 ldc) (1 *))
                                                       c-%offset%)))))))))))))
   end_label
          (return
           (values nil nil nil nil nil nil nil nil nil nil nil nil nil))))))
```

6.2 dsymm BLAS

⟨*dsymm.input*⟩≡

```
  )set break resume
  )sys rm -f dsymm.output
  )spool dsymm.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dsymm.help*⟩≡

```
=====================================================================
dsymm examples
=====================================================================

=====================================================================
Man Page Details
=====================================================================

 NAME
      DSYMM - perform one of the matrix-matrix operations   C :=
      alpha*A*B + beta*C,

 SYNOPSIS
      SUBROUTINE DSYMM ( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB,
                         BETA, C, LDC )

          CHARACTER*1  SIDE, UPLO

          INTEGER      M, N, LDA, LDB, LDC

          DOUBLE       PRECISION ALPHA, BETA

          DOUBLE       PRECISION A( LDA, * ), B( LDB, * ), C( LDC,
                       * )

 PURPOSE
      DSYMM  performs one of the matrix-matrix operations

      or

         C := alpha*B*A + beta*C,

      where alpha and beta are scalars,  A is a symmetric matrix
      and  B and C are  m by n matrices.

 PARAMETERS
      SIDE   - CHARACTER*1.
             On entry,  SIDE  specifies whether  the  symmetric
             matrix  A appears on the  left or right  in the
             operation as follows:

             SIDE = 'L' or 'l'   C := alpha*A*B + beta*C,

             SIDE = 'R' or 'r'   C := alpha*B*A + beta*C,
```

```
              Unchanged on exit.

       UPLO   - CHARACTER*1.
              On entry,  UPLO specifies whether the upper or
              lower triangular part of the symmetric matrix
              A is to be referenced as follows:

              UPLO = 'U' or 'u'   Only the upper triangular part of
              the symmetric matrix is to be referenced.

              UPLO = 'L' or 'l'   Only the lower triangular part of
              the symmetric matrix is to be referenced.

              Unchanged on exit.

       M      - INTEGER.
              On entry,  M specifies the number of rows of the
              matrix  C.  M  must be at least zero.  Unchanged on
              exit.

       N      - INTEGER.
              On entry, N specifies the number of columns of the
              matrix C.  N  must be at least zero.  Unchanged on
              exit.

       ALPHA  - DOUBLE PRECISION.
              On entry, ALPHA specifies the scalar alpha.
              Unchanged on exit.

  ka is
       A      -
               DOUBLE PRECISION array of DIMENSION ( LDA, ka ), where
              m  when  SIDE = 'L' or 'l'  and is  n otherwise.
              Before entry  with  SIDE = 'L' or 'l',  the  m by m
              part of the array  A  must contain the  symmetric
              matrix,  such that when  UPLO = 'U' or 'u', the lead-
              ing m by m upper triangular part of the array  A
              must contain the upper triangular part of the  sym-
              metric matrix and the  strictly  lower triangular
              part of  A  is not referenced,  and when  UPLO = 'L'
              or 'l', the leading  m by m  lower triangular part
              of the  array  A must  contain  the  lower triangular
              part  of the  symmetric matrix and the  strictly
              upper triangular part of  A  is not referenced.
              Before entry  with  SIDE = 'R' or 'r',  the  n by n
              part of the array  A  must contain the  symmetric
```

```
          matrix,  such that when  UPLO = 'U' or 'u', the lead-
          ing n by n upper triangular part of the array  A
          must contain the upper triangular part of the  sym-
          metric matrix and the  strictly  lower triangular
          part of  A  is not referenced,  and when  UPLO = 'L'
          or 'l', the leading  n by n  lower triangular part
          of the  array  A must  contain  the  lower triangular
          part  of the  symmetric matrix and the  strictly
          upper triangular part of  A  is not referenced.
          Unchanged on exit.

LDA     - INTEGER.
          On entry, LDA specifies the first dimension of A as
          declared in the calling (sub) program.  When  SIDE =
          'L' or 'l'  then LDA must be at least  max( 1, m ),
          otherwise  LDA must be at least  max( 1, n ).
          Unchanged on exit.

B       - DOUBLE PRECISION array of DIMENSION ( LDB, n ).
          Before entry, the leading  m by n part of the array
          B  must contain the matrix B.  Unchanged on exit.

LDB     - INTEGER.
          On entry, LDB specifies the first dimension of B as
          declared in  the  calling  (sub)  program.   LDB
          must  be  at  least max( 1, m ).  Unchanged on exit.

BETA    - DOUBLE PRECISION.
          On entry,  BETA  specifies the scalar  beta.  When
          BETA  is supplied as zero then C need not be set on
          input.  Unchanged on exit.

C       - DOUBLE PRECISION array of DIMENSION ( LDC, n ).
          Before entry, the leading  m by n  part of the array
          C must contain the matrix  C,  except when  beta  is
          zero, in which case C need not be set on entry.  On
          exit, the array  C  is overwritten by the  m by n
          updated matrix.

LDC     - INTEGER.
          On entry, LDC specifies the first dimension of C as
          declared in  the  calling  (sub)  program.   LDC
          must  be  at  least max( 1, m ).  Unchanged on exit.

          Level 3 Blas routine.
```

⟨*BLAS 3 dsymm*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dsymm (side uplo m n alpha a lda b ldb$ beta c ldc)
    (declare (type (simple-array double-float (*)) c b a)
             (type (double-float) beta alpha)
             (type fixnum ldc ldb$ lda n m)
             (type character uplo side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (uplo character uplo-%data% uplo-%offset%)
         (a double-float a-%data% a-%offset%)
         (b double-float b-%data% b-%offset%)
         (c double-float c-%data% c-%offset%))
      (prog ((temp1 0.0) (temp2 0.0) (i 0) (info 0) (j 0) (k 0) (nrowa 0)
             (upper nil))
        (declare (type (double-float) temp1 temp2)
                 (type fixnum i info j k nrowa)
                 (type (member t nil) upper))
        (cond
          ((char-equal side #\L)
           (setf nrowa m))
          (t
           (setf nrowa n)))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not (char-equal side #\L)) (not (char-equal side #\R)))
           (setf info 1))
          ((and (not upper) (not (char-equal uplo #\L)))
           (setf info 2))
          ((< m 0)
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))
           (setf info 7))
          ((< ldb$ (max (the fixnum 1) (the fixnum m)))
           (setf info 9))
          ((< ldc (max (the fixnum 1) (the fixnum m)))
           (setf info 12)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
```

```
    "DSYMM" info)
   (go end_label)))
(if (or (= m 0) (= n 0) (and (= alpha zero) (= beta one)))
    (go end_label))
(cond
  ((= alpha zero)
   (cond
     ((= beta zero)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                    zero))))))
     (t
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                    (* beta
                       (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)))))))))
   (go end_label)))
(cond
  ((char-equal side #\L)
   (cond
     (upper
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
```

```
(setf temp1
        (* alpha
           (f2cl-lib:fref b-%data%
                          (i j)
                          ((1 ldb$) (1 *))
                          b-%offset%)))
(setf temp2 zero)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k
                  (f2cl-lib:int-add i
                                    (f2cl-lib:int-sub
                                     1)))
               nil)
  (tagbody
    (setf (f2cl-lib:fref c-%data%
                         (k j)
                         ((1 ldc) (1 *))
                         c-%offset%)
          (+
           (f2cl-lib:fref c-%data%
                          (k j)
                          ((1 ldc) (1 *))
                          c-%offset%)
           (* temp1
              (f2cl-lib:fref a-%data%
                             (k i)
                             ((1 lda) (1 *))
                             a-%offset%))))
    (setf temp2
            (+ temp2
               (*
                (f2cl-lib:fref b-%data%
                               (k j)
                               ((1 ldb$) (1 *))
                               b-%offset%)
                (f2cl-lib:fref a-%data%
                               (k i)
                               ((1 lda) (1 *))
                               a-%offset%))))))
(cond
  ((= beta zero)
   (setf (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%)
         (+
```

```
                                (* temp1
                                   (f2cl-lib:fref a-%data%
                                                  (i i)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))
                                (* alpha temp2))))
                          (t
                           (setf (f2cl-lib:fref c-%data%
                                                (i j)
                                                ((1 ldc) (1 *))
                                                c-%offset%)
                                 (+
                                  (* beta
                                     (f2cl-lib:fref c-%data%
                                                    (i j)
                                                    ((1 ldc) (1 *))
                                                    c-%offset%))
                                  (* temp1
                                     (f2cl-lib:fref a-%data%
                                                    (i i)
                                                    ((1 lda) (1 *))
                                                    a-%offset%))
                                  (* alpha temp2)))))))))
                (t
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (f2cl-lib:fdo (i m (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                                   ((> i 1) nil)
                       (tagbody
                         (setf temp1
                                 (* alpha
                                    (f2cl-lib:fref b-%data%
                                                   (i j)
                                                   ((1 ldb$) (1 *))
                                                   b-%offset%)))
                         (setf temp2 zero)
                         (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
                                          (f2cl-lib:int-add k 1))
                                       ((> k m) nil)
                           (tagbody
                             (setf (f2cl-lib:fref c-%data%
                                                  (k j)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)
                                   (+
```

```
                    (f2cl-lib:fref c-%data%
                                  (k j)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                    (* temp1
                       (f2cl-lib:fref a-%data%
                                      (k i)
                                      ((1 lda) (1 *))
                                      a-%offset%))))
      (setf temp2
              (+ temp2
                 (*
                  (f2cl-lib:fref b-%data%
                                 (k j)
                                 ((1 ldb$) (1 *))
                                 b-%offset%)
                  (f2cl-lib:fref a-%data%
                                 (k i)
                                 ((1 lda) (1 *))
                                 a-%offset%))))))
  (cond
    ((= beta zero)
     (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%)
             (+
              (* temp1
                 (f2cl-lib:fref a-%data%
                                (i i)
                                ((1 lda) (1 *))
                                a-%offset%))
              (* alpha temp2))))
    (t
     (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%)
             (+
              (* beta
                 (f2cl-lib:fref c-%data%
                                (i j)
                                ((1 ldc) (1 *))
                                c-%offset%))
              (* temp1
                 (f2cl-lib:fref a-%data%
```

```
                                                  (i i)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))
                                (* alpha temp2))))))))))))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf temp1
                       (* alpha
                          (f2cl-lib:fref a-%data%
                                         (j j)
                                         ((1 lda) (1 *))
                                         a-%offset%)))
               (cond
                 ((= beta zero)
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i m) nil)
                    (tagbody
                      (setf (f2cl-lib:fref c-%data%
                                           (i j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                            (* temp1
                               (f2cl-lib:fref b-%data%
                                              (i j)
                                              ((1 ldb$) (1 *))
                                              b-%offset%))))))
                 (t
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i m) nil)
                    (tagbody
                      (setf (f2cl-lib:fref c-%data%
                                           (i j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                            (+
                             (* beta
                                (f2cl-lib:fref c-%data%
                                               (i j)
                                               ((1 ldc) (1 *))
                                               c-%offset%))
                             (* temp1
                                (f2cl-lib:fref b-%data%
                                               (i j)
                                               ((1 ldb$) (1 *))
```

```
                                        b-%offset%))))))))
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
               nil)
  (tagbody
    (cond
      (upper
       (setf temp1
               (* alpha
                  (f2cl-lib:fref a-%data%
                                 (k j)
                                 ((1 lda) (1 *))
                                 a-%offset%))))
      (t
       (setf temp1
               (* alpha
                  (f2cl-lib:fref a-%data%
                                 (j k)
                                 ((1 lda) (1 *))
                                 a-%offset%)))))
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i m) nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data%
                             (i j)
                             ((1 ldc) (1 *))
                             c-%offset%)
                (+
                 (f2cl-lib:fref c-%data%
                                (i j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* temp1
                    (f2cl-lib:fref b-%data%
                                   (i k)
                                   ((1 ldb$) (1 *))
                                   b-%offset%))))))))
(f2cl-lib:fdo (k (f2cl-lib:int-add j 1) (f2cl-lib:int-add k 1))
              ((> k n) nil)
  (tagbody
    (cond
      (upper
       (setf temp1
               (* alpha
                  (f2cl-lib:fref a-%data%
                                 (j k)
```

```
                                                ((1 lda) (1 *))
                                                a-%offset%))))
                        (t
                         (setf temp1
                                 (* alpha
                                    (f2cl-lib:fref a-%data%
                                                   (k j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%)))))
                      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                    ((> i m) nil)
                        (tagbody
                          (setf (f2cl-lib:fref c-%data%
                                               (i j)
                                               ((1 ldc) (1 *))
                                               c-%offset%)
                                  (+
                                   (f2cl-lib:fref c-%data%
                                                  (i j)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)
                                   (* temp1
                                      (f2cl-lib:fref b-%data%
                                                     (i k)
                                                     ((1 ldb$) (1 *))
                                                     b-%offset%)))))))))
   end_label
          (return (values nil nil nil nil nil nil nil nil nil nil nil nil))))))
```

6.3 dsyr2k BLAS

⟨*dsyr2k.input*⟩≡

```
  )set break resume
  )sys rm -f dsyr2k.output
  )spool dsyr2k.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dsyr2k.help*⟩≡

```
  ======================================================================
  dsyr2k examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        DSYR2K - perform one of the symmetric rank 2k operations   C
        := alpha*A*B' + alpha*B*A' + beta*C,

   SYNOPSIS
        SUBROUTINE DSYR2K( UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB,
                           BETA, C, LDC )

            CHARACTER*1    UPLO, TRANS

            INTEGER        N, K, LDA, LDB, LDC

            DOUBLE         PRECISION ALPHA, BETA

            DOUBLE         PRECISION A( LDA, * ), B( LDB, * ), C(
                           LDC, * )

   PURPOSE
        DSYR2K  performs one of the symmetric rank 2k operations

        or

           C := alpha*A'*B + alpha*B'*A + beta*C,

        where  alpha and beta  are scalars, C is an  n by n  sym-
        metric matrix and  A and B  are  n by k  matrices  in the
        first  case  and  k by n matrices in the second case.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On  entry,   UPLO  specifies  whether  the  upper  or
                 lower triangular  part  of the  array  C  is to be
                 referenced  as follows:

                 UPLO = 'U' or 'u'   Only the  upper triangular part
                 of  C is to be referenced.
```

```
              UPLO = 'L' or 'l'   Only the  lower triangular part
              of  C is to be referenced.

              Unchanged on exit.

       TRANS  - CHARACTER*1.
              On entry,  TRANS  specifies the operation to be per-
              formed as follows:

              TRANS = 'N' or 'n'   C := alpha*A*B' + alpha*B*A' +
              beta*C.

              TRANS = 'T' or 't'   C := alpha*A'*B + alpha*B'*A +
              beta*C.

              TRANS = 'C' or 'c'   C := alpha*A'*B + alpha*B'*A +
              beta*C.

              Unchanged on exit.

       N      - INTEGER.
              On entry,  N specifies the order of the matrix C.  N
              must be at least zero.  Unchanged on exit.

       K      - INTEGER.
              On entry with  TRANS = 'N' or 'n',  K  specifies  the
              number of  columns  of the  matrices  A and B,  and
              on  entry  with TRANS = 'T' or 't' or 'C' or 'c',  K
              specifies  the  number of rows of the matrices  A and
              B.  K must be at least  zero.  Unchanged on exit.

       ALPHA  - DOUBLE PRECISION.
              On entry, ALPHA specifies the scalar alpha.
              Unchanged on exit.

  ka is
       A      -
               DOUBLE PRECISION array of DIMENSION ( LDA, ka ), where
              k  when  TRANS = 'N' or 'n',  and is  n  otherwise.
              Before entry with  TRANS = 'N' or 'n',  the  leading
              n by k part of the array  A  must contain the matrix
              A,  otherwise the leading  k by n  part of the array
              A  must contain  the matrix A.  Unchanged on exit.

       LDA    - INTEGER.
              On entry, LDA specifies the first dimension of A as
```

declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k). Unchanged on exit.

kb is

B - DOUBLE PRECISION array of DIMENSION (LDB, kb), where k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array B must contain the matrix B, otherwise the leading k by n part of the array B must contain the matrix B. Unchanged on exit.

LDB - INTEGER.
On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDB must be at least max(1, n), otherwise LDB must be at least max(1, k). Unchanged on exit.

BETA - DOUBLE PRECISION.
On entry, BETA specifies the scalar beta. Unchanged on exit.

C - DOUBLE PRECISION array of DIMENSION (LDC, n).
Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

LDC - INTEGER.
On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n). Unchanged on exit.

⟨*BLAS 3 dsyr2k*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dsyr2k (uplo trans n k alpha a lda b ldb$ beta c ldc)
    (declare (type (simple-array double-float (*)) c b a)
             (type (double-float) beta alpha)
             (type fixnum ldc ldb$ lda k n)
             (type character trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (a double-float a-%data% a-%offset%)
         (b double-float b-%data% b-%offset%)
         (c double-float c-%data% c-%offset%))
      (prog ((temp1 0.0) (temp2 0.0) (i 0) (info 0) (j 0) (l 0) (nrowa 0)
             (upper nil))
        (declare (type (double-float) temp1 temp2)
                 (type fixnum i info j l nrowa)
                 (type (member t nil) upper))
        (cond
          ((char-equal trans #\N)
           (setf nrowa n))
          (t
           (setf nrowa k)))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not upper) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 2))
          ((< n 0)
           (setf info 3))
          ((< k 0)
           (setf info 4))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))
           (setf info 7))
          ((< ldb$
              (max (the fixnum 1) (the fixnum nrowa)))
           (setf info 9))
          ((< ldc (max (the fixnum 1) (the fixnum n)))
           (setf info 12)))
        (cond
```

```
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DSYR2K" info)
   (go end_label)))
(if (or (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
    (go end_label))
(cond
  ((= alpha zero)
   (cond
     (upper
      (cond
        ((= beta zero)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i j) nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       zero))))))
        (t
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i j) nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       (* beta
                          (f2cl-lib:fref c-%data%
                                         (i j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)))))))))))
     (t
      (cond
        ((= beta zero)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
```

```
                    (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                  ((> i n) nil)
                      (tagbody
                        (setf (f2cl-lib:fref c-%data%
                                             (i j)
                                             ((1 ldc) (1 *))
                                             c-%offset%)
                                      zero))))))
                  (t
                   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                 ((> j n) nil)
                     (tagbody
                       (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                     ((> i n) nil)
                         (tagbody
                           (setf (f2cl-lib:fref c-%data%
                                                (i j)
                                                ((1 ldc) (1 *))
                                                c-%offset%)
                                 (* beta
                                    (f2cl-lib:fref c-%data%
                                                   (i j)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%)))))))))
             (go end_label)))
          (cond
            ((char-equal trans #\N)
             (cond
               (upper
                (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                              ((> j n) nil)
                  (tagbody
                    (cond
                      ((= beta zero)
                       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                     ((> i j) nil)
                         (tagbody
                           (setf (f2cl-lib:fref c-%data%
                                                (i j)
                                                ((1 ldc) (1 *))
                                                c-%offset%)
                                 zero))))
                      ((/= beta one)
                       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                     ((> i j) nil)
                         (tagbody
```

```
                (setf (f2cl-lib:fref c-%data%
                                     (i j)
                                     ((1 ldc) (1 *))
                                     c-%offset%)
                        (* beta
                           (f2cl-lib:fref c-%data%
                                          (i j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)))))))
          (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                        ((> l k) nil)
            (tagbody
              (cond
                ((or (/= (f2cl-lib:fref a (j l) ((1 lda) (1 *))) zero)
                     (/= (f2cl-lib:fref b (j l) ((1 ldb$) (1 *))) zero))
                 (setf temp1
                         (* alpha
                            (f2cl-lib:fref b-%data%
                                           (j l)
                                           ((1 ldb$) (1 *))
                                           b-%offset%)))
                 (setf temp2
                         (* alpha
                            (f2cl-lib:fref a-%data%
                                           (j l)
                                           ((1 lda) (1 *))
                                           a-%offset%)))
                 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                               ((> i j) nil)
                   (tagbody
                     (setf (f2cl-lib:fref c-%data%
                                          (i j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                             (+
                              (f2cl-lib:fref c-%data%
                                             (i j)
                                             ((1 ldc) (1 *))
                                             c-%offset%)
                              (*
                               (f2cl-lib:fref a-%data%
                                              (i l)
                                              ((1 lda) (1 *))
                                              a-%offset%)
                               temp1)
                              (*
```

```
                                (f2cl-lib:fref b-%data%
                                               (i l)
                                               ((1 ldb$) (1 *))
                                               b-%offset%)
                                temp2))))))))))))
                (t
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (cond
                       ((= beta zero)
                        (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                      ((> i n) nil)
                          (tagbody
                            (setf (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  zero))))
                       ((/= beta one)
                        (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                      ((> i n) nil)
                          (tagbody
                            (setf (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (* beta
                                     (f2cl-lib:fref c-%data%
                                                    (i j)
                                                    ((1 ldc) (1 *))
                                                    c-%offset%)))))))
                     (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                   ((> l k) nil)
                       (tagbody
                         (cond
                           ((or (/= (f2cl-lib:fref a (j l) ((1 lda) (1 *))) zero)
                                (/= (f2cl-lib:fref b (j l) ((1 ldb$) (1 *))) zero))
                            (setf temp1
                                    (* alpha
                                       (f2cl-lib:fref b-%data%
                                                      (j l)
                                                      ((1 ldb$) (1 *))
                                                      b-%offset%)))
                            (setf temp2
                                    (* alpha
```

```
                                (f2cl-lib:fref a-%data%
                                               (j l)
                                               ((1 lda) (1 *))
                                               a-%offset%)))
                        (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                      ((> i n) nil)
                          (tagbody
                            (setf (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (+
                                   (f2cl-lib:fref c-%data%
                                                  (i j)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)
                                   (*
                                    (f2cl-lib:fref a-%data%
                                                   (i l)
                                                   ((1 lda) (1 *))
                                                   a-%offset%)
                                    temp1)
                                   (*
                                    (f2cl-lib:fref b-%data%
                                                   (i l)
                                                   ((1 ldb$) (1 *))
                                                   b-%offset%)
                                    temp2))))))))))))
          (t
           (cond
             (upper
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i j) nil)
                    (tagbody
                      (setf temp1 zero)
                      (setf temp2 zero)
                      (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                    ((> l k) nil)
                        (tagbody
                          (setf temp1
                                  (+ temp1
                                     (*
                                      (f2cl-lib:fref a-%data%
```

```
                                                (l i)
                                                ((1 lda) (1 *))
                                                a-%offset%)
                                    (f2cl-lib:fref b-%data%
                                                (l j)
                                                ((1 ldb$) (1 *))
                                                b-%offset%))))
                            (setf temp2
                                    (+ temp2
                                       (*
                                        (f2cl-lib:fref b-%data%
                                                (l i)
                                                ((1 ldb$) (1 *))
                                                b-%offset%)
                                        (f2cl-lib:fref a-%data%
                                                (l j)
                                                ((1 lda) (1 *))
                                                a-%offset%))))))
                          (cond
                            ((= beta zero)
                             (setf (f2cl-lib:fref c-%data%
                                                  (i j)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)
                                   (+ (* alpha temp1) (* alpha temp2))))
                            (t
                             (setf (f2cl-lib:fref c-%data%
                                                  (i j)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)
                                   (+
                                    (* beta
                                       (f2cl-lib:fref c-%data%
                                                      (i j)
                                                      ((1 ldc) (1 *))
                                                      c-%offset%))
                                    (* alpha temp1)
                                    (* alpha temp2)))))))))
                    (t
                     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                   ((> j n) nil)
                       (tagbody
                         (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                       ((> i n) nil)
                           (tagbody
                             (setf temp1 zero)
```

```
(setf temp2 zero)
(f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
              ((> l k) nil)
  (tagbody
    (setf temp1
            (+ temp1
               (*
                (f2cl-lib:fref a-%data%
                               (l i)
                               ((1 lda) (1 *))
                               a-%offset%)
                (f2cl-lib:fref b-%data%
                               (l j)
                               ((1 ldb$) (1 *))
                               b-%offset%))))
    (setf temp2
            (+ temp2
               (*
                (f2cl-lib:fref b-%data%
                               (l i)
                               ((1 ldb$) (1 *))
                               b-%offset%)
                (f2cl-lib:fref a-%data%
                               (l j)
                               ((1 lda) (1 *))
                               a-%offset%))))))
(cond
  ((= beta zero)
   (setf (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%)
           (+ (* alpha temp1) (* alpha temp2))))
  (t
   (setf (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%)
           (+
            (* beta
               (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%))
            (* alpha temp1)
            (* alpha temp2))))))
```

```
   end_label
          (return (values nil nil nil nil nil nil nil nil nil nil nil nil)))))
```

6.4 dsyrk BLAS

⟨*dsyrk.input*⟩≡

```
  )set break resume
  )sys rm -f dsyrk.output
  )spool dsyrk.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dsyrk.help*⟩≡

```
======================================================================
dsyrk examples
======================================================================

======================================================================
Man Page Details
======================================================================

 NAME
      DSYRK - perform one of the symmetric rank k operations   C
      := alpha*A*A' + beta*C,

 SYNOPSIS
      SUBROUTINE DSYRK ( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA,
                         C, LDC )

          CHARACTER*1  UPLO, TRANS

          INTEGER      N, K, LDA, LDC

          DOUBLE       PRECISION ALPHA, BETA

          DOUBLE       PRECISION A( LDA, * ), C( LDC, * )

 PURPOSE
      DSYRK  performs one of the symmetric rank k operations

      or

         C := alpha*A'*A + beta*C,

      where  alpha and beta  are scalars, C is an  n by n  sym-
      metric matrix and  A  is an  n by k  matrix in the first
      case and a  k by n  matrix in the second case.

 PARAMETERS
      UPLO   - CHARACTER*1.
               On  entry,  UPLO  specifies  whether  the  upper  or
               lower triangular  part  of the  array  C  is to be
               referenced  as follows:

               UPLO = 'U' or 'u'   Only the  upper triangular part
               of  C is to be referenced.

               UPLO = 'L' or 'l'   Only the  lower triangular part
```

```
              of  C is to be referenced.

              Unchanged on exit.

       TRANS  - CHARACTER*1.
              On entry,  TRANS  specifies the operation to be per-
              formed as follows:

              TRANS = 'N' or 'n'   C := alpha*A*A' + beta*C.

              TRANS = 'T' or 't'   C := alpha*A'*A + beta*C.

              TRANS = 'C' or 'c'   C := alpha*A'*A + beta*C.

              Unchanged on exit.

       N      - INTEGER.
              On entry,  N specifies the order of the matrix C.  N
              must be at least zero.  Unchanged on exit.

       K      - INTEGER.
              On entry with  TRANS = 'N' or 'n',  K  specifies  the
              number of  columns  of  the  matrix  A,  and  on
              entry  with TRANS = 'T' or 't' or 'C' or 'c',  K
              specifies  the  number of rows of the matrix  A.  K
              must be at least zero.  Unchanged on exit.

       ALPHA  - DOUBLE PRECISION.
              On entry, ALPHA specifies the scalar alpha.
              Unchanged on exit.

  ka is
       A      -
               DOUBLE PRECISION array of DIMENSION ( LDA, ka ), where
              k  when  TRANS = 'N' or 'n',  and is  n  otherwise.
              Before entry with  TRANS = 'N' or 'n',  the  leading
              n by k part of the array  A  must contain the matrix
              A,  otherwise the leading  k by n  part of the array
              A  must contain  the matrix A.  Unchanged on exit.

       LDA    - INTEGER.
              On entry, LDA specifies the first dimension of A as
              declared in  the  calling  (sub)  program.   When
              TRANS = 'N' or 'n' then  LDA must be at least  max(
              1, n ), otherwise  LDA must be at least  max( 1, k ).
              Unchanged on exit.
```

```
BETA   - DOUBLE PRECISION.
         On entry, BETA specifies the scalar beta.  Unchanged
         on exit.

C      - DOUBLE PRECISION array of DIMENSION ( LDC, n ).
         Before entry  with  UPLO = 'U' or 'u',  the leading
         n by n upper triangular part of the array C must con-
         tain the upper triangular part  of the  symmetric
         matrix  and the strictly lower triangular part of C
         is not referenced.  On exit, the upper triangular
         part of the array  C is overwritten by the upper tri-
         angular part of the updated matrix.  Before entry
         with  UPLO = 'L' or 'l',  the leading  n by n lower
         triangular part of the array C must contain the lower
         triangular part  of the  symmetric matrix  and the
         strictly upper triangular part of C is not refer-
         enced.  On exit, the lower triangular part of the
         array  C is overwritten by the lower triangular part
         of the updated matrix.

LDC    - INTEGER.
         On entry, LDC specifies the first dimension of C as
         declared in  the  calling  (sub)  program.   LDC
         must  be  at  least max( 1, n ).  Unchanged on exit.
```

⟨*BLAS 3 dsyrk*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dsyrk (uplo trans n k alpha a lda beta c ldc)
    (declare (type (simple-array double-float (*)) c a)
             (type (double-float) beta alpha)
             (type fixnum ldc lda k n)
             (type character trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (a double-float a-%data% a-%offset%)
         (c double-float c-%data% c-%offset%))
      (prog ((temp 0.0) (i 0) (info 0) (j 0) (l 0) (nrowa 0) (upper nil))
        (declare (type (double-float) temp)
                 (type fixnum i info j l nrowa)
                 (type (member t nil) upper))
        (cond
          ((char-equal trans #\N)
           (setf nrowa n))
          (t
           (setf nrowa k)))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not upper) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 2))
          ((< n 0)
           (setf info 3))
          ((< k 0)
           (setf info 4))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))
           (setf info 7))
          ((< ldc (max (the fixnum 1) (the fixnum n)))
           (setf info 10)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DSYRK" info)
           (go end_label)))
```

```
(if (or (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
    (go end_label))
(cond
  ((= alpha zero)
   (cond
     (upper
      (cond
        ((= beta zero)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i j) nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       zero))))))
        (t
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i j) nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       (* beta
                          (f2cl-lib:fref c-%data%
                                         (i j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)))))))))))
     (t
      (cond
        ((= beta zero)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                           ((> i n) nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
```

```
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                              zero))))))
                 (t
                  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                ((> j n) nil)
                    (tagbody
                      (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                    ((> i n) nil)
                        (tagbody
                          (setf (f2cl-lib:fref c-%data%
                                               (i j)
                                               ((1 ldc) (1 *))
                                               c-%offset%)
                                (* beta
                                   (f2cl-lib:fref c-%data%
                                                  (i j)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)))))))))
             (go end_label)))
          (cond
            ((char-equal trans #\N)
             (cond
               (upper
                (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                              ((> j n) nil)
                  (tagbody
                    (cond
                      ((= beta zero)
                       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                     ((> i j) nil)
                         (tagbody
                           (setf (f2cl-lib:fref c-%data%
                                                (i j)
                                                ((1 ldc) (1 *))
                                                c-%offset%)
                                 zero))))
                      ((/= beta one)
                       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                     ((> i j) nil)
                         (tagbody
                           (setf (f2cl-lib:fref c-%data%
                                                (i j)
                                                ((1 ldc) (1 *))
                                                c-%offset%)
                                 (* beta
```

```
                              (f2cl-lib:fref c-%data%
                                            (i j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)))))))
          (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                        ((> l k) nil)
            (tagbody
              (cond
                ((/= (f2cl-lib:fref a (j l) ((1 lda) (1 *))) zero)
                 (setf temp
                         (* alpha
                            (f2cl-lib:fref a-%data%
                                           (j l)
                                           ((1 lda) (1 *))
                                           a-%offset%)))
                 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                               ((> i j) nil)
                   (tagbody
                     (setf (f2cl-lib:fref c-%data%
                                          (i j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                           (+
                            (f2cl-lib:fref c-%data%
                                           (i j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                            (* temp
                               (f2cl-lib:fref a-%data%
                                              (i l)
                                              ((1 lda) (1 *))
                                              a-%offset%)))))))))))
        (t
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (cond
               ((= beta zero)
                (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                              ((> i n) nil)
                  (tagbody
                    (setf (f2cl-lib:fref c-%data%
                                         (i j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                          zero))))
```

```
                    ((/= beta one)
                     (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                   ((> i n) nil)
                       (tagbody
                         (setf (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                                 (* beta
                                    (f2cl-lib:fref c-%data%
                                                   (i j)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%)))))))
                   (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                 ((> l k) nil)
                     (tagbody
                       (cond
                         ((/= (f2cl-lib:fref a (j l) ((1 lda) (1 *))) zero)
                          (setf temp
                                  (* alpha
                                     (f2cl-lib:fref a-%data%
                                                    (j l)
                                                    ((1 lda) (1 *))
                                                    a-%offset%)))
                          (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                        ((> i n) nil)
                            (tagbody
                              (setf (f2cl-lib:fref c-%data%
                                                   (i j)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%)
                                      (+
                                       (f2cl-lib:fref c-%data%
                                                      (i j)
                                                      ((1 ldc) (1 *))
                                                      c-%offset%)
                                       (* temp
                                          (f2cl-lib:fref a-%data%
                                                         (i l)
                                                         ((1 lda) (1 *))
                                                         a-%offset%)))))))))))
                  (t
                   (cond
                     (upper
                      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                    ((> j n) nil)
```

```
      (tagbody
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i j) nil)
          (tagbody
            (setf temp zero)
            (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                          ((> l k) nil)
              (tagbody
                (setf temp
                        (+ temp
                           (*
                            (f2cl-lib:fref a-%data%
                                           (l i)
                                           ((1 lda) (1 *))
                                           a-%offset%)
                            (f2cl-lib:fref a-%data%
                                           (l j)
                                           ((1 lda) (1 *))
                                           a-%offset%))))))
            (cond
              ((= beta zero)
               (setf (f2cl-lib:fref c-%data%
                                    (i j)
                                    ((1 ldc) (1 *))
                                    c-%offset%)
                     (* alpha temp)))
              (t
               (setf (f2cl-lib:fref c-%data%
                                    (i j)
                                    ((1 ldc) (1 *))
                                    c-%offset%)
                     (+ (* alpha temp)
                        (* beta
                           (f2cl-lib:fref c-%data%
                                          (i j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)))))))))
    (t
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   ((> j n) nil)
       (tagbody
         (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                       ((> i n) nil)
           (tagbody
             (setf temp zero)
             (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
```

```
                                ((> l k) nil)
                              (tagbody
                                (setf temp
                                        (+ temp
                                           (*
                                            (f2cl-lib:fref a-%data%
                                                           (l i)
                                                           ((1 lda) (1 *))
                                                           a-%offset%)
                                            (f2cl-lib:fref a-%data%
                                                           (l j)
                                                           ((1 lda) (1 *))
                                                           a-%offset%))))))
                             (cond
                               ((= beta zero)
                                (setf (f2cl-lib:fref c-%data%
                                                     (i j)
                                                     ((1 ldc) (1 *))
                                                     c-%offset%)
                                        (* alpha temp)))
                               (t
                                (setf (f2cl-lib:fref c-%data%
                                                     (i j)
                                                     ((1 ldc) (1 *))
                                                     c-%offset%)
                                        (+ (* alpha temp)
                                           (* beta
                                              (f2cl-lib:fref c-%data%
                                                             (i j)
                                                             ((1 ldc) (1 *))
                                                             c-%offset%)))))))))))
  end_label
       (return (values nil nil nil nil nil nil nil nil nil nil))))))
```

6.5 dtrmm BLAS

⟨*dtrmm.input*⟩≡

```
)set break resume
)sys rm -f dtrmm.output
)spool dtrmm.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dtrmm.help*⟩≡

```
 ========================================================================
 dtrmm examples
 ========================================================================

 ========================================================================
 Man Page Details
 ========================================================================

  NAME
       DTRMM - perform one of the matrix-matrix operations   B :=
       alpha*op( A )*B, or B := alpha*B*op( A ),

  SYNOPSIS
       SUBROUTINE DTRMM ( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A,
                          LDA, B, LDB )

           CHARACTER*1  SIDE, UPLO, TRANSA, DIAG

           INTEGER      M, N, LDA, LDB

           DOUBLE       PRECISION ALPHA

           DOUBLE       PRECISION A( LDA, * ), B( LDB, * )

  PURPOSE
       DTRMM  performs one of the matrix-matrix operations

       where  alpha  is a scalar,  B  is an m by n matrix,  A  is a
       unit, or non-unit,  upper or lower triangular matrix  and
       op( A )  is one  of

          op( A ) = A   or   op( A ) = A'.

  PARAMETERS
       SIDE   - CHARACTER*1.
              On entry,  SIDE specifies whether  op( A ) multiplies
              B from the left or right as follows:

              SIDE = 'L' or 'l'   B := alpha*op( A )*B.

              SIDE = 'R' or 'r'   B := alpha*B*op( A ).

              Unchanged on exit.

       UPLO   - CHARACTER*1.
```

```
          On entry, UPLO specifies whether the matrix A is an
          upper or lower triangular matrix as follows:

          UPLO = 'U' or 'u'   A is an upper triangular matrix.

          UPLO = 'L' or 'l'   A is a lower triangular matrix.

          Unchanged on exit.

          TRANSA - CHARACTER*1.  On entry, TRANSA specifies the
          form of op( A ) to be used in the matrix multiplica-
          tion as follows:

          TRANSA = 'N' or 'n'   op( A ) = A.

          TRANSA = 'T' or 't'   op( A ) = A'.

          TRANSA = 'C' or 'c'   op( A ) = A'.

          Unchanged on exit.

  DIAG    - CHARACTER*1.
          On entry, DIAG specifies whether or not A is unit
          triangular as follows:

          DIAG = 'U' or 'u'   A is assumed to be unit triangu-
          lar.

          DIAG = 'N' or 'n'   A is not assumed to be unit tri-
          angular.

          Unchanged on exit.

  M       - INTEGER.
          On entry, M specifies the number of rows of B. M must
          be at least zero.  Unchanged on exit.

  N       - INTEGER.
          On entry, N specifies the number of columns of B.  N
          must be at least zero.  Unchanged on exit.

  ALPHA   - DOUBLE PRECISION.
          On entry,  ALPHA specifies the scalar  alpha. When
          alpha is zero then  A is not referenced and  B need
          not be set before entry.  Unchanged on exit.
```

```
is m
     A      -
             DOUBLE PRECISION array of DIMENSION ( LDA, k ), where k
            when  SIDE = 'L' or 'l'  and is  n  when  SIDE = 'R'
            or 'r'.  Before entry  with  UPLO = 'U' or 'u',  the
            leading  k by k upper triangular part of the array  A
            must contain the upper triangular matrix  and the
            strictly lower triangular part of A is not refer-
            enced.  Before entry  with  UPLO = 'L' or 'l',  the
            leading  k by k lower triangular part of the array  A
            must contain the lower triangular matrix  and the
            strictly upper triangular part of A is not refer-
            enced.  Note that when  DIAG = 'U' or 'u',  the diag-
            onal elements of A  are not referenced either,  but
            are assumed to be  unity.  Unchanged on exit.

     LDA    - INTEGER.
            On entry, LDA specifies the first dimension of A as
            declared in the calling (sub) program.  When  SIDE =
            'L' or 'l'  then LDA  must be at least  max( 1, m ),
            when  SIDE = 'R' or 'r' then LDA must be at least

            max( 1, n ).  Unchanged on exit.

     B      - DOUBLE PRECISION array of DIMENSION ( LDB, n ).
            Before entry,  the leading  m by n part of the array
            B must contain the matrix  B,  and  on exit  is
            overwritten  by the transformed matrix.

     LDB    - INTEGER.
            On entry, LDB specifies the first dimension of B as
            declared  in  the  calling  (sub)  program.   LDB
            must  be  at  least max( 1, m ).  Unchanged on exit.
```

⟨*BLAS 3 dtrmm*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dtrmm (side uplo transa diag m n alpha a lda b ldb$)
    (declare (type (simple-array double-float (*)) b a)
             (type (double-float) alpha)
             (type fixnum ldb$ lda n m)
             (type character diag transa uplo side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (uplo character uplo-%data% uplo-%offset%)
         (transa character transa-%data% transa-%offset%)
         (diag character diag-%data% diag-%offset%)
         (a double-float a-%data% a-%offset%)
         (b double-float b-%data% b-%offset%))
      (prog ((temp 0.0) (i 0) (info 0) (j 0) (k 0) (nrowa 0) (lside nil)
             (nounit nil) (upper nil))
        (declare (type (double-float) temp)
                 (type fixnum i info j k nrowa)
                 (type (member t nil) lside nounit upper))
        (setf lside (char-equal side #\L))
        (cond
          (lside
           (setf nrowa m))
          (t
           (setf nrowa n)))
        (setf nounit (char-equal diag #\N))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not lside) (not (char-equal side #\R)))
           (setf info 1))
          ((and (not upper) (not (char-equal uplo #\L)))
           (setf info 2))
          ((and (not (char-equal transa #\N))
                (not (char-equal transa #\T))
                (not (char-equal transa #\C)))
           (setf info 3))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 4))
          ((< m 0)
           (setf info 5))
          ((< n 0)
           (setf info 6))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))
```

```
   (setf info 9))
  ((< ldb$ (max (the fixnum 1) (the fixnum m)))
   (setf info 11)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DTRMM" info)
   (go end_label)))
(if (= n 0) (go end_label))
(cond
  ((= alpha zero)
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                 ((> j n) nil)
     (tagbody
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i m) nil)
         (tagbody
           (setf (f2cl-lib:fref b-%data%
                                (i j)
                                ((1 ldb$) (1 *))
                                b-%offset%)
                 zero)))))
   (go end_label)))
(cond
  (lside
   (cond
     ((char-equal transa #\N)
      (cond
        (upper
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                           ((> k m) nil)
               (tagbody
                 (cond
                   ((/= (f2cl-lib:fref b (k j) ((1 ldb$) (1 *))) zero)
                    (setf temp
                            (* alpha
                               (f2cl-lib:fref b-%data%
                                              (k j)
                                              ((1 ldb$) (1 *))
                                              b-%offset%)))
                    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                  ((> i
```

```
                                        (f2cl-lib:int-add k
                                                         (f2cl-lib:int-sub
                                                          1)))
                                    nil)
                              (tagbody
                                (setf (f2cl-lib:fref b-%data%
                                                     (i j)
                                                     ((1 ldb$) (1 *))
                                                     b-%offset%)
                                      (+
                                       (f2cl-lib:fref b-%data%
                                                      (i j)
                                                      ((1 ldb$) (1 *))
                                                      b-%offset%)
                                       (* temp
                                          (f2cl-lib:fref a-%data%
                                                         (i k)
                                                         ((1 lda) (1 *))
                                                         a-%offset%))))))
                              (if nounit
                                  (setf temp
                                        (* temp
                                           (f2cl-lib:fref a-%data%
                                                          (k k)
                                                          ((1 lda) (1 *))
                                                          a-%offset%))))
                              (setf (f2cl-lib:fref b-%data%
                                                   (k j)
                                                   ((1 ldb$) (1 *))
                                                   b-%offset%)
                                    temp))))))))
                    (t
                     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                   ((> j n) nil)
                       (tagbody
                         (f2cl-lib:fdo (k m
                                        (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
                                       ((> k 1) nil)
                           (tagbody
                             (cond
                               ((/= (f2cl-lib:fref b (k j) ((1 ldb$) (1 *))) zero)
                                (setf temp
                                      (* alpha
                                         (f2cl-lib:fref b-%data%
                                                        (k j)
                                                        ((1 ldb$) (1 *))
```

```
                                    b-%offset%)))
                        (setf (f2cl-lib:fref b-%data%
                                             (k j)
                                             ((1 ldb$) (1 *))
                                             b-%offset%)
                              temp)
                        (if nounit
                            (setf (f2cl-lib:fref b-%data%
                                                 (k j)
                                                 ((1 ldb$) (1 *))
                                                 b-%offset%)
                                  (*
                                   (f2cl-lib:fref b-%data%
                                                  (k j)
                                                  ((1 ldb$) (1 *))
                                                  b-%offset%)
                                   (f2cl-lib:fref a-%data%
                                                  (k k)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))))
                        (f2cl-lib:fdo (i (f2cl-lib:int-add k 1)
                                       (f2cl-lib:int-add i 1))
                                      ((> i m) nil)
                          (tagbody
                            (setf (f2cl-lib:fref b-%data%
                                                 (i j)
                                                 ((1 ldb$) (1 *))
                                                 b-%offset%)
                                  (+
                                   (f2cl-lib:fref b-%data%
                                                  (i j)
                                                  ((1 ldb$) (1 *))
                                                  b-%offset%)
                                   (* temp
                                      (f2cl-lib:fref a-%data%
                                                     (i k)
                                                     ((1 lda) (1 *))
                                                     a-%offset%))))))))))))
                (t
                 (cond
                   (upper
                    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                  ((> j n) nil)
                      (tagbody
                        (f2cl-lib:fdo (i m
                                       (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
```

```
                    ((> i 1) nil)
                (tagbody
                  (setf temp
                          (f2cl-lib:fref b-%data%
                                         (i j)
                                         ((1 ldb$) (1 *))
                                         b-%offset%))
                  (if nounit
                      (setf temp
                              (* temp
                                 (f2cl-lib:fref a-%data%
                                                (i i)
                                                ((1 lda) (1 *))
                                                a-%offset%))))
                  (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                                ((> k
                                    (f2cl-lib:int-add i
                                                      (f2cl-lib:int-sub
                                                       1)))
                                 nil)
                    (tagbody
                      (setf temp
                              (+ temp
                                 (*
                                  (f2cl-lib:fref a-%data%
                                                 (k i)
                                                 ((1 lda) (1 *))
                                                 a-%offset%)
                                  (f2cl-lib:fref b-%data%
                                                 (k j)
                                                 ((1 ldb$) (1 *))
                                                 b-%offset%))))))
                  (setf (f2cl-lib:fref b-%data%
                                       (i j)
                                       ((1 ldb$) (1 *))
                                       b-%offset%)
                        (* alpha temp)))))))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i m) nil)
                 (tagbody
                   (setf temp
                           (f2cl-lib:fref b-%data%
```

```
                                        (i j)
                                        ((1 ldb$) (1 *))
                                        b-%offset%))
                        (if nounit
                            (setf temp
                                    (* temp
                                       (f2cl-lib:fref a-%data%
                                                      (i i)
                                                      ((1 lda) (1 *))
                                                      a-%offset%))))
                        (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
                                         (f2cl-lib:int-add k 1))
                                      ((> k m) nil)
                          (tagbody
                            (setf temp
                                    (+ temp
                                       (*
                                        (f2cl-lib:fref a-%data%
                                                       (k i)
                                                       ((1 lda) (1 *))
                                                       a-%offset%)
                                        (f2cl-lib:fref b-%data%
                                                       (k j)
                                                       ((1 ldb$) (1 *))
                                                       b-%offset%))))))
                        (setf (f2cl-lib:fref b-%data%
                                             (i j)
                                             ((1 ldb$) (1 *))
                                             b-%offset%)
                                (* alpha temp)))))))))
                (t
                 (cond
                   ((char-equal transa #\N)
                    (cond
                      (upper
                       (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                     ((> j 1) nil)
                         (tagbody
                           (setf temp alpha)
                           (if nounit
                               (setf temp
                                       (* temp
                                          (f2cl-lib:fref a-%data%
                                                         (j j)
                                                         ((1 lda) (1 *))
                                                         a-%offset%))))
```

```
                    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                  ((> i m) nil)
                      (tagbody
                        (setf (f2cl-lib:fref b-%data%
                                             (i j)
                                             ((1 ldb$) (1 *))
                                             b-%offset%)
                                (* temp
                                   (f2cl-lib:fref b-%data%
                                                  (i j)
                                                  ((1 ldb$) (1 *))
                                                  b-%offset%)))))
                    (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                                  ((> k
                                      (f2cl-lib:int-add j
                                                        (f2cl-lib:int-sub 1)))
                                   nil)
                      (tagbody
                        (cond
                          ((/= (f2cl-lib:fref a (k j) ((1 lda) (1 *))) zero)
                           (setf temp
                                   (* alpha
                                      (f2cl-lib:fref a-%data%
                                                     (k j)
                                                     ((1 lda) (1 *))
                                                     a-%offset%)))
                           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                         ((> i m) nil)
                             (tagbody
                               (setf (f2cl-lib:fref b-%data%
                                                    (i j)
                                                    ((1 ldb$) (1 *))
                                                    b-%offset%)
                                       (+
                                        (f2cl-lib:fref b-%data%
                                                       (i j)
                                                       ((1 ldb$) (1 *))
                                                       b-%offset%)
                                        (* temp
                                           (f2cl-lib:fref b-%data%
                                                          (i k)
                                                          ((1 ldb$) (1 *))
                                                          b-%offset%))))))))))))
                   (t
                    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                  ((> j n) nil)
```

```
(tagbody
  (setf temp alpha)
  (if nounit
      (setf temp
              (* temp
                 (f2cl-lib:fref a-%data%
                                (j j)
                                ((1 lda) (1 *))
                                a-%offset%))))
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                ((> i m) nil)
    (tagbody
      (setf (f2cl-lib:fref b-%data%
                           (i j)
                           ((1 ldb$) (1 *))
                           b-%offset%)
              (* temp
                 (f2cl-lib:fref b-%data%
                                (i j)
                                ((1 ldb$) (1 *))
                                b-%offset%)))))
  (f2cl-lib:fdo (k (f2cl-lib:int-add j 1)
                 (f2cl-lib:int-add k 1))
                ((> k n) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref a (k j) ((1 lda) (1 *))) zero)
         (setf temp
                 (* alpha
                    (f2cl-lib:fref a-%data%
                                   (k j)
                                   ((1 lda) (1 *))
                                   a-%offset%)))
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i m) nil)
           (tagbody
             (setf (f2cl-lib:fref b-%data%
                                  (i j)
                                  ((1 ldb$) (1 *))
                                  b-%offset%)
                     (+
                      (f2cl-lib:fref b-%data%
                                     (i j)
                                     ((1 ldb$) (1 *))
                                     b-%offset%)
                      (* temp
```

```
                                    (f2cl-lib:fref b-%data%
                                              (i k)
                                              ((1 ldb$) (1 *))
                                              b-%offset%)))))))))))))))
                (t
                 (cond
                   (upper
                    (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                                  ((> k n) nil)
                      (tagbody
                        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                      ((> j
                                          (f2cl-lib:int-add k
                                                            (f2cl-lib:int-sub 1)))
                                       nil)
                          (tagbody
                            (cond
                              (((/= (f2cl-lib:fref a (j k) ((1 lda) (1 *))) zero)
                               (setf temp
                                       (* alpha
                                          (f2cl-lib:fref a-%data%
                                                         (j k)
                                                         ((1 lda) (1 *))
                                                         a-%offset%)))
                               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                             ((> i m) nil)
                                 (tagbody
                                   (setf (f2cl-lib:fref b-%data%
                                                        (i j)
                                                        ((1 ldb$) (1 *))
                                                        b-%offset%)
                                           (+
                                            (f2cl-lib:fref b-%data%
                                                           (i j)
                                                           ((1 ldb$) (1 *))
                                                           b-%offset%)
                                            (* temp
                                               (f2cl-lib:fref b-%data%
                                                         (i k)
                                                         ((1 ldb$) (1 *))
                                                         b-%offset%)))))))))))
                        (setf temp alpha)
                        (if nounit
                            (setf temp
                                    (* temp
                                       (f2cl-lib:fref a-%data%
```

```
                                        (k k)
                                        ((1 lda) (1 *))
                                        a-%offset%))))
          (cond
            ((/= temp one)
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i m) nil)
               (tagbody
                 (setf (f2cl-lib:fref b-%data%
                                      (i k)
                                      ((1 ldb$) (1 *))
                                      b-%offset%)
                       (* temp
                          (f2cl-lib:fref b-%data%
                                         (i k)
                                         ((1 ldb$) (1 *))
                                         b-%offset%)))))))))
        (t
         (f2cl-lib:fdo (k n (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
                       ((> k 1) nil)
           (tagbody
             (f2cl-lib:fdo (j (f2cl-lib:int-add k 1)
                              (f2cl-lib:int-add j 1))
                           ((> j n) nil)
               (tagbody
                 (cond
                   ((/= (f2cl-lib:fref a (j k) ((1 lda) (1 *))) zero)
                    (setf temp
                            (* alpha
                               (f2cl-lib:fref a-%data%
                                              (j k)
                                              ((1 lda) (1 *))
                                              a-%offset%)))
                    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                  ((> i m) nil)
                      (tagbody
                        (setf (f2cl-lib:fref b-%data%
                                             (i j)
                                             ((1 ldb$) (1 *))
                                             b-%offset%)
                              (+
                               (f2cl-lib:fref b-%data%
                                              (i j)
                                              ((1 ldb$) (1 *))
                                              b-%offset%)
                               (* temp
```

```
                                          (f2cl-lib:fref b-%data%
                                                    (i k)
                                                    ((1 ldb$) (1 *))
                                                    b-%offset%))))))))))
                        (setf temp alpha)
                        (if nounit
                            (setf temp
                                    (* temp
                                       (f2cl-lib:fref a-%data%
                                                      (k k)
                                                      ((1 lda) (1 *))
                                                      a-%offset%))))
                        (cond
                          ((/= temp one)
                           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                         ((> i m) nil)
                             (tagbody
                               (setf (f2cl-lib:fref b-%data%
                                                    (i k)
                                                    ((1 ldb$) (1 *))
                                                    b-%offset%)
                                     (* temp
                                        (f2cl-lib:fref b-%data%
                                                       (i k)
                                                       ((1 ldb$) (1 *))
                                                       b-%offset%)))))))))))
   end_label
          (return (values nil nil nil nil nil nil nil nil nil nil nil))))))
```

6.6 dtrsm BLAS

⟨*dtrsm.input*⟩≡

```
  )set break resume
  )sys rm -f dtrsm.output
  )spool dtrsm.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dtrsm.help*⟩≡

```
  ==========================================================================
  dtrsm examples
  ==========================================================================

  ==========================================================================
  Man Page Details
  ==========================================================================

   NAME
        DTRSM - solve one of the matrix equations   op( A )*X =
        alpha*B, or X*op( A ) = alpha*B,

   SYNOPSIS
        SUBROUTINE DTRSM ( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A,
                           LDA, B, LDB )

            CHARACTER*1  SIDE, UPLO, TRANSA, DIAG

            INTEGER      M, N, LDA, LDB

            DOUBLE       PRECISION ALPHA

            DOUBLE       PRECISION A( LDA, * ), B( LDB, * )

   PURPOSE
        DTRSM  solves one of the matrix equations

        where alpha is a scalar, X and B are m by n matrices, A is a
        unit, or non-unit,  upper or lower triangular matrix  and
        op( A )  is one  of

           op( A ) = A   or   op( A ) = A'.

        The matrix X is overwritten on B.

   PARAMETERS
        SIDE   - CHARACTER*1.
                 On entry, SIDE specifies whether op( A ) appears on
                 the left or right of X as follows:

                 SIDE = 'L' or 'l'   op( A )*X = alpha*B.

                 SIDE = 'R' or 'r'   X*op( A ) = alpha*B.

                 Unchanged on exit.
```

```
UPLO   - CHARACTER*1.
       On entry, UPLO specifies whether the matrix A is an
       upper or lower triangular matrix as follows:

       UPLO = 'U' or 'u'   A is an upper triangular matrix.

       UPLO = 'L' or 'l'   A is a lower triangular matrix.

       Unchanged on exit.

       TRANSA - CHARACTER*1.  On entry, TRANSA specifies the
       form of op( A ) to be used in the matrix multiplica-
       tion as follows:

       TRANSA = 'N' or 'n'   op( A ) = A.

       TRANSA = 'T' or 't'   op( A ) = A'.

       TRANSA = 'C' or 'c'   op( A ) = A'.

       Unchanged on exit.

DIAG   - CHARACTER*1.
       On entry, DIAG specifies whether or not A is unit
       triangular as follows:

       DIAG = 'U' or 'u'   A is assumed to be unit triangu-
       lar.

       DIAG = 'N' or 'n'   A is not assumed to be unit tri-
       angular.

       Unchanged on exit.

M      - INTEGER.
       On entry, M specifies the number of rows of B. M must
       be at least zero.  Unchanged on exit.

N      - INTEGER.
       On entry, N specifies the number of columns of B.  N
       must be at least zero.  Unchanged on exit.

ALPHA  - DOUBLE PRECISION.
       On entry,  ALPHA specifies the scalar  alpha. When
       alpha is zero then  A is not referenced and  B need
```

```
              not be set before entry.  Unchanged on exit.

   is m
        A      -
               DOUBLE PRECISION array of DIMENSION ( LDA, k ), where k
               when  SIDE = 'L' or 'l'  and is  n  when  SIDE = 'R'
               or 'r'.  Before entry  with  UPLO = 'U' or 'u',  the
               leading  k by k upper triangular part of the array  A
               must contain the upper triangular matrix  and the
               strictly lower triangular part of A is not refer-
               enced.  Before entry  with  UPLO = 'L' or 'l',  the
               leading  k by k lower triangular part of the array  A
               must contain the lower triangular matrix  and the
               strictly upper triangular part of A is not refer-
               enced.  Note that when  DIAG = 'U' or 'u',  the diag-
               onal elements of A  are not referenced either,  but
               are assumed to be  unity.  Unchanged on exit.

        LDA    - INTEGER.
               On entry, LDA specifies the first dimension of A as
               declared in the calling (sub) program.  When  SIDE =

               'L' or 'l'  then LDA  must be at least  max( 1, m ),
               when  SIDE = 'R' or 'r' then LDA must be at least
               max( 1, n ).  Unchanged on exit.

        B      - DOUBLE PRECISION array of DIMENSION ( LDB, n ).
               Before entry,  the leading  m by n part of the array
               B must contain  the  right-hand  side  matrix  B,
               and  on exit  is overwritten by the solution matrix
               X.

        LDB    - INTEGER.
               On entry, LDB specifies the first dimension of B as
               declared in  the  calling  (sub)  program.   LDB
               must  be  at  least max( 1, m ).  Unchanged on exit.
```

⟨*BLAS 3 dtrsm*⟩≡
```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dtrsm (side uplo transa diag m n alpha a lda b ldb$)
    (declare (type (simple-array double-float (*)) b a)
             (type (double-float) alpha)
             (type fixnum ldb$ lda n m)
             (type character diag transa uplo side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (uplo character uplo-%data% uplo-%offset%)
         (transa character transa-%data% transa-%offset%)
         (diag character diag-%data% diag-%offset%)
         (a double-float a-%data% a-%offset%)
         (b double-float b-%data% b-%offset%))
      (prog ((temp 0.0) (i 0) (info 0) (j 0) (k 0) (nrowa 0) (lside nil)
             (nounit nil) (upper nil))
        (declare (type (double-float) temp)
                 (type fixnum i info j k nrowa)
                 (type (member t nil) lside nounit upper))
        (setf lside (char-equal side #\L))
        (cond
          (lside
           (setf nrowa m))
          (t
           (setf nrowa n)))
        (setf nounit (char-equal diag #\N))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not lside) (not (char-equal side #\R)))
           (setf info 1))
          ((and (not upper) (not (char-equal uplo #\L)))
           (setf info 2))
          ((and (not (char-equal transa #\N))
                (not (char-equal transa #\T))
                (not (char-equal transa #\C)))
           (setf info 3))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 4))
          ((< m 0)
           (setf info 5))
          ((< n 0)
           (setf info 6))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))
```

```
   (setf info 9))
  ((< ldb$ (max (the fixnum 1) (the fixnum m)))
   (setf info 11)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DTRSM" info)
   (go end_label)))
(if (= n 0) (go end_label))
(cond
  ((= alpha zero)
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                 ((> j n) nil)
     (tagbody
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i m) nil)
         (tagbody
           (setf (f2cl-lib:fref b-%data%
                                (i j)
                                ((1 ldb$) (1 *))
                                b-%offset%)
                 zero)))))
   (go end_label)))
(cond
  (lside
   (cond
     ((char-equal transa #\N)
      (cond
        (upper
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (cond
               ((/= alpha one)
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                              ((> i m) nil)
                  (tagbody
                    (setf (f2cl-lib:fref b-%data%
                                         (i j)
                                         ((1 ldb$) (1 *))
                                         b-%offset%)
                          (* alpha
                             (f2cl-lib:fref b-%data%
                                            (i j)
                                            ((1 ldb$) (1 *))
```

```
                                        b-%offset%)))))))
                (f2cl-lib:fdo (k m
                               (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
                              ((> k 1) nil)
                  (tagbody
                    (cond
                      ((/= (f2cl-lib:fref b (k j) ((1 ldb$) (1 *))) zero)
                       (if nounit
                           (setf (f2cl-lib:fref b-%data%
                                                (k j)
                                                ((1 ldb$) (1 *))
                                                b-%offset%)
                                 (/
                                  (f2cl-lib:fref b-%data%
                                                 (k j)
                                                 ((1 ldb$) (1 *))
                                                 b-%offset%)
                                  (f2cl-lib:fref a-%data%
                                                 (k k)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))))
                       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                     ((> i
                                         (f2cl-lib:int-add k
                                                           (f2cl-lib:int-sub
                                                            1)))
                                      nil)
                         (tagbody
                           (setf (f2cl-lib:fref b-%data%
                                                (i j)
                                                ((1 ldb$) (1 *))
                                                b-%offset%)
                                 (-
                                  (f2cl-lib:fref b-%data%
                                                 (i j)
                                                 ((1 ldb$) (1 *))
                                                 b-%offset%)
                                  (*
                                   (f2cl-lib:fref b-%data%
                                                  (k j)
                                                  ((1 ldb$) (1 *))
                                                  b-%offset%)
                                   (f2cl-lib:fref a-%data%
                                                  (i k)
                                                  ((1 lda) (1 *))
                                                  a-%offset%)))))))))))))
```

```
(t
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j n) nil)
   (tagbody
     (cond
       ((/= alpha one)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i m) nil)
          (tagbody
            (setf (f2cl-lib:fref b-%data%
                                 (i j)
                                 ((1 ldb$) (1 *))
                                 b-%offset%)
                  (* alpha
                     (f2cl-lib:fref b-%data%
                                    (i j)
                                    ((1 ldb$) (1 *))
                                    b-%offset%)))))))
     (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                   ((> k m) nil)
       (tagbody
         (cond
           ((/= (f2cl-lib:fref b (k j) ((1 ldb$) (1 *))) zero)
            (if nounit
                (setf (f2cl-lib:fref b-%data%
                                     (k j)
                                     ((1 ldb$) (1 *))
                                     b-%offset%)
                      (/
                       (f2cl-lib:fref b-%data%
                                      (k j)
                                      ((1 ldb$) (1 *))
                                      b-%offset%)
                       (f2cl-lib:fref a-%data%
                                      (k k)
                                      ((1 lda) (1 *))
                                      a-%offset%))))
            (f2cl-lib:fdo (i (f2cl-lib:int-add k 1)
                             (f2cl-lib:int-add i 1))
                          ((> i m) nil)
              (tagbody
                (setf (f2cl-lib:fref b-%data%
                                     (i j)
                                     ((1 ldb$) (1 *))
                                     b-%offset%)
                      (-
```

```
                                    (f2cl-lib:fref b-%data%
                                                   (i j)
                                                   ((1 ldb$) (1 *))
                                                   b-%offset%)
                                    (*
                                     (f2cl-lib:fref b-%data%
                                                    (k j)
                                                    ((1 ldb$) (1 *))
                                                    b-%offset%)
                                     (f2cl-lib:fref a-%data%
                                                    (i k)
                                                    ((1 lda) (1 *))
                                                    a-%offset%))))))))))))))))
                (t
                 (cond
                   (upper
                    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                  ((> j n) nil)
                      (tagbody
                        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                      ((> i m) nil)
                          (tagbody
                            (setf temp
                                    (* alpha
                                       (f2cl-lib:fref b-%data%
                                                      (i j)
                                                      ((1 ldb$) (1 *))
                                                      b-%offset%)))
                            (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                                          ((> k
                                              (f2cl-lib:int-add i
                                                                (f2cl-lib:int-sub
                                                                 1)))
                                           nil)
                              (tagbody
                                (setf temp
                                        (- temp
                                           (*
                                            (f2cl-lib:fref a-%data%
                                                           (k i)
                                                           ((1 lda) (1 *))
                                                           a-%offset%)
                                            (f2cl-lib:fref b-%data%
                                                           (k j)
                                                           ((1 ldb$) (1 *))
                                                           b-%offset%))))))
```

```
                (if nounit
                    (setf temp
                            (/ temp
                               (f2cl-lib:fref a-%data%
                                              (i i)
                                              ((1 lda) (1 *))
                                              a-%offset%))))
                (setf (f2cl-lib:fref b-%data%
                                     (i j)
                                     ((1 ldb$) (1 *))
                                     b-%offset%)
                      temp))))))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (f2cl-lib:fdo (i m
                              (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                             ((> i 1) nil)
                 (tagbody
                   (setf temp
                           (* alpha
                              (f2cl-lib:fref b-%data%
                                             (i j)
                                             ((1 ldb$) (1 *))
                                             b-%offset%)))
                   (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
                                  (f2cl-lib:int-add k 1))
                                 ((> k m) nil)
                     (tagbody
                       (setf temp
                               (- temp
                                  (*
                                   (f2cl-lib:fref a-%data%
                                                  (k i)
                                                  ((1 lda) (1 *))
                                                  a-%offset%)
                                   (f2cl-lib:fref b-%data%
                                                  (k j)
                                                  ((1 ldb$) (1 *))
                                                  b-%offset%))))))
                   (if nounit
                       (setf temp
                               (/ temp
                                  (f2cl-lib:fref a-%data%
                                                 (i i)
```

```
                                                ((1 lda) (1 *))
                                                a-%offset%))))
                    (setf (f2cl-lib:fref b-%data%
                                         (i j)
                                         ((1 ldb$) (1 *))
                                         b-%offset%)
                          temp))))))))))
          (t
           (cond
             ((char-equal transa #\N)
              (cond
                (upper
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (cond
                       ((/= alpha one)
                        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                      ((> i m) nil)
                          (tagbody
                            (setf (f2cl-lib:fref b-%data%
                                                 (i j)
                                                 ((1 ldb$) (1 *))
                                                 b-%offset%)
                                  (* alpha
                                     (f2cl-lib:fref b-%data%
                                                    (i j)
                                                    ((1 ldb$) (1 *))
                                                    b-%offset%)))))))
                     (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                                   ((> k
                                       (f2cl-lib:int-add j
                                                         (f2cl-lib:int-sub 1)))
                                    nil)
                       (tagbody
                         (cond
                           ((/= (f2cl-lib:fref a (k j) ((1 lda) (1 *))) zero)
                            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                          ((> i m) nil)
                              (tagbody
                                (setf (f2cl-lib:fref b-%data%
                                                     (i j)
                                                     ((1 ldb$) (1 *))
                                                     b-%offset%)
                                      (-
                                       (f2cl-lib:fref b-%data%
```

```
                                                (i j)
                                                ((1 ldb$) (1 *))
                                                b-%offset%)
                                        (*
                                         (f2cl-lib:fref a-%data%
                                                        (k j)
                                                        ((1 lda) (1 *))
                                                        a-%offset%)
                                         (f2cl-lib:fref b-%data%
                                                        (i k)
                                                        ((1 ldb$) (1 *))
                                                        b-%offset%))))))))))
                    (cond
                      (nounit
                       (setf temp
                             (/ one
                                (f2cl-lib:fref a-%data%
                                               (j j)
                                               ((1 lda) (1 *))
                                               a-%offset%)))
                       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                     ((> i m) nil)
                         (tagbody
                           (setf (f2cl-lib:fref b-%data%
                                                (i j)
                                                ((1 ldb$) (1 *))
                                                b-%offset%)
                                 (* temp
                                    (f2cl-lib:fref b-%data%
                                                   (i j)
                                                   ((1 ldb$) (1 *))
                                                   b-%offset%)))))))))
                (t
                 (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                               ((> j 1) nil)
                   (tagbody
                     (cond
                       ((/= alpha one)
                        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                      ((> i m) nil)
                          (tagbody
                            (setf (f2cl-lib:fref b-%data%
                                                 (i j)
                                                 ((1 ldb$) (1 *))
                                                 b-%offset%)
                                  (* alpha
```

```
                    (f2cl-lib:fref b-%data%
                                  (i j)
                                  ((1 ldb$) (1 *))
                                  b-%offset%)))))))
(f2cl-lib:fdo (k (f2cl-lib:int-add j 1)
                 (f2cl-lib:int-add k 1))
              ((> k n) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref a (k j) ((1 lda) (1 *))) zero)
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i m) nil)
         (tagbody
           (setf (f2cl-lib:fref b-%data%
                                (i j)
                                ((1 ldb$) (1 *))
                                b-%offset%)
                 (-
                  (f2cl-lib:fref b-%data%
                                 (i j)
                                 ((1 ldb$) (1 *))
                                 b-%offset%)
                  (*
                   (f2cl-lib:fref a-%data%
                                  (k j)
                                  ((1 lda) (1 *))
                                  a-%offset%)
                   (f2cl-lib:fref b-%data%
                                  (i k)
                                  ((1 ldb$) (1 *))
                                  b-%offset%)))))))))
(cond
  (nounit
   (setf temp
         (/ one
            (f2cl-lib:fref a-%data%
                           (j j)
                           ((1 lda) (1 *))
                           a-%offset%)))
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                 ((> i m) nil)
     (tagbody
       (setf (f2cl-lib:fref b-%data%
                            (i j)
                            ((1 ldb$) (1 *))
                            b-%offset%)
```

```
                                (* temp
                                   (f2cl-lib:fref b-%data%
                                                  (i j)
                                                  ((1 ldb$) (1 *))
                                                  b-%offset%))))))))))))
                        (t
                         (cond
                           (upper
                            (f2cl-lib:fdo (k n (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
                                          ((> k 1) nil)
                              (tagbody
                                (cond
                                  (nounit
                                   (setf temp
                                           (/ one
                                              (f2cl-lib:fref a-%data%
                                                             (k k)
                                                             ((1 lda) (1 *))
                                                             a-%offset%)))
                                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                                 ((> i m) nil)
                                     (tagbody
                                       (setf (f2cl-lib:fref b-%data%
                                                            (i k)
                                                            ((1 ldb$) (1 *))
                                                            b-%offset%)
                                             (* temp
                                                (f2cl-lib:fref b-%data%
                                                               (i k)
                                                               ((1 ldb$) (1 *))
                                                               b-%offset%)))))))))
                                (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                              ((> j
                                                  (f2cl-lib:int-add k
                                                                    (f2cl-lib:int-sub 1)))
                                               nil)
                                  (tagbody
                                    (cond
                                      ((/= (f2cl-lib:fref a (j k) ((1 lda) (1 *))) zero)
                                       (setf temp
                                               (f2cl-lib:fref a-%data%
                                                              (j k)
                                                              ((1 lda) (1 *))
                                                              a-%offset%))
                                       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                                     ((> i m) nil)
```

```
                    (tagbody
                      (setf (f2cl-lib:fref b-%data%
                                           (i j)
                                           ((1 ldb$) (1 *))
                                           b-%offset%)
                            (-
                             (f2cl-lib:fref b-%data%
                                            (i j)
                                            ((1 ldb$) (1 *))
                                            b-%offset%)
                             (* temp
                                (f2cl-lib:fref b-%data%
                                               (i k)
                                               ((1 ldb$) (1 *))
                                               b-%offset%)))))))
          (cond
            ((/= alpha one)
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i m) nil)
               (tagbody
                 (setf (f2cl-lib:fref b-%data%
                                      (i k)
                                      ((1 ldb$) (1 *))
                                      b-%offset%)
                       (* alpha
                          (f2cl-lib:fref b-%data%
                                         (i k)
                                         ((1 ldb$) (1 *))
                                         b-%offset%)))))))))
        (t
         (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                       ((> k n) nil)
           (tagbody
             (cond
               (nounit
                (setf temp
                      (/ one
                         (f2cl-lib:fref a-%data%
                                        (k k)
                                        ((1 lda) (1 *))
                                        a-%offset%)))
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                              ((> i m) nil)
                  (tagbody
                    (setf (f2cl-lib:fref b-%data%
                                         (i k)
```

```
                                ((1 ldb$) (1 *))
                                b-%offset%)
                        (* temp
                           (f2cl-lib:fref b-%data%
                                          (i k)
                                          ((1 ldb$) (1 *))
                                          b-%offset%)))))))
        (f2cl-lib:fdo (j (f2cl-lib:int-add k 1)
                       (f2cl-lib:int-add j 1))
                      ((> j n) nil)
          (tagbody
            (cond
              (((/= (f2cl-lib:fref a (j k) ((1 lda) (1 *))) zero)
               (setf temp
                       (f2cl-lib:fref a-%data%
                                      (j k)
                                      ((1 lda) (1 *))
                                      a-%offset%))
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i m) nil)
                 (tagbody
                   (setf (f2cl-lib:fref b-%data%
                                        (i j)
                                        ((1 ldb$) (1 *))
                                        b-%offset%)
                         (-
                          (f2cl-lib:fref b-%data%
                                         (i j)
                                         ((1 ldb$) (1 *))
                                         b-%offset%)
                          (* temp
                             (f2cl-lib:fref b-%data%
                                            (i k)
                                            ((1 ldb$) (1 *))
                                            b-%offset%)))))))))))
        (cond
          ((/= alpha one)
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                         ((> i m) nil)
             (tagbody
               (setf (f2cl-lib:fref b-%data%
                                    (i k)
                                    ((1 ldb$) (1 *))
                                    b-%offset%)
                     (* alpha
                        (f2cl-lib:fref b-%data%
```

```
                                                    (i k)
                                                    ((1 ldb$) (1 *))
                                                    b-%offset%))))))))))))))))
   end_label
          (return (values nil nil nil nil nil nil nil nil nil nil nil))))))
```

6.7 zgemm BLAS

⟨*zgemm.input*⟩≡

```
  )set break resume
  )sys rm -f zgemm.output
  )spool zgemm.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*zgemm.help*⟩≡

```
  ========================================================================
  zgemm examples
  ========================================================================

  ========================================================================
  Man Page Details
  ========================================================================

   NAME
        ZGEMM - perform one of the matrix-matrix operations   C :=
        alpha*op( A )*op( B ) + beta*C,

   SYNOPSIS
        SUBROUTINE ZGEMM ( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA,
                           B, LDB, BETA, C, LDC )

            CHARACTER*1  TRANSA, TRANSB

            INTEGER      M, N, K, LDA, LDB, LDC

            COMPLEX*16   ALPHA, BETA

            COMPLEX*16   A( LDA, * ), B( LDB, * ), C( LDC, * )

   PURPOSE
        ZGEMM  performs one of the matrix-matrix operations

        where  op( X ) is one of

           op( X ) = X   or   op( X ) = X'   or   op( X ) = conjg(
        X' ),

        alpha and beta are scalars, and A, B and C are matrices,
        with op( A ) an m by k matrix,  op( B )  a  k by n matrix
        and  C an m by n matrix.

   PARAMETERS
        TRANSA - CHARACTER*1.  On entry, TRANSA specifies the form
        of op( A ) to be used in the matrix multiplication as fol-
        lows:

        TRANSA = 'N' or 'n',  op( A ) = A.

        TRANSA = 'T' or 't',  op( A ) = A'.
```

```
        TRANSA = 'C' or 'c',  op( A ) = conjg( A' ).

        Unchanged on exit.

        TRANSB - CHARACTER*1.  On entry, TRANSB specifies the form
        of op( B ) to be used in the matrix multiplication as fol-
        lows:

        TRANSB = 'N' or 'n',  op( B ) = B.

        TRANSB = 'T' or 't',  op( B ) = B'.

        TRANSB = 'C' or 'c',  op( B ) = conjg( B' ).

        Unchanged on exit.

        M      - INTEGER.
               On entry,  M  specifies  the number  of rows  of the
               matrix op( A )  and of the  matrix  C.  M  must  be
               at least  zero.  Unchanged on exit.

        N      - INTEGER.
               On entry,  N  specifies the number  of columns of the
               matrix op( B ) and the number of columns of the
               matrix C. N must be at least zero.  Unchanged on
               exit.

        K      - INTEGER.
               On entry,  K  specifies  the number of columns of the
               matrix op( A ) and the number of rows of the matrix
               op( B ). K must be at least  zero.  Unchanged on
               exit.

        ALPHA  - COMPLEX*16      .
               On entry, ALPHA specifies the scalar alpha.
               Unchanged on exit.

   ka is
        A      -
                COMPLEX*16       array of DIMENSION ( LDA, ka ), where
               k  when  TRANSA = 'N' or 'n',  and is  m  otherwise.
               Before entry with  TRANSA = 'N' or 'n',  the leading
               m by k part of the array  A  must contain the matrix
               A,  otherwise the leading  k by m  part of the array
               A  must contain  the matrix A.  Unchanged on exit.
```

```
        LDA    - INTEGER.
               On entry, LDA specifies the first dimension of A as
               declared in the calling (sub) program. When  TRANSA =
               'N' or 'n' then LDA must be at least  max( 1, m ),
               otherwise  LDA must be at least  max( 1, k ).
               Unchanged on exit.

   kb is
        B      -
                COMPLEX*16       array of DIMENSION ( LDB, kb ), where
               n  when  TRANSB = 'N' or 'n',  and is  k  otherwise.
               Before entry with  TRANSB = 'N' or 'n',  the leading
               k by n part of the array  B  must contain the matrix
               B,  otherwise the leading  n by k  part of the array
               B  must contain  the matrix B.  Unchanged on exit.

        LDB    - INTEGER.
               On entry, LDB specifies the first dimension of B as
               declared in the calling (sub) program. When  TRANSB =
               'N' or 'n' then LDB must be at least  max( 1, k ),

               otherwise  LDB must be at least  max( 1, n ).
               Unchanged on exit.

        BETA   - COMPLEX*16      .
               On entry,  BETA  specifies the scalar  beta.  When
               BETA  is supplied as zero then C need not be set on
               input.  Unchanged on exit.

        C      - COMPLEX*16       array of DIMENSION ( LDC, n ).
               Before entry, the leading  m by n  part of the array
               C must contain the matrix  C,  except when  beta  is
               zero, in which case C need not be set on entry.  On
               exit, the array  C  is overwritten by the  m by n
               matrix ( alpha*op( A )*op( B ) + beta*C ).

        LDC    - INTEGER.
               On entry, LDC specifies the first dimension of C as
               declared in  the  calling  (sub)  program.   LDC
               must  be  at  least max( 1, m ).  Unchanged on exit.
```

⟨*BLAS 3 zgemm*⟩≡

```
(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
 (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zgemm (transa transb m n k alpha a lda b ldb$ beta c ldc)
    (declare (type (simple-array (complex double-float) (*)) c b a)
             (type (complex double-float) beta alpha)
             (type fixnum ldc ldb$ lda k n m)
             (type character transb transa))
    (f2cl-lib:with-multi-array-data
        ((transa character transa-%data% transa-%offset%)
         (transb character transb-%data% transb-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (b (complex double-float) b-%data% b-%offset%)
         (c (complex double-float) c-%data% c-%offset%))
      (prog ((temp #C(0.0 0.0)) (i 0) (info 0) (j 0) (l 0) (ncola 0) (nrowa 0)
             (nrowb 0) (conja nil) (conjb nil) (nota nil) (notb nil))
        (declare (type (complex double-float) temp)
                 (type fixnum i info j l ncola nrowa nrowb)
                 (type (member t nil) conja conjb nota notb))
        (setf nota (char-equal transa #\N))
        (setf notb (char-equal transb #\N))
        (setf conja (char-equal transa #\C))
        (setf conjb (char-equal transb #\C))
        (cond
          (nota
           (setf nrowa m)
           (setf ncola k))
          (t
           (setf nrowa k)
           (setf ncola m)))
        (cond
          (notb
           (setf nrowb k))
          (t
           (setf nrowb n)))
        (setf info 0)
        (cond
          ((and (not nota) (not conja) (not (char-equal transa #\T)))
           (setf info 1))
          ((and (not notb) (not conjb) (not (char-equal transb #\T)))
           (setf info 2))
          ((< m 0)
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((< k 0)
```

```
   (setf info 5))
  ((< lda (max (the fixnum 1) (the fixnum nrowa)))
   (setf info 8))
  ((< ldb$
      (max (the fixnum 1) (the fixnum nrowb)))
   (setf info 10))
  ((< ldc (max (the fixnum 1) (the fixnum m)))
   (setf info 13)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "ZGEMM" info)
   (go end_label)))
(if (or (= m 0) (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
    (go end_label))
(cond
  ((= alpha zero)
   (cond
     ((= beta zero)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                    zero))))))
     (t
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                    (* beta
                       (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
```

```
                                                c-%offset%)))))))))
          (go end_label)))
        (cond
          (notb
           (cond
             (nota
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (cond
                    ((= beta zero)
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i m) nil)
                       (tagbody
                         (setf (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                                       zero))))
                    ((/= beta one)
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i m) nil)
                       (tagbody
                         (setf (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                                       (* beta
                                          (f2cl-lib:fref c-%data%
                                                         (i j)
                                                         ((1 ldc) (1 *))
                                                         c-%offset%)))))))
                  (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                ((> l k) nil)
                    (tagbody
                      (cond
                        ((/= (f2cl-lib:fref b (l j) ((1 ldb$) (1 *))) zero)
                         (setf temp
                                 (* alpha
                                    (f2cl-lib:fref b-%data%
                                                   (l j)
                                                   ((1 ldb$) (1 *))
                                                   b-%offset%)))
                         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                       ((> i m) nil)
                           (tagbody
```

```
                        (setf (f2cl-lib:fref c-%data%
                                             (i j)
                                             ((1 ldc) (1 *))
                                             c-%offset%)
                              (+
                               (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                               (* temp
                                  (f2cl-lib:fref a-%data%
                                                 (i l)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))))))))))))
                (conja
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i m) nil)
                       (tagbody
                         (setf temp zero)
                         (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                       ((> l k) nil)
                           (tagbody
                             (setf temp
                                   (+ temp
                                      (*
                                       (f2cl-lib:dconjg
                                        (f2cl-lib:fref a-%data%
                                                       (l i)
                                                       ((1 lda) (1 *))
                                                       a-%offset%))
                                       (f2cl-lib:fref b-%data%
                                                      (l j)
                                                      ((1 ldb$) (1 *))
                                                      b-%offset%))))))
                         (cond
                           ((= beta zero)
                            (setf (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (* alpha temp)))
                           (t
                            (setf (f2cl-lib:fref c-%data%
```

```
                                        (i j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                                (+ (* alpha temp)
                                   (* beta
                                      (f2cl-lib:fref c-%data%
                                                     (i j)
                                                     ((1 ldc) (1 *))
                                                     c-%offset%)))))))))))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i m) nil)
                 (tagbody
                   (setf temp zero)
                   (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                 ((> l k) nil)
                     (tagbody
                       (setf temp
                               (+ temp
                                  (*
                                   (f2cl-lib:fref a-%data%
                                                  (l i)
                                                  ((1 lda) (1 *))
                                                  a-%offset%)
                                   (f2cl-lib:fref b-%data%
                                                  (l j)
                                                  ((1 ldb$) (1 *))
                                                  b-%offset%))))))
                   (cond
                     ((= beta zero)
                      (setf (f2cl-lib:fref c-%data%
                                           (i j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                            (* alpha temp)))
                     (t
                      (setf (f2cl-lib:fref c-%data%
                                           (i j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                            (+ (* alpha temp)
                               (* beta
                                  (f2cl-lib:fref c-%data%
```

```
                                                    (i j)
                                                    ((1 ldc) (1 *))
                                                    c-%offset%))))))))))))))
                (nota
                 (cond
                   (conjb
                    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                  ((> j n) nil)
                      (tagbody
                        (cond
                          ((= beta zero)
                           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                         ((> i m) nil)
                             (tagbody
                               (setf (f2cl-lib:fref c-%data%
                                                    (i j)
                                                    ((1 ldc) (1 *))
                                                    c-%offset%)
                                     zero))))
                          ((/= beta one)
                           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                         ((> i m) nil)
                             (tagbody
                               (setf (f2cl-lib:fref c-%data%
                                                    (i j)
                                                    ((1 ldc) (1 *))
                                                    c-%offset%)
                                     (* beta
                                        (f2cl-lib:fref c-%data%
                                                       (i j)
                                                       ((1 ldc) (1 *))
                                                       c-%offset%)))))))
                        (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                      ((> l k) nil)
                          (tagbody
                            (cond
                              ((/= (f2cl-lib:fref b (j l) ((1 ldb$) (1 *))) zero)
                               (setf temp
                                       (* alpha
                                          (f2cl-lib:dconjg
                                           (f2cl-lib:fref b-%data%
                                                          (j l)
                                                          ((1 ldb$) (1 *))
                                                          b-%offset%))))
                               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                             ((> i m) nil)
```

```
                    (tagbody
                      (setf (f2cl-lib:fref c-%data%
                                           (i j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                            (+
                             (f2cl-lib:fref c-%data%
                                            (i j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)
                             (* temp
                                (f2cl-lib:fref a-%data%
                                               (i l)
                                               ((1 lda) (1 *))
                                               a-%offset%)))))))))))))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (cond
                 ((= beta zero)
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i m) nil)
                    (tagbody
                      (setf (f2cl-lib:fref c-%data%
                                           (i j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                            zero))))
                 ((/= beta one)
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i m) nil)
                    (tagbody
                      (setf (f2cl-lib:fref c-%data%
                                           (i j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                            (* beta
                               (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)))))))
               (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                             ((> l k) nil)
                 (tagbody
                   (cond
```

```
                    (((/= (f2cl-lib:fref b (j l) ((1 ldb$) (1 *))) zero)
                     (setf temp
                             (* alpha
                                (f2cl-lib:fref b-%data%
                                               (j l)
                                               ((1 ldb$) (1 *))
                                               b-%offset%)))
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i m) nil)
                       (tagbody
                         (setf (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                                 (+
                                  (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (* temp
                                     (f2cl-lib:fref a-%data%
                                                    (i l)
                                                    ((1 lda) (1 *))
                                                    a-%offset%))))))))))
               (conja
                (cond
                  (conjb
                   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                 ((> j n) nil)
                     (tagbody
                       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                     ((> i m) nil)
                         (tagbody
                           (setf temp zero)
                           (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                         ((> l k) nil)
                             (tagbody
                               (setf temp
                                       (+ temp
                                          (*
                                           (f2cl-lib:dconjg
                                            (f2cl-lib:fref a-%data%
                                                           (l i)
                                                           ((1 lda) (1 *))
                                                           a-%offset%))
                                           (f2cl-lib:dconjg
```

```
                                (f2cl-lib:fref b-%data%
                                               (j l)
                                               ((1 ldb$) (1 *))
                                               b-%offset%)))))))
                        (cond
                          ((= beta zero)
                           (setf (f2cl-lib:fref c-%data%
                                                (i j)
                                                ((1 ldc) (1 *))
                                                c-%offset%)
                                 (* alpha temp)))
                          (t
                           (setf (f2cl-lib:fref c-%data%
                                                (i j)
                                                ((1 ldc) (1 *))
                                                c-%offset%)
                                 (+ (* alpha temp)
                                    (* beta
                                       (f2cl-lib:fref c-%data%
                                                      (i j)
                                                      ((1 ldc) (1 *))
                                                      c-%offset%)))))))))
                (t
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i m) nil)
                       (tagbody
                         (setf temp zero)
                         (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                       ((> l k) nil)
                           (tagbody
                             (setf temp
                                   (+ temp
                                      (*
                                       (f2cl-lib:dconjg
                                        (f2cl-lib:fref a-%data%
                                                       (l i)
                                                       ((1 lda) (1 *))
                                                       a-%offset%))
                                       (f2cl-lib:fref b-%data%
                                                      (j l)
                                                      ((1 ldb$) (1 *))
                                                      b-%offset%))))))
                         (cond
```

```
                    ((= beta zero)
                     (setf (f2cl-lib:fref c-%data%
                                          (i j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                             (* alpha temp)))
                    (t
                     (setf (f2cl-lib:fref c-%data%
                                          (i j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                             (+ (* alpha temp)
                                (* beta
                                   (f2cl-lib:fref c-%data%
                                                  (i j)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)))))))))
          (t
           (cond
             (conjb
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i m) nil)
                    (tagbody
                      (setf temp zero)
                      (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                    ((> l k) nil)
                        (tagbody
                          (setf temp
                                  (+ temp
                                     (*
                                      (f2cl-lib:fref a-%data%
                                                     (l i)
                                                     ((1 lda) (1 *))
                                                     a-%offset%)
                                      (f2cl-lib:dconjg
                                       (f2cl-lib:fref b-%data%
                                                      (j l)
                                                      ((1 ldb$) (1 *))
                                                      b-%offset%)))))))
                      (cond
                        ((= beta zero)
                         (setf (f2cl-lib:fref c-%data%
                                              (i j)
```

```
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                                (* alpha temp)))
                    (t
                     (setf (f2cl-lib:fref c-%data%
                                        (i j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                                (+ (* alpha temp)
                                   (* beta
                                      (f2cl-lib:fref c-%data%
                                                     (i j)
                                                     ((1 ldc) (1 *))
                                                     c-%offset%)))))))))))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i m) nil)
                 (tagbody
                   (setf temp zero)
                   (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                 ((> l k) nil)
                     (tagbody
                       (setf temp
                                (+ temp
                                   (*
                                    (f2cl-lib:fref a-%data%
                                                   (l i)
                                                   ((1 lda) (1 *))
                                                   a-%offset%)
                                    (f2cl-lib:fref b-%data%
                                                   (j l)
                                                   ((1 ldb$) (1 *))
                                                   b-%offset%))))))
                   (cond
                     ((= beta zero)
                      (setf (f2cl-lib:fref c-%data%
                                        (i j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                                (* alpha temp)))
                     (t
                      (setf (f2cl-lib:fref c-%data%
                                        (i j)
```

```
                                                ((1 ldc) (1 *))
                                                c-%offset%)
                                        (+ (* alpha temp)
                                           (* beta
                                              (f2cl-lib:fref c-%data%
                                                             (i j)
                                                             ((1 ldc) (1 *))
                                                             c-%offset%)))))))))))))
   end_label
          (return
           (values nil nil nil nil nil nil nil nil nil nil nil nil nil))))))
```

6.8 zhemm BLAS

⟨*zhemm.input*⟩≡

```
  )set break resume
  )sys rm -f zhemm.output
  )spool zhemm.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*zhemm.help*⟩≡

```
  ======================================================================
  zhemm examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        ZHEMM - perform one of the matrix-matrix operations   C :=
        alpha*A*B + beta*C,

   SYNOPSIS
        SUBROUTINE ZHEMM ( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB,
                           BETA, C, LDC )

            CHARACTER*1  SIDE, UPLO

            INTEGER      M, N, LDA, LDB, LDC

            COMPLEX*16   ALPHA, BETA

            COMPLEX*16   A( LDA, * ), B( LDB, * ), C( LDC, * )

   PURPOSE
        ZHEMM  performs one of the matrix-matrix operations

        or

           C := alpha*B*A + beta*C,

        where alpha and beta are scalars, A is an hermitian matrix
        and  B and C are m by n matrices.

   PARAMETERS
        SIDE   - CHARACTER*1.
                 On entry,  SIDE  specifies whether  the  hermitian
                 matrix  A appears on the  left or right  in the
                 operation as follows:

                 SIDE = 'L' or 'l'   C := alpha*A*B + beta*C,

                 SIDE = 'R' or 'r'   C := alpha*B*A + beta*C,

                 Unchanged on exit.
```

```
        UPLO   - CHARACTER*1.
               On  entry,  UPLO  specifies  whether  the  upper  or
               lower triangular  part  of  the  hermitian  matrix
               A  is  to  be referenced as follows:

               UPLO = 'U' or 'u'   Only the upper triangular part of
               the hermitian matrix is to be referenced.

               UPLO = 'L' or 'l'   Only the lower triangular part of
               the hermitian matrix is to be referenced.

               Unchanged on exit.

        M      - INTEGER.
               On entry,  M  specifies the number of rows of the
               matrix  C.  M  must be at least zero.  Unchanged on
               exit.

        N      - INTEGER.
               On entry, N specifies the number of columns of the
               matrix C.  N  must be at least zero.  Unchanged on
               exit.

        ALPHA  - COMPLEX*16      .
               On entry, ALPHA specifies the scalar alpha.
               Unchanged on exit.

   ka is
        A      -
                COMPLEX*16       array of DIMENSION ( LDA, ka ), where
               m  when  SIDE = 'L' or 'l'  and is n  otherwise.
               Before entry  with  SIDE = 'L' or 'l',  the  m by m
               part of the array  A  must contain the  hermitian
               matrix,  such that when  UPLO = 'U' or 'u', the lead-
               ing m by m upper triangular part of the array  A
               must contain the upper triangular part of the  hermi-
               tian matrix and the  strictly  lower triangular part
               of  A  is not referenced,  and when  UPLO = 'L' or
               'l', the leading  m by m  lower triangular part  of
               the  array  A must  contain  the  lower triangular
               part  of the  hermitian matrix and the  strictly
               upper triangular part of  A  is not referenced.
               Before entry  with  SIDE = 'R' or 'r',  the  n by n
               part of the array  A  must contain the  hermitian
               matrix,  such that when  UPLO = 'U' or 'u', the lead-
```

ing n by n upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero. Unchanged on exit.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, n). Unchanged on exit.

B - COMPLEX*16 array of DIMENSION (LDB, n).

Before entry, the leading m by n part of the array B must contain the matrix B. Unchanged on exit.

LDB - INTEGER.
On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m). Unchanged on exit.

BETA - COMPLEX*16 .
On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input. Unchanged on exit.

C - COMPLEX*16 array of DIMENSION (LDC, n).
Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n updated matrix.

LDC - INTEGER.
On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m). Unchanged on exit.

⟨*BLAS 3 zhemm*⟩≡

```
(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
 (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zhemm (side uplo m n alpha a lda b ldb$ beta c ldc)
    (declare (type (simple-array (complex double-float) (*)) c b a)
             (type (complex double-float) beta alpha)
             (type fixnum ldc ldb$ lda n m)
             (type character uplo side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (uplo character uplo-%data% uplo-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (b (complex double-float) b-%data% b-%offset%)
         (c (complex double-float) c-%data% c-%offset%))
      (prog ((temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)) (i 0) (info 0) (j 0) (k 0)
             (nrowa 0) (upper nil))
        (declare (type (complex double-float) temp1 temp2)
                 (type fixnum i info j k nrowa)
                 (type (member t nil) upper))
        (cond
          ((char-equal side #\L)
           (setf nrowa m))
          (t
           (setf nrowa n)))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not (char-equal side #\L)) (not (char-equal side #\R)))
           (setf info 1))
          ((and (not upper) (not (char-equal uplo #\L)))
           (setf info 2))
          ((< m 0)
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))
           (setf info 7))
          ((< ldb$ (max (the fixnum 1) (the fixnum m)))
           (setf info 9))
          ((< ldc (max (the fixnum 1) (the fixnum m)))
           (setf info 12)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZHEMM" info)
```

```
   (go end_label)))
(if (or (= m 0) (= n 0) (and (= alpha zero) (= beta one)))
    (go end_label))
(cond
  ((= alpha zero)
   (cond
     ((= beta zero)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                    zero))))))
     (t
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                    (* beta
                       (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)))))))))
   (go end_label)))
(cond
  ((char-equal side #\L)
   (cond
     (upper
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf temp1
```

```
        (* alpha
           (f2cl-lib:fref b-%data%
                          (i j)
                          ((1 ldb$) (1 *))
                          b-%offset%)))
(setf temp2 zero)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k
                  (f2cl-lib:int-add i
                                    (f2cl-lib:int-sub
                                     1)))
               nil)
  (tagbody
    (setf (f2cl-lib:fref c-%data%
                         (k j)
                         ((1 ldc) (1 *))
                         c-%offset%)
          (+
           (f2cl-lib:fref c-%data%
                          (k j)
                          ((1 ldc) (1 *))
                          c-%offset%)
           (* temp1
              (f2cl-lib:fref a-%data%
                             (k i)
                             ((1 lda) (1 *))
                             a-%offset%))))
    (setf temp2
          (+ temp2
             (*
              (f2cl-lib:fref b-%data%
                             (k j)
                             ((1 ldb$) (1 *))
                             b-%offset%)
              (f2cl-lib:dconjg
               (f2cl-lib:fref a-%data%
                              (k i)
                              ((1 lda) (1 *))
                              a-%offset%)))))))
(cond
  ((= beta zero)
   (setf
    (f2cl-lib:fref c-%data% (i j) ((1 ldc) (1 *))
                   c-%offset%)
    (+
     (* temp1
```

```
                        (coerce (realpart
                         (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *))
                                                       a-%offset%))
                           'double-float))
                       (* alpha temp2))))
                  (t
                   (setf (f2cl-lib:fref c-%data%
                                        (i j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                         (+
                          (* beta
                             (f2cl-lib:fref c-%data%
                                            (i j)
                                            ((1 ldc) (1 *))
                                            c-%offset%))
                          (* temp1
                             (coerce (realpart
                              (f2cl-lib:fref a-%data%
                                             (i i)
                                             ((1 lda) (1 *))
                                             a-%offset%)) 'double-float))

                          (* alpha temp2)))))))))
(t
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j n) nil)
   (tagbody
     (f2cl-lib:fdo (i m (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                   ((> i 1) nil)
       (tagbody
         (setf temp1
                 (* alpha
                    (f2cl-lib:fref b-%data%
                                   (i j)
                                   ((1 ldb$) (1 *))
                                   b-%offset%)))
         (setf temp2 zero)
         (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
                          (f2cl-lib:int-add k 1))
                       ((> k m) nil)
           (tagbody
             (setf (f2cl-lib:fref c-%data%
                                  (k j)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
```

```
                  (+
                   (f2cl-lib:fref c-%data%
                                  (k j)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                   (* temp1
                      (f2cl-lib:fref a-%data%
                                     (k i)
                                     ((1 lda) (1 *))
                                     a-%offset%))))
              (setf temp2
                      (+ temp2
                         (*
                          (f2cl-lib:fref b-%data%
                                         (k j)
                                         ((1 ldb$) (1 *))
                                         b-%offset%)
                          (f2cl-lib:dconjg
                           (f2cl-lib:fref a-%data%
                                          (k i)
                                          ((1 lda) (1 *))
                                          a-%offset%)))))))
          (cond
            ((= beta zero)
             (setf (f2cl-lib:fref c-%data%
                                  (i j)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                    (+
                     (* temp1
                       (coerce (realpart
                        (f2cl-lib:fref a-%data%
                                       (i i)
                                       ((1 lda) (1 *))
                                       a-%offset%)) 'double-float))
                     (* alpha temp2))))
            (t
             (setf (f2cl-lib:fref c-%data%
                                  (i j)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                    (+
                     (* beta
                        (f2cl-lib:fref c-%data%
                                       (i j)
                                       ((1 ldc) (1 *))
```

```
                                                c-%offset%))
                                (* temp1
                                   (coerce (realpart
                                    (f2cl-lib:fref a-%data%
                                              (i i)
                                              ((1 lda) (1 *))
                                              a-%offset%)) 'double-float))
                                (* alpha temp2))))))))))))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf temp1
                       (* alpha
                          (coerce (realpart
                           (f2cl-lib:fref a-%data%
                                          (j j)
                                          ((1 lda) (1 *))
                                          a-%offset%)) 'double-float)))
               (cond
                 ((= beta zero)
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i m) nil)
                    (tagbody
                      (setf (f2cl-lib:fref c-%data%
                                           (i j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                            (* temp1
                               (f2cl-lib:fref b-%data%
                                              (i j)
                                              ((1 ldb$) (1 *))
                                              b-%offset%))))))
                 (t
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i m) nil)
                    (tagbody
                      (setf (f2cl-lib:fref c-%data%
                                           (i j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                            (+
                             (* beta
                                (f2cl-lib:fref c-%data%
                                               (i j)
                                               ((1 ldc) (1 *))
```

```
                                        c-%offset%))
                     (* temp1
                        (f2cl-lib:fref b-%data%
                                       (i j)
                                       ((1 ldb$) (1 *))
                                       b-%offset%))))))))
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
               nil)
  (tagbody
    (cond
      (upper
       (setf temp1
               (* alpha
                  (f2cl-lib:fref a-%data%
                                 (k j)
                                 ((1 lda) (1 *))
                                 a-%offset%))))
      (t
       (setf temp1
               (* alpha
                  (f2cl-lib:dconjg
                   (f2cl-lib:fref a-%data%
                                  (j k)
                                  ((1 lda) (1 *))
                                  a-%offset%))))))
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i m) nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data%
                             (i j)
                             ((1 ldc) (1 *))
                             c-%offset%)
                (+
                 (f2cl-lib:fref c-%data%
                                (i j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* temp1
                    (f2cl-lib:fref b-%data%
                                   (i k)
                                   ((1 ldb$) (1 *))
                                   b-%offset%))))))))
(f2cl-lib:fdo (k (f2cl-lib:int-add j 1) (f2cl-lib:int-add k 1))
              ((> k n) nil)
  (tagbody
```

```
                    (cond
                      (upper
                       (setf temp1
                               (* alpha
                                  (f2cl-lib:dconjg
                                   (f2cl-lib:fref a-%data%
                                                  (j k)
                                                  ((1 lda) (1 *))
                                                  a-%offset%)))))
                      (t
                       (setf temp1
                               (* alpha
                                  (f2cl-lib:fref a-%data%
                                                 (k j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%)))))
                    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                  ((> i m) nil)
                      (tagbody
                        (setf (f2cl-lib:fref c-%data%
                                             (i j)
                                             ((1 ldc) (1 *))
                                             c-%offset%)
                              (+
                               (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                               (* temp1
                                  (f2cl-lib:fref b-%data%
                                                 (i k)
                                                 ((1 ldb$) (1 *))
                                                 b-%offset%)))))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil nil nil nil nil)))))
```

6.9 zher2k BLAS

⟨*zher2k.input*⟩≡

```
)set break resume
)sys rm -f zher2k.output
)spool zher2k.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zher2k.help*⟩≡

```
  ======================================================================
  zher2k examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        ZHER2K - perform one of the hermitian rank 2k operations   C
        := alpha*A*conjg( B' ) + conjg( alpha )*B*conjg( A' ) +
        beta*C,

   SYNOPSIS
        SUBROUTINE ZHER2K( UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB,
                           BETA, C, LDC )

            CHARACTER*1    UPLO, TRANS

            INTEGER        N, K, LDA, LDB, LDC

            DOUBLE         PRECISION BETA

            COMPLEX*16     ALPHA

            COMPLEX*16     A( LDA, * ), B( LDB, * ), C( LDC, * )

   PURPOSE
        ZHER2K  performs one of the hermitian rank 2k operations

        or

           C := alpha*conjg( A' )*B + conjg( alpha )*conjg( B' )*A +
        beta*C,

        where  alpha and beta  are scalars with  beta  real,  C is
        an  n by n hermitian matrix and  A and B  are  n by k
        matrices in the first case and  k by n  matrices in the
        second case.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On  entry,   UPLO  specifies  whether  the  upper  or
                 lower triangular  part  of the  array  C  is to be
                 referenced  as follows:
```

```
              UPLO = 'U' or 'u'   Only the  upper triangular part
              of  C is to be referenced.

              UPLO = 'L' or 'l'   Only the  lower triangular part
              of  C is to be referenced.

              Unchanged on exit.

       TRANS  - CHARACTER*1.
              On entry,  TRANS  specifies the operation to be per-
              formed as follows:

              TRANS = 'N' or 'n'    C := alpha*A*conjg( B' )

              + conjg( alpha )*B*conjg( A' ) + beta*C.

              TRANS = 'C' or 'c'    C := alpha*conjg( A' )*B
              + conjg( alpha )*conjg( B' )*A + beta*C.

              Unchanged on exit.

       N      - INTEGER.
              On entry,  N specifies the order of the matrix C.  N
              must be at least zero.  Unchanged on exit.

       K      - INTEGER.
              On entry with  TRANS = 'N' or 'n',  K  specifies  the
              number of  columns  of the  matrices  A and B,  and
              on  entry  with TRANS = 'C' or 'c',  K  specifies
              the number of rows of the matrices  A and B.  K must
              be at least zero.  Unchanged on exit.

       ALPHA  - COMPLEX*16      .
              On entry, ALPHA specifies the scalar alpha.
              Unchanged on exit.

  ka is
       A      -
               COMPLEX*16       array of DIMENSION ( LDA, ka ), where
              k  when  TRANS = 'N' or 'n',  and is  n  otherwise.
              Before entry with  TRANS = 'N' or 'n',  the  leading
              n by k part of the array  A  must contain the matrix
              A,  otherwise the leading  k by n  part of the array
              A  must contain  the matrix A.  Unchanged on exit.
```

```
        LDA     - INTEGER.
                On entry, LDA specifies the first dimension of A as
                declared in  the  calling  (sub)  program.   When
                TRANS = 'N' or 'n' then  LDA must be at least  max(
                1, n ), otherwise  LDA must be at least  max( 1, k ).
                Unchanged on exit.

   kb is
        B       -
                 COMPLEX*16       array of DIMENSION ( LDB, kb ), where
                k  when  TRANS = 'N' or 'n',  and is  n  otherwise.
                Before entry with  TRANS = 'N' or 'n',  the  leading
                n by k part of the array  B  must contain the matrix
                B,  otherwise the leading  k by n  part of the array
                B  must contain  the matrix B.  Unchanged on exit.

        LDB     - INTEGER.
                On entry, LDB specifies the first dimension of B as
                declared in  the  calling  (sub)  program.   When
                TRANS = 'N' or 'n' then  LDB must be at least  max(
                1, n ), otherwise  LDB must be at least  max( 1, k ).

                Unchanged on exit.

        BETA    - DOUBLE PRECISION.
                On entry, BETA specifies the scalar beta.  Unchanged
                on exit.

        C       - COMPLEX*16       array of DIMENSION ( LDC, n ).
                Before entry  with  UPLO = 'U' or 'u',  the leading
                n by n upper triangular part of the array C must con-
                tain the upper triangular part  of the  hermitian
                matrix  and the strictly lower triangular part of C
                is not referenced.  On exit, the upper triangular
                part of the array  C is overwritten by the upper tri-
                angular part of the updated matrix.  Before entry
                with  UPLO = 'L' or 'l',  the leading  n by n lower
                triangular part of the array C must contain the lower
                triangular part  of the  hermitian matrix  and the
                strictly upper triangular part of C is not refer-
                enced.  On exit, the lower triangular part of the
                array  C is overwritten by the lower triangular part
                of the updated matrix.  Note that the imaginary parts
                of the diagonal elements need not be set,  they are
                assumed to be zero,  and on exit they are set to
                zero.
```

```
LDC    - INTEGER.
       On entry, LDC specifies the first dimension of C as
       declared in the calling (sub) program.  LDC
       must be at least max( 1, n ).  Unchanged on exit.
```

⟨*BLAS 3 zher2k*⟩≡
```
(let* ((one 1.0) (zero (complex 0.0 0.0)))
 (declare (type (double-float 1.0 1.0) one) (type (complex double-float) zero))
  (defun zher2k (uplo trans n k alpha a lda b ldb$ beta c ldc)
    (declare (type (double-float) beta)
             (type (simple-array (complex double-float) (*)) c b a)
             (type (complex double-float) alpha)
             (type fixnum ldc ldb$ lda k n)
             (type character trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (b (complex double-float) b-%data% b-%offset%)
         (c (complex double-float) c-%data% c-%offset%))
      (prog ((temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)) (i 0) (info 0) (j 0) (l 0)
             (nrowa 0) (upper nil))
        (declare (type (complex double-float) temp1 temp2)
                 (type fixnum i info j l nrowa)
                 (type (member t nil) upper))
        (cond
          ((char-equal trans #\N)
           (setf nrowa n))
          (t
           (setf nrowa k)))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not upper) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N)) (not (char-equal trans #\C)))
           (setf info 2))
          ((< n 0)
           (setf info 3))
          ((< k 0)
           (setf info 4))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))
           (setf info 7))
          ((< ldb$
              (max (the fixnum 1) (the fixnum nrowa)))
           (setf info 9))
          ((< ldc (max (the fixnum 1) (the fixnum n)))
           (setf info 12)))
        (cond
          ((/= info 0)
           (error
```

```
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "ZHER2K" info)
   (go end_label)))
(if (or (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
    (go end_label))
(cond
  ((= alpha zero)
   (cond
     (upper
      (cond
        ((= beta (coerce (realpart zero) 'double-float))
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i j) nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       zero))))))
        (t
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i
                               (f2cl-lib:int-add j
                                                 (f2cl-lib:int-sub 1)))
                            nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       (* beta
                          (f2cl-lib:fref c-%data%
                                         (i j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)))))
             (setf (f2cl-lib:fref c-%data%
                                  (j j)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                   (coerce
```

```
                    (* beta
                       (coerce (realpart
                        (f2cl-lib:fref c-%data%
                                       (j j)
                                       ((1 ldc) (1 *))
                                       c-%offset%)) 'double-float))
                    '(complex double-float))))))))
          (t
           (cond
             ((= beta (coerce (realpart zero) 'double-float))
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                ((> i n) nil)
                    (tagbody
                      (setf (f2cl-lib:fref c-%data%
                                           (i j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                            zero))))))
             (t
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (setf (f2cl-lib:fref c-%data%
                                       (j j)
                                       ((1 ldc) (1 *))
                                       c-%offset%)
                        (coerce
                         (* beta
                            (coerce (realpart
                             (f2cl-lib:fref c-%data%
                                            (j j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)) 'double-float))
                         '(complex double-float)))
                  (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                   (f2cl-lib:int-add i 1))
                                ((> i n) nil)
                    (tagbody
                      (setf (f2cl-lib:fref c-%data%
                                           (i j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                            (* beta
```

```
                                    (f2cl-lib:fref c-%data%
                                                   (i j)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%))))))))))))
          (go end_label)))
        (cond
          ((char-equal trans #\N)
           (cond
             (upper
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (cond
                    ((= beta (coerce (realpart zero) 'double-float))
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i j) nil)
                       (tagbody
                         (setf (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                               zero))))
                    ((/= beta one)
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i
                                       (f2cl-lib:int-add j
                                                         (f2cl-lib:int-sub 1)))
                                    nil)
                       (tagbody
                         (setf (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                               (* beta
                                  (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)))))
                     (setf (f2cl-lib:fref c-%data%
                                          (j j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                           (coerce
                            (* beta
                               (coerce (realpart
                                        (f2cl-lib:fref c-%data%
```

```
                                        (j j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)) 'double-float))
                        '(complex double-float))))
                (t
                 (setf (f2cl-lib:fref c-%data%
                                      (j j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       (coerce
                        (coerce (realpart
                         (f2cl-lib:fref c-%data%
                                        (j j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)) 'double-float)
                        '(complex double-float)))))
          (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                        ((> l k) nil)
            (tagbody
              (cond
                ((or (/= (f2cl-lib:fref a (j l) ((1 lda) (1 *))) zero)
                     (/= (f2cl-lib:fref b (j l) ((1 ldb$) (1 *))) zero))
                 (setf temp1
                         (* alpha
                            (f2cl-lib:dconjg
                             (f2cl-lib:fref b-%data%
                                            (j l)
                                            ((1 ldb$) (1 *))
                                            b-%offset%))))
                 (setf temp2
                         (coerce
                          (f2cl-lib:dconjg
                           (* alpha
                              (f2cl-lib:fref a-%data%
                                             (j l)
                                             ((1 lda) (1 *))
                                             a-%offset%)))
                          '(complex double-float)))
                 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                               ((> i
                                   (f2cl-lib:int-add j
                                                     (f2cl-lib:int-sub
                                                      1)))
                                nil)
                   (tagbody
                     (setf (f2cl-lib:fref c-%data%
```

```
                                (i j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                            (+
                             (f2cl-lib:fref c-%data%
                                            (i j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)
                             (*
                              (f2cl-lib:fref a-%data%
                                             (i l)
                                             ((1 lda) (1 *))
                                             a-%offset%)
                              temp1)
                             (*
                              (f2cl-lib:fref b-%data%
                                             (i l)
                                             ((1 ldb$) (1 *))
                                             b-%offset%)
                              temp2)))))
(setf (f2cl-lib:fref c-%data%
                     (j j)
                     ((1 ldc) (1 *))
                     c-%offset%)
        (coerce
         (+
          (coerce (realpart
           (f2cl-lib:fref c-%data%
                          (j j)
                          ((1 ldc) (1 *))
                          c-%offset%)) 'double-float)
          (coerce (realpart
           (+
            (*
             (f2cl-lib:fref a-%data%
                            (j l)
                            ((1 lda) (1 *))
                            a-%offset%)
             temp1)
            (*
             (f2cl-lib:fref b-%data%
                            (j l)
                            ((1 ldb$) (1 *))
                            b-%offset%)
             temp2))) 'double-float))
         '(complex double-float)))))))))
```

```
(t
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j n) nil)
   (tagbody
     (cond
       ((= beta (coerce (realpart zero) 'double-float))
        (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                      ((> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref c-%data%
                                 (i j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  zero))))
       ((/= beta one)
        (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                       (f2cl-lib:int-add i 1))
                      ((> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref c-%data%
                                 (i j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (* beta
                     (f2cl-lib:fref c-%data%
                                    (i j)
                                    ((1 ldc) (1 *))
                                    c-%offset%)))))
        (setf (f2cl-lib:fref c-%data%
                             (j j)
                             ((1 ldc) (1 *))
                             c-%offset%)
              (coerce
               (* beta
                  (coerce (realpart
                           (f2cl-lib:fref c-%data%
                                          (j j)
                                          ((1 ldc) (1 *))
                                          c-%offset%))' double-float))
               '(complex double-float))))
       (t
        (setf (f2cl-lib:fref c-%data%
                             (j j)
                             ((1 ldc) (1 *))
                             c-%offset%)
              (coerce
```

```
                (coerce (realpart
                 (f2cl-lib:fref c-%data%
                                (j j)
                                ((1 ldc) (1 *))
                                c-%offset%)) 'double-float)
                '(complex double-float)))))
(f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
              ((> l k) nil)
  (tagbody
    (cond
     ((or (/= (f2cl-lib:fref a (j l) ((1 lda) (1 *))) zero)
          (/= (f2cl-lib:fref b (j l) ((1 ldb$) (1 *))) zero))
       (setf temp1
               (* alpha
                  (f2cl-lib:dconjg
                   (f2cl-lib:fref b-%data%
                                  (j l)
                                  ((1 ldb$) (1 *))
                                  b-%offset%))))
       (setf temp2
               (coerce
                (f2cl-lib:dconjg
                 (* alpha
                    (f2cl-lib:fref a-%data%
                                   (j l)
                                   ((1 lda) (1 *))
                                   a-%offset%)))
                '(complex double-float)))
       (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                        (f2cl-lib:int-add i 1))
                     ((> i n) nil)
         (tagbody
           (setf (f2cl-lib:fref c-%data%
                                (i j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (+
                  (f2cl-lib:fref c-%data%
                                 (i j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (*
                   (f2cl-lib:fref a-%data%
                                  (i l)
                                  ((1 lda) (1 *))
                                  a-%offset%)
```

```
                                    temp1)
                                   (*
                                    (f2cl-lib:fref b-%data%
                                                  (i l)
                                                  ((1 ldb$) (1 *))
                                                  b-%offset%)
                                    temp2)))))
                        (setf (f2cl-lib:fref c-%data%
                                             (j j)
                                             ((1 ldc) (1 *))
                                             c-%offset%)
                              (coerce
                               (+
                                (coerce (realpart
                                 (f2cl-lib:fref c-%data%
                                               (j j)
                                               ((1 ldc) (1 *))
                                               c-%offset%)) 'double-float)
                                (coerce (realpart
                                 (+
                                  (*
                                   (f2cl-lib:fref a-%data%
                                                  (j l)
                                                  ((1 lda) (1 *))
                                                  a-%offset%)
                                   temp1)
                                  (*
                                   (f2cl-lib:fref b-%data%
                                                  (j l)
                                                  ((1 ldb$) (1 *))
                                                  b-%offset%)
                                   temp2))) 'double-float))
                               '(complex double-float)))))))))))
          (t
           (cond
             (upper
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i j) nil)
                    (tagbody
                      (setf temp1 zero)
                      (setf temp2 zero)
                      (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                    ((> l k) nil)
```

```
                (tagbody
                  (setf temp1
                          (+ temp1
                             (*
                              (f2cl-lib:dconjg
                               (f2cl-lib:fref a-%data%
                                              (l i)
                                              ((1 lda) (1 *))
                                              a-%offset%))
                              (f2cl-lib:fref b-%data%
                                             (l j)
                                             ((1 ldb$) (1 *))
                                             b-%offset%))))
                  (setf temp2
                          (+ temp2
                             (*
                              (f2cl-lib:dconjg
                               (f2cl-lib:fref b-%data%
                                              (l i)
                                              ((1 ldb$) (1 *))
                                              b-%offset%))
                              (f2cl-lib:fref a-%data%
                                             (l j)
                                             ((1 lda) (1 *))
                                             a-%offset%))))))
              (cond
                ((= i j)
                 (cond
                   ((= beta (coerce (realpart zero) 'double-float))
                    (setf (f2cl-lib:fref c-%data%
                                         (j j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                            (coerce
                             (coerce (realpart
                              (+ (* alpha temp1)
                                 (* (f2cl-lib:dconjg alpha) temp2)))
                                 'double-float)
                             '(complex double-float))))
                   (t
                    (setf (f2cl-lib:fref c-%data%
                                         (j j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                            (coerce
                             (+
```

```
                                (* beta
                                   (coerce (realpart
                                    (f2cl-lib:fref c-%data%
                                                   (j j)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%))
                                      'double-float))
                                (coerce (realpart
                                 (+ (* alpha temp1)
                                    (* (f2cl-lib:dconjg alpha) temp2)))
                                 'double-float))
                               '(complex double-float))))))
                      (t
                       (cond
                         ((= beta (coerce (realpart zero) 'double-float))
                          (setf (f2cl-lib:fref c-%data%
                                               (i j)
                                               ((1 ldc) (1 *))
                                               c-%offset%)
                                (+ (* alpha temp1)
                                   (* (f2cl-lib:dconjg alpha) temp2))))
                         (t
                          (setf (f2cl-lib:fref c-%data%
                                               (i j)
                                               ((1 ldc) (1 *))
                                               c-%offset%)
                                (+
                                 (* beta
                                    (f2cl-lib:fref c-%data%
                                                   (i j)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%))
                                 (* alpha temp1)
                                 (* (f2cl-lib:dconjg alpha) temp2))))))))))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                             ((> i n) nil)
                 (tagbody
                   (setf temp1 zero)
                   (setf temp2 zero)
                   (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                 ((> l k) nil)
                     (tagbody
```

```
          (setf temp1
                  (+ temp1
                     (*
                      (f2cl-lib:dconjg
                       (f2cl-lib:fref a-%data%
                                      (l i)
                                      ((1 lda) (1 *))
                                      a-%offset%))
                      (f2cl-lib:fref b-%data%
                                     (l j)
                                     ((1 ldb$) (1 *))
                                     b-%offset%))))
          (setf temp2
                  (+ temp2
                     (*
                      (f2cl-lib:dconjg
                       (f2cl-lib:fref b-%data%
                                      (l i)
                                      ((1 ldb$) (1 *))
                                      b-%offset%))
                      (f2cl-lib:fref a-%data%
                                     (l j)
                                     ((1 lda) (1 *))
                                     a-%offset%))))))
      (cond
        ((= i j)
         (cond
           ((= beta (coerce (realpart zero) 'double-float))
            (setf (f2cl-lib:fref c-%data%
                                 (j j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (coerce
                   (coerce (realpart
                    (+ (* alpha temp1)
                       (* (f2cl-lib:dconjg alpha) temp2)))
                    'double-float)
                   '(complex double-float))))
           (t
            (setf (f2cl-lib:fref c-%data%
                                 (j j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (coerce
                   (+
                    (* beta
```

```
                                        (coerce (realpart
                                         (f2cl-lib:fref c-%data%
                                                        (j j)
                                                        ((1 ldc) (1 *))
                                                        c-%offset%))
                                         'double-float))
                                       (coerce (realpart
                                        (+ (* alpha temp1)
                                           (* (f2cl-lib:dconjg alpha) temp2)))
                                         'double-float))
                                      '(complex double-float))))))
                           (t
                            (cond
                              ((= beta (coerce (realpart zero) 'double-float))
                               (setf (f2cl-lib:fref c-%data%
                                                    (i j)
                                                    ((1 ldc) (1 *))
                                                    c-%offset%)
                                     (+ (* alpha temp1)
                                        (* (f2cl-lib:dconjg alpha) temp2))))
                              (t
                               (setf (f2cl-lib:fref c-%data%
                                                    (i j)
                                                    ((1 ldc) (1 *))
                                                    c-%offset%)
                                     (+
                                      (* beta
                                         (f2cl-lib:fref c-%data%
                                                        (i j)
                                                        ((1 ldc) (1 *))
                                                        c-%offset%))
                                      (* alpha temp1)
                                      (* (f2cl-lib:dconjg alpha) temp2))))))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil nil nil nil nil))))))
```

6.10 zherk BLAS

⟨*zherk.input*⟩≡

```
)set break resume
)sys rm -f zherk.output
)spool zherk.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zherk.help*⟩≡

```
======================================================================
zherk examples
======================================================================

======================================================================
Man Page Details
======================================================================

 NAME
      ZHERK - perform one of the hermitian rank k operations   C
      := alpha*A*conjg( A' ) + beta*C,

 SYNOPSIS
      SUBROUTINE ZHERK ( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA,
                         C, LDC )

          CHARACTER*1  UPLO, TRANS

          INTEGER      N, K, LDA, LDC

          DOUBLE       PRECISION ALPHA, BETA

          COMPLEX*16   A( LDA, * ), C( LDC, * )

 PURPOSE
      ZHERK  performs one of the hermitian rank k operations

      or

         C := alpha*conjg( A' )*A + beta*C,

      where  alpha and beta  are  real scalars,  C is an  n by n
      hermitian matrix and  A  is an  n by k  matrix in the  first
      case and a  k by n matrix in the second case.

 PARAMETERS
      UPLO   - CHARACTER*1.
               On  entry,   UPLO  specifies  whether  the  upper  or
               lower triangular  part  of the  array  C  is to be
               referenced  as follows:

               UPLO = 'U' or 'u'   Only the  upper triangular part
               of  C is to be referenced.

               UPLO = 'L' or 'l'   Only the  lower triangular part
```

```
              of  C is to be referenced.

              Unchanged on exit.

       TRANS  - CHARACTER*1.
              On entry,  TRANS  specifies the operation to be per-
              formed as follows:

              TRANS = 'N' or 'n'   C := alpha*A*conjg( A' ) +
              beta*C.

              TRANS = 'C' or 'c'   C := alpha*conjg( A' )*A +
              beta*C.

              Unchanged on exit.

       N      - INTEGER.
              On entry,  N specifies the order of the matrix C.  N
              must be at least zero.  Unchanged on exit.

       K      - INTEGER.
              On entry with  TRANS = 'N' or 'n',  K  specifies  the
              number of  columns   of  the   matrix   A,   and  on
              entry   with TRANS = 'C' or 'c',  K  specifies  the
              number of rows of the matrix A.  K must be at least
              zero.  Unchanged on exit.

       ALPHA  - DOUBLE PRECISION.
              On entry, ALPHA specifies the scalar alpha.
              Unchanged on exit.

  ka is
       A      -
               COMPLEX*16       array of DIMENSION ( LDA, ka ), where
              k  when  TRANS = 'N' or 'n',  and is  n  otherwise.
              Before entry with  TRANS = 'N' or 'n',  the  leading
              n by k part of the array  A  must contain the matrix
              A,  otherwise the leading  k by n  part of the array
              A  must contain  the matrix A.  Unchanged on exit.

       LDA    - INTEGER.
              On entry, LDA specifies the first dimension of A as
              declared in  the  calling  (sub)  program.   When
              TRANS = 'N' or 'n' then  LDA must be at least  max(
              1, n ), otherwise  LDA must be at least  max( 1, k ).
              Unchanged on exit.
```

```
BETA   - DOUBLE PRECISION.
         On entry, BETA specifies the scalar beta.  Unchanged
         on exit.

C      - COMPLEX*16       array of DIMENSION ( LDC, n ).
         Before entry  with  UPLO = 'U' or 'u',  the leading
         n by n upper triangular part of the array C must con-
         tain the upper triangular part  of the  hermitian
         matrix  and the strictly lower triangular part of C
         is not referenced.  On exit, the upper triangular
         part of the array  C is overwritten by the upper tri-
         angular part of the updated matrix.  Before entry
         with  UPLO = 'L' or 'l',  the leading  n by n lower
         triangular part of the array C must contain the lower
         triangular part  of the  hermitian matrix  and the
         strictly upper triangular part of C is not refer-
         enced.  On exit, the lower triangular part of the
         array  C is overwritten by the lower triangular part
         of the updated matrix.  Note that the imaginary parts

         of the diagonal elements need not be set,  they are
         assumed to be zero,  and on exit they are set to
         zero.

LDC    - INTEGER.
         On entry, LDC specifies the first dimension of C as
         declared in  the  calling  (sub)  program.   LDC
         must  be  at  least max( 1, n ).  Unchanged on exit.
```

⟨*BLAS 3 zherk*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun zherk (uplo trans n k alpha a lda beta c ldc)
    (declare (type (simple-array (complex double-float) (*)) c a)
             (type (double-float) beta alpha)
             (type fixnum ldc lda k n)
             (type character trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (c (complex double-float) c-%data% c-%offset%))
      (prog ((temp #C(0.0 0.0)) (rtemp 0.0) (i 0) (info 0) (j 0) (l 0)
             (nrowa 0) (upper nil))
        (declare (type (complex double-float) temp)
                 (type (double-float) rtemp)
                 (type fixnum i info j l nrowa)
                 (type (member t nil) upper))
        (cond
          ((char-equal trans #\N)
           (setf nrowa n))
          (t
           (setf nrowa k)))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not upper) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N)) (not (char-equal trans #\C)))
           (setf info 2))
          ((< n 0)
           (setf info 3))
          ((< k 0)
           (setf info 4))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))
           (setf info 7))
          ((< ldc (max (the fixnum 1) (the fixnum n)))
           (setf info 10)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZHERK" info)
           (go end_label)))
```

```
(if (or (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
    (go end_label))
(cond
  ((= alpha zero)
   (cond
     (upper
      (cond
        ((= beta zero)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i j) nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       (coerce zero '(complex double-float))))))))
        (t
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i
                               (f2cl-lib:int-add j
                                                 (f2cl-lib:int-sub 1)))
                            nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       (* beta
                          (f2cl-lib:fref c-%data%
                                         (i j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)))))
             (setf (f2cl-lib:fref c-%data%
                                  (j j)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                   (coerce
                    (* beta
                       (coerce (realpart
                        (f2cl-lib:fref c-%data%
```

```
                                        (j j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)) 'double-float))
                    '(complex double-float))))))))
(t
 (cond
   ((= beta zero)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  ((> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                      ((> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref c-%data%
                                 (i j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (coerce zero '(complex double-float))))))))
   (t
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  ((> j n) nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data%
                             (j j)
                             ((1 ldc) (1 *))
                             c-%offset%)
              (coerce
               (* beta
                  (coerce (realpart
                           (f2cl-lib:fref c-%data%
                                          (j j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)) 'double-float))
               '(complex double-float)))
        (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                         (f2cl-lib:int-add i 1))
                      ((> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref c-%data%
                                 (i j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (* beta
                     (f2cl-lib:fref c-%data%
                                    (i j)
                                    ((1 ldc) (1 *))
```

```
                                                 c-%offset%)))))))))))
          (go end_label)))
        (cond
          ((char-equal trans #\N)
           (cond
             (upper
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (cond
                    ((= beta zero)
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i j) nil)
                       (tagbody
                         (setf (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                                 (coerce zero '(complex double-float))))))
                    ((/= beta one)
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i
                                       (f2cl-lib:int-add j
                                                         (f2cl-lib:int-sub 1)))
                                    nil)
                       (tagbody
                         (setf (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                                 (* beta
                                    (f2cl-lib:fref c-%data%
                                                   (i j)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%)))))
                     (setf (f2cl-lib:fref c-%data%
                                          (j j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                             (coerce
                              (* beta
                                 (coerce (realpart
                                  (f2cl-lib:fref c-%data%
                                                 (j j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)) 'double-float))
```

```
                ’(complex double-float))))
          (t
           (setf (f2cl-lib:fref c-%data%
                                (j j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (coerce
                  (coerce (realpart
                   (f2cl-lib:fref c-%data%
                                  (j j)
                                  ((1 ldc) (1 *))
                                  c-%offset%)) ’double-float)
                  ’(complex double-float)))))
        (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                      ((> l k) nil)
          (tagbody
            (cond
              ((/= (f2cl-lib:fref a (j l) ((1 lda) (1 *)))
                   (coerce (complex zero) ’(complex double-float)))
               (setf temp
                       (coerce
                        (* alpha
                           (f2cl-lib:dconjg
                            (f2cl-lib:fref a-%data%
                                           (j l)
                                           ((1 lda) (1 *))
                                           a-%offset%)))
                        ’(complex double-float)))
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i
                                 (f2cl-lib:int-add j
                                                   (f2cl-lib:int-sub
                                                    1)))
                              nil)
                 (tagbody
                   (setf (f2cl-lib:fref c-%data%
                                        (i j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                         (+
                          (f2cl-lib:fref c-%data%
                                         (i j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                          (* temp
                             (f2cl-lib:fref a-%data%
```

```
                                                (i l)
                                                ((1 lda) (1 *))
                                                a-%offset%))))))
                    (setf (f2cl-lib:fref c-%data%
                                         (j j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                          (coerce
                           (+
                            (coerce (realpart
                             (f2cl-lib:fref c-%data%
                                            (j j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)) 'double-float)
                            (coerce (realpart
                             (* temp
                                (f2cl-lib:fref a-%data%
                                               (i l)
                                               ((1 lda) (1 *))
                                               a-%offset%)))
                              'double-float))
                           '(complex double-float)))))))))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (cond
                 ((= beta zero)
                  (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                ((> i n) nil)
                    (tagbody
                      (setf (f2cl-lib:fref c-%data%
                                           (i j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                            (coerce zero '(complex double-float))))))
                 ((/= beta one)
                  (setf (f2cl-lib:fref c-%data%
                                       (j j)
                                       ((1 ldc) (1 *))
                                       c-%offset%)
                        (coerce
                         (* beta
                            (coerce (realpart
                             (f2cl-lib:fref c-%data%
                                            (j j)
```

```
                                    ((1 ldc) (1 *))
                                    c-%offset%)) 'double-float))
                 '(complex double-float)))
    (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                     (f2cl-lib:int-add i 1))
                  ((> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data%
                             (i j)
                             ((1 ldc) (1 *))
                             c-%offset%)
                  (* beta
                     (f2cl-lib:fref c-%data%
                                    (i j)
                                    ((1 ldc) (1 *))
                                    c-%offset%))))))
   (t
    (setf (f2cl-lib:fref c-%data%
                         (j j)
                         ((1 ldc) (1 *))
                         c-%offset%)
          (coerce
           (coerce (realpart
            (f2cl-lib:fref c-%data%
                           (j j)
                           ((1 ldc) (1 *))
                           c-%offset%)) 'double-float)
           '(complex double-float)))))
 (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
               ((> l k) nil)
   (tagbody
     (cond
       ((/= (f2cl-lib:fref a (j l) ((1 lda) (1 *)))
            (coerce (complex zero) '(complex double-float)))
        (setf temp
                (coerce
                 (* alpha
                    (f2cl-lib:dconjg
                     (f2cl-lib:fref a-%data%
                                    (j l)
                                    ((1 lda) (1 *))
                                    a-%offset%)))
                 '(complex double-float)))
        (setf (f2cl-lib:fref c-%data%
                             (j j)
                             ((1 ldc) (1 *))
```

```
                                        c-%offset%)
                                (coerce
                                 (+
                                  (coerce (realpart
                                   (f2cl-lib:fref c-%data%
                                                  (j j)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)) 'double-float)
                                  (coerce (realpart
                                   (* temp
                                      (f2cl-lib:fref a-%data%
                                                     (j l)
                                                     ((1 lda) (1 *))
                                                     a-%offset%)))
                                      'double-float))
                                 '(complex double-float)))
                          (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                                           (f2cl-lib:int-add i 1))
                                        ((> i n) nil)
                            (tagbody
                              (setf (f2cl-lib:fref c-%data%
                                                   (i j)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%)
                                    (+
                                     (f2cl-lib:fref c-%data%
                                                    (i j)
                                                    ((1 ldc) (1 *))
                                                    c-%offset%)
                                     (* temp
                                        (f2cl-lib:fref a-%data%
                                                       (i l)
                                                       ((1 lda) (1 *))
                                                       a-%offset%)))))))))
                (t
                 (cond
                   (upper
                    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                  ((> j n) nil)
                      (tagbody
                        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                      ((> i
                                          (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                       nil)
                          (tagbody
                            (setf temp (coerce zero '(complex double-float)))
```

```
    (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                  ((> l k) nil)
      (tagbody
        (setf temp
                (+ temp
                   (*
                    (f2cl-lib:dconjg
                     (f2cl-lib:fref a-%data%
                                    (l i)
                                    ((1 lda) (1 *))
                                    a-%offset%))
                     (f2cl-lib:fref a-%data%
                                    (l j)
                                    ((1 lda) (1 *))
                                    a-%offset%))))))
    (cond
      ((= beta zero)
       (setf (f2cl-lib:fref c-%data%
                            (i j)
                            ((1 ldc) (1 *))
                            c-%offset%)
             (* alpha temp)))
      (t
       (setf (f2cl-lib:fref c-%data%
                            (i j)
                            ((1 ldc) (1 *))
                            c-%offset%)
             (+ (* alpha temp)
                (* beta
                   (f2cl-lib:fref c-%data%
                                  (i j)
                                  ((1 ldc) (1 *))
                                  c-%offset%)))))))
(setf rtemp zero)
(f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
              ((> l k) nil)
  (tagbody
    (setf rtemp
            (coerce
             (realpart
              (+ rtemp
                 (*
                  (f2cl-lib:dconjg
                   (f2cl-lib:fref a-%data%
                                  (l j)
                                  ((1 lda) (1 *))
```

```
                                          a-%offset%))
                              (f2cl-lib:fref a-%data%
                                          (l j)
                                          ((1 lda) (1 *))
                                          a-%offset%))))
                       'double-float))))
            (cond
              ((= beta zero)
               (setf (f2cl-lib:fref c-%data%
                                    (j j)
                                    ((1 ldc) (1 *))
                                    c-%offset%)
                     (coerce (* alpha rtemp) '(complex double-float))))
              (t
               (setf (f2cl-lib:fref c-%data%
                                    (j j)
                                    ((1 ldc) (1 *))
                                    c-%offset%)
                     (coerce
                      (+ (* alpha rtemp)
                         (* beta
                            (coerce (realpart
                             (f2cl-lib:fref c-%data%
                                          (j j)
                                          ((1 ldc) (1 *))
                                          c-%offset%))
                                    'double-float)))
                      '(complex double-float))))))))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf rtemp zero)
               (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                             ((> l k) nil)
                 (tagbody
                   (setf rtemp
                           (coerce
                            (realpart
                             (+ rtemp
                                (*
                                 (f2cl-lib:dconjg
                                  (f2cl-lib:fref a-%data%
                                          (l j)
                                          ((1 lda) (1 *))
                                          a-%offset%))
```

```
                    (f2cl-lib:fref a-%data%
                                   (l j)
                                   ((1 lda) (1 *))
                                   a-%offset%))))
              'double-float))))
(cond
  ((= beta zero)
   (setf (f2cl-lib:fref c-%data%
                        (j j)
                        ((1 ldc) (1 *))
                        c-%offset%)
          (coerce (* alpha rtemp) '(complex double-float))))
  (t
   (setf (f2cl-lib:fref c-%data%
                        (j j)
                        ((1 ldc) (1 *))
                        c-%offset%)
          (coerce
           (+ (* alpha rtemp)
              (* beta
                 (coerce (realpart
                  (f2cl-lib:fref c-%data%
                                 (j j)
                                 ((1 ldc) (1 *))
                                 c-%offset%))
                   'double-float)))
            '(complex double-float)))))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                 (f2cl-lib:int-add i 1))
               ((> i n) nil)
  (tagbody
    (setf temp (coerce zero '(complex double-float)))
    (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                  ((> l k) nil)
      (tagbody
        (setf temp
                (+ temp
                   (*
                    (f2cl-lib:dconjg
                     (f2cl-lib:fref a-%data%
                                    (l i)
                                    ((1 lda) (1 *))
                                    a-%offset%))
                    (f2cl-lib:fref a-%data%
                                   (l j)
                                   ((1 lda) (1 *))
```

```
                                                a-%offset%))))))
                          (cond
                            ((= beta zero)
                             (setf (f2cl-lib:fref c-%data%
                                                  (i j)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)
                                   (* alpha temp)))
                            (t
                             (setf (f2cl-lib:fref c-%data%
                                                  (i j)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)
                                   (+ (* alpha temp)
                                      (* beta
                                         (f2cl-lib:fref c-%data%
                                                        (i j)
                                                        ((1 ldc) (1 *))
                                                        c-%offset%))))))))))
   end_label
          (return (values nil nil nil nil nil nil nil nil nil nil))))))
```

6.11 zsymm BLAS

⟨*zsymm.input*⟩≡

```
  )set break resume
  )sys rm -f zsymm.output
  )spool zsymm.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*zsymm.help*⟩≡

```
 ==========================================================================
 zsymm examples
 ==========================================================================

 ==========================================================================
 Man Page Details
 ==========================================================================

  NAME
       ZSYMM - perform one of the matrix-matrix operations   C :=
       alpha*A*B + beta*C,

  SYNOPSIS
       SUBROUTINE ZSYMM ( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB,
                          BETA, C, LDC )

           CHARACTER*1  SIDE, UPLO

           INTEGER      M, N, LDA, LDB, LDC

           COMPLEX*16   ALPHA, BETA

           COMPLEX*16   A( LDA, * ), B( LDB, * ), C( LDC, * )

  PURPOSE
       ZSYMM  performs one of the matrix-matrix operations

       or

          C := alpha*B*A + beta*C,

       where  alpha and beta are scalars, A is a symmetric matrix
       and  B and C are m by n matrices.

  PARAMETERS
       SIDE   - CHARACTER*1.
              On entry,  SIDE  specifies whether  the  symmetric
              matrix  A appears on the  left or right  in the
              operation as follows:

              SIDE = 'L' or 'l'   C := alpha*A*B + beta*C,

              SIDE = 'R' or 'r'   C := alpha*B*A + beta*C,

              Unchanged on exit.
```

```
       UPLO   - CHARACTER*1.
              On  entry,  UPLO  specifies  whether  the  upper  or
              lower triangular  part  of  the  symmetric  matrix
              A  is  to  be referenced as follows:

              UPLO = 'U' or 'u'   Only the upper triangular part of
              the symmetric matrix is to be referenced.

              UPLO = 'L' or 'l'   Only the lower triangular part of
              the symmetric matrix is to be referenced.

              Unchanged on exit.

       M      - INTEGER.
              On entry,  M  specifies the number of rows of the
              matrix  C.  M  must be at least zero.  Unchanged on
              exit.

       N      - INTEGER.
              On entry, N specifies the number of columns of the
              matrix C.  N  must be at least zero.  Unchanged on
              exit.

       ALPHA  - COMPLEX*16      .
              On entry, ALPHA specifies the scalar alpha.
              Unchanged on exit.

  ka is
       A      -
               COMPLEX*16       array of DIMENSION ( LDA, ka ), where
              m  when  SIDE = 'L' or 'l'  and is n  otherwise.
              Before entry  with  SIDE = 'L' or 'l',  the  m by m
              part of the array  A  must contain the  symmetric
              matrix,  such that when  UPLO = 'U' or 'u', the lead-
              ing m by m upper triangular part of the array  A
              must contain the upper triangular part of the  sym-
              metric matrix and the  strictly  lower triangular
              part of  A  is not referenced,  and when  UPLO = 'L'
              or 'l', the leading  m by m  lower triangular part
              of the  array  A must  contain  the  lower triangular
              part  of the  symmetric matrix and the  strictly
              upper triangular part of  A  is not referenced.
              Before entry  with  SIDE = 'R' or 'r',  the  n by n
              part of the array  A  must contain the  symmetric
              matrix,  such that when  UPLO = 'U' or 'u', the lead-
```

```
          ing n by n upper triangular part of the array  A
          must contain the upper triangular part of the  sym-
          metric matrix and the  strictly  lower triangular
          part of  A  is not referenced,  and when  UPLO = 'L'
          or 'l', the leading  n by n  lower triangular part
          of the  array  A must  contain  the  lower triangular
          part  of the  symmetric matrix and the  strictly
          upper triangular part of  A  is not referenced.
          Unchanged on exit.

LDA     - INTEGER.
          On entry, LDA specifies the first dimension of A as
          declared in the  calling (sub) program. When  SIDE =
          'L' or 'l'  then LDA must be at least  max( 1, m ),
          otherwise  LDA must be at least max( 1, n ).
          Unchanged on exit.

B       - COMPLEX*16       array of DIMENSION ( LDB, n ).
          Before entry, the leading  m by n part of the array
          B  must contain the matrix B.  Unchanged on exit.

LDB     - INTEGER.
          On entry, LDB specifies the first dimension of B as
          declared in  the  calling  (sub)  program.   LDB
          must  be  at  least max( 1, m ).  Unchanged on exit.

BETA    - COMPLEX*16      .
          On entry,  BETA  specifies the scalar  beta.  When
          BETA  is supplied as zero then C need not be set on
          input.  Unchanged on exit.

C       - COMPLEX*16       array of DIMENSION ( LDC, n ).
          Before entry, the leading  m by n  part of the array
          C must contain the matrix  C,  except when  beta  is
          zero, in which case C need not be set on entry.  On
          exit, the array  C  is overwritten by the  m by n
          updated matrix.

LDC     - INTEGER.
          On entry, LDC specifies the first dimension of C as
          declared in  the  calling  (sub)  program.   LDC
          must  be  at  least max( 1, m ).  Unchanged on exit.
```

⟨*BLAS 3 zsymm*⟩≡

```
(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
 (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zsymm (side uplo m n alpha a lda b ldb$ beta c ldc)
    (declare (type (simple-array (complex double-float) (*)) c b a)
             (type (complex double-float) beta alpha)
             (type fixnum ldc ldb$ lda n m)
             (type character uplo side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (uplo character uplo-%data% uplo-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (b (complex double-float) b-%data% b-%offset%)
         (c (complex double-float) c-%data% c-%offset%))
      (prog ((temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)) (i 0) (info 0) (j 0) (k 0)
             (nrowa 0) (upper nil))
        (declare (type (complex double-float) temp1 temp2)
                 (type fixnum i info j k nrowa)
                 (type (member t nil) upper))
        (cond
          ((char-equal side #\L)
           (setf nrowa m))
          (t
           (setf nrowa n)))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not (char-equal side #\L)) (not (char-equal side #\R)))
           (setf info 1))
          ((and (not upper) (not (char-equal uplo #\L)))
           (setf info 2))
          ((< m 0)
           (setf info 3))
          ((< n 0)
           (setf info 4))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))
           (setf info 7))
          ((< ldb$ (max (the fixnum 1) (the fixnum m)))
           (setf info 9))
          ((< ldc (max (the fixnum 1) (the fixnum m)))
           (setf info 12)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZSYMM" info)
```

```
   (go end_label)))
(if (or (= m 0) (= n 0) (and (= alpha zero) (= beta one)))
    (go end_label))
(cond
  ((= alpha zero)
   (cond
     ((= beta zero)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                    zero))))))
     (t
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                    (* beta
                       (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)))))))))
   (go end_label)))
(cond
  ((char-equal side #\L)
   (cond
     (upper
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf temp1
```

```
          (* alpha
             (f2cl-lib:fref b-%data%
                            (i j)
                            ((1 ldb$) (1 *))
                            b-%offset%)))
(setf temp2 zero)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k
                  (f2cl-lib:int-add i
                                    (f2cl-lib:int-sub
                                     1)))
               nil)
  (tagbody
    (setf (f2cl-lib:fref c-%data%
                         (k j)
                         ((1 ldc) (1 *))
                         c-%offset%)
           (+
            (f2cl-lib:fref c-%data%
                           (k j)
                           ((1 ldc) (1 *))
                           c-%offset%)
            (* temp1
               (f2cl-lib:fref a-%data%
                              (k i)
                              ((1 lda) (1 *))
                              a-%offset%))))
    (setf temp2
            (+ temp2
               (*
                (f2cl-lib:fref b-%data%
                               (k j)
                               ((1 ldb$) (1 *))
                               b-%offset%)
                (f2cl-lib:fref a-%data%
                               (k i)
                               ((1 lda) (1 *))
                               a-%offset%))))))
(cond
  ((= beta zero)
   (setf (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%)
           (+
            (* temp1
```

```
                                    (f2cl-lib:fref a-%data%
                                                   (i i)
                                                   ((1 lda) (1 *))
                                                   a-%offset%))
                             (* alpha temp2))))
                       (t
                        (setf (f2cl-lib:fref c-%data%
                                             (i j)
                                             ((1 ldc) (1 *))
                                             c-%offset%)
                              (+
                               (* beta
                                  (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%))
                               (* temp1
                                  (f2cl-lib:fref a-%data%
                                                 (i i)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))
                               (* alpha temp2)))))))))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (f2cl-lib:fdo (i m (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                             ((> i 1) nil)
                 (tagbody
                   (setf temp1
                         (* alpha
                            (f2cl-lib:fref b-%data%
                                           (i j)
                                           ((1 ldb$) (1 *))
                                           b-%offset%)))
                   (setf temp2 zero)
                   (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
                                    (f2cl-lib:int-add k 1))
                                 ((> k m) nil)
                     (tagbody
                       (setf (f2cl-lib:fref c-%data%
                                            (k j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)
                             (+
                              (f2cl-lib:fref c-%data%
```

```
                                    (k j)
                                    ((1 ldc) (1 *))
                                    c-%offset%)
                     (* temp1
                        (f2cl-lib:fref a-%data%
                                       (k i)
                                       ((1 lda) (1 *))
                                       a-%offset%))))
          (setf temp2
                  (+ temp2
                     (*
                      (f2cl-lib:fref b-%data%
                                     (k j)
                                     ((1 ldb$) (1 *))
                                     b-%offset%)
                      (f2cl-lib:fref a-%data%
                                     (k i)
                                     ((1 lda) (1 *))
                                     a-%offset%))))))
      (cond
        ((= beta zero)
         (setf (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%)
                 (+
                  (* temp1
                     (f2cl-lib:fref a-%data%
                                    (i i)
                                    ((1 lda) (1 *))
                                    a-%offset%))
                  (* alpha temp2))))
        (t
         (setf (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%)
                 (+
                  (* beta
                     (f2cl-lib:fref c-%data%
                                    (i j)
                                    ((1 ldc) (1 *))
                                    c-%offset%))
                  (* temp1
                     (f2cl-lib:fref a-%data%
                                    (i i)
```

```
                                                ((1 lda) (1 *))
                                                a-%offset%))
                                (* alpha temp2))))))))))))
                (t
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (setf temp1
                             (* alpha
                                (f2cl-lib:fref a-%data%
                                               (j j)
                                               ((1 lda) (1 *))
                                               a-%offset%)))
                     (cond
                       ((= beta zero)
                        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                      ((> i m) nil)
                          (tagbody
                            (setf (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (* temp1
                                     (f2cl-lib:fref b-%data%
                                                    (i j)
                                                    ((1 ldb$) (1 *))
                                                    b-%offset%))))))
                       (t
                        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                      ((> i m) nil)
                          (tagbody
                            (setf (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (+
                                   (* beta
                                      (f2cl-lib:fref c-%data%
                                                     (i j)
                                                     ((1 ldc) (1 *))
                                                     c-%offset%))
                                   (* temp1
                                      (f2cl-lib:fref b-%data%
                                                     (i j)
                                                     ((1 ldb$) (1 *))
                                                     b-%offset%)))))))))))
```

```
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
               nil)
  (tagbody
    (cond
      (upper
       (setf temp1
               (* alpha
                  (f2cl-lib:fref a-%data%
                                 (k j)
                                 ((1 lda) (1 *))
                                 a-%offset%))))
      (t
       (setf temp1
               (* alpha
                  (f2cl-lib:fref a-%data%
                                 (j k)
                                 ((1 lda) (1 *))
                                 a-%offset%)))))
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i m) nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data%
                             (i j)
                             ((1 ldc) (1 *))
                             c-%offset%)
                (+
                 (f2cl-lib:fref c-%data%
                                (i j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* temp1
                    (f2cl-lib:fref b-%data%
                                   (i k)
                                   ((1 ldb$) (1 *))
                                   b-%offset%))))))))
(f2cl-lib:fdo (k (f2cl-lib:int-add j 1) (f2cl-lib:int-add k 1))
              ((> k n) nil)
  (tagbody
    (cond
      (upper
       (setf temp1
               (* alpha
                  (f2cl-lib:fref a-%data%
                                 (j k)
                                 ((1 lda) (1 *))
```

```
                                              a-%offset%))))
                        (t
                         (setf temp1
                                 (* alpha
                                    (f2cl-lib:fref a-%data%
                                                   (k j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%)))))
                      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                    ((> i m) nil)
                        (tagbody
                          (setf (f2cl-lib:fref c-%data%
                                               (i j)
                                               ((1 ldc) (1 *))
                                               c-%offset%)
                                  (+
                                   (f2cl-lib:fref c-%data%
                                                  (i j)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)
                                   (* temp1
                                      (f2cl-lib:fref b-%data%
                                                     (i k)
                                                     ((1 ldb$) (1 *))
                                                     b-%offset%)))))))))
   end_label
          (return (values nil nil nil nil nil nil nil nil nil nil nil nil))))))
```

6.12 zsyr2k BLAS

⟨*zsyr2k.input*⟩≡

```
  )set break resume
  )sys rm -f zsyr2k.output
  )spool zsyr2k.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*zsyr2k.help*⟩≡

```
  ======================================================================
  zsyr2k examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        ZSYR2K - perform one of the symmetric rank 2k operations   C
        := alpha*A*B' + alpha*B*A' + beta*C,

   SYNOPSIS
        SUBROUTINE ZSYR2K( UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB,
                           BETA, C, LDC )

            CHARACTER*1    UPLO, TRANS

            INTEGER        N, K, LDA, LDB, LDC

            COMPLEX*16     ALPHA, BETA

            COMPLEX*16     A( LDA, * ), B( LDB, * ), C( LDC, * )

   PURPOSE
        ZSYR2K  performs one of the symmetric rank 2k operations

        or

           C := alpha*A'*B + alpha*B'*A + beta*C,

        where  alpha and beta  are scalars,  C is an  n by n sym-
        metric matrix and  A and B  are  n by k  matrices  in the
        first  case  and  k by n matrices in the second case.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On  entry,   UPLO  specifies  whether  the  upper  or
                 lower triangular  part  of the  array  C  is to be
                 referenced  as follows:

                 UPLO = 'U' or 'u'   Only the  upper triangular part
                 of  C is to be referenced.

                 UPLO = 'L' or 'l'   Only the  lower triangular part
```

```
              of  C is to be referenced.

              Unchanged on exit.

       TRANS  - CHARACTER*1.
              On entry,  TRANS  specifies the operation to be per-
              formed as follows:

              TRANS = 'N' or 'n'    C := alpha*A*B' + alpha*B*A' +
              beta*C.

              TRANS = 'T' or 't'    C := alpha*A'*B + alpha*B'*A +
              beta*C.

              Unchanged on exit.

       N      - INTEGER.
              On entry,  N specifies the order of the matrix C.  N
              must be at least zero.  Unchanged on exit.

       K      - INTEGER.
              On entry with  TRANS = 'N' or 'n',  K  specifies  the
              number of  columns  of the  matrices  A and B,  and
              on  entry  with TRANS = 'T' or 't',  K  specifies
              the number of rows of the matrices  A and B.  K must
              be at least zero.  Unchanged on exit.

       ALPHA  - COMPLEX*16      .
              On entry, ALPHA specifies the scalar alpha.
              Unchanged on exit.

  ka is
       A      -
               COMPLEX*16       array of DIMENSION ( LDA, ka ), where
              k  when  TRANS = 'N' or 'n',  and is  n  otherwise.
              Before entry with  TRANS = 'N' or 'n',  the  leading
              n by k part of the array  A  must contain the matrix
              A,  otherwise the leading  k by n  part of the array
              A  must contain  the matrix A.  Unchanged on exit.

       LDA    - INTEGER.
              On entry, LDA specifies the first dimension of A as
              declared in  the  calling  (sub)  program.   When
              TRANS = 'N' or 'n' then  LDA must be at least  max(
              1, n ), otherwise  LDA must be at least  max( 1, k ).
              Unchanged on exit.
```

```
kb is
     B      -
             COMPLEX*16       array of DIMENSION ( LDB, kb ), where
            k  when  TRANS = 'N' or 'n',  and is  n  otherwise.
            Before entry with  TRANS = 'N' or 'n',  the  leading
            n by k part of the array  B  must contain the matrix
            B,  otherwise the leading  k by n  part of the array
            B  must contain  the matrix B.  Unchanged on exit.

     LDB    - INTEGER.
            On entry, LDB specifies the first dimension of B as
            declared in  the  calling  (sub)  program.   When
            TRANS = 'N' or 'n' then  LDB must be at least  max(
            1, n ), otherwise  LDB must be at least  max( 1, k ).
            Unchanged on exit.

     BETA   - COMPLEX*16      .
            On entry, BETA specifies the scalar beta.  Unchanged
            on exit.

     C      - COMPLEX*16       array of DIMENSION ( LDC, n ).
            Before entry  with  UPLO = 'U' or 'u',  the leading
            n by n upper triangular part of the array C must con-
            tain the upper triangular part  of the  symmetric
            matrix  and the strictly lower triangular part of C
            is not referenced.  On exit, the upper triangular
            part of the array  C is overwritten by the upper tri-
            angular part of the updated matrix.  Before entry
            with  UPLO = 'L' or 'l',  the leading  n by n lower
            triangular part of the array C must contain the lower
            triangular part  of the  symmetric matrix  and the
            strictly upper triangular part of C is not refer-
            enced.  On exit, the lower triangular part of the
            array  C is overwritten by the lower triangular part
            of the updated matrix.

     LDC    - INTEGER.
            On entry, LDC specifies the first dimension of C as
            declared in  the  calling  (sub)  program.   LDC
            must  be  at  least max( 1, n ).  Unchanged on exit.
```

⟨*BLAS 3 zsyr2k*⟩≡

```
(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
 (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zsyr2k (uplo trans n k alpha a lda b ldb$ beta c ldc)
    (declare (type (simple-array (complex double-float) (*)) c b a)
             (type (complex double-float) beta alpha)
             (type fixnum ldc ldb$ lda k n)
             (type character trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (b (complex double-float) b-%data% b-%offset%)
         (c (complex double-float) c-%data% c-%offset%))
      (prog ((temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)) (i 0) (info 0) (j 0) (l 0)
             (nrowa 0) (upper nil))
        (declare (type (complex double-float) temp1 temp2)
                 (type fixnum i info j l nrowa)
                 (type (member t nil) upper))
        (cond
          ((char-equal trans #\N)
           (setf nrowa n))
          (t
           (setf nrowa k)))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not upper) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N)) (not (char-equal trans #\T)))
           (setf info 2))
          ((< n 0)
           (setf info 3))
          ((< k 0)
           (setf info 4))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))
           (setf info 7))
          ((< ldb$
              (max (the fixnum 1) (the fixnum nrowa)))
           (setf info 9))
          ((< ldc (max (the fixnum 1) (the fixnum n)))
           (setf info 12)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
```

```
          "ZSYR2K" info)
         (go end_label)))
      (if (or (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
          (go end_label))
      (cond
        ((= alpha zero)
         (cond
           (upper
            (cond
              ((= beta zero)
               (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                             ((> j n) nil)
                 (tagbody
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i j) nil)
                     (tagbody
                       (setf (f2cl-lib:fref c-%data%
                                            (i j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)
                             zero))))))
              (t
               (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                             ((> j n) nil)
                 (tagbody
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i j) nil)
                     (tagbody
                       (setf (f2cl-lib:fref c-%data%
                                            (i j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)
                             (* beta
                                (f2cl-lib:fref c-%data%
                                               (i j)
                                               ((1 ldc) (1 *))
                                               c-%offset%)))))))))))
           (t
            (cond
              ((= beta zero)
               (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                             ((> j n) nil)
                 (tagbody
                   (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
```

```
                        (setf (f2cl-lib:fref c-%data%
                                             (i j)
                                             ((1 ldc) (1 *))
                                             c-%offset%)
                              zero))))))
                (t
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                   ((> i n) nil)
                       (tagbody
                         (setf (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                               (* beta
                                  (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)))))))))
           (go end_label)))
        (cond
          ((char-equal trans #\N)
           (cond
             (upper
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (cond
                    ((= beta zero)
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i j) nil)
                       (tagbody
                         (setf (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                               zero))))
                    ((/= beta one)
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i j) nil)
                       (tagbody
                         (setf (f2cl-lib:fref c-%data%
                                              (i j)
                                              ((1 ldc) (1 *))
```

```
                                    c-%offset%)
                          (* beta
                             (f2cl-lib:fref c-%data%
                                            (i j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)))))))
        (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                      ((> l k) nil)
          (tagbody
            (cond
              ((or (/= (f2cl-lib:fref a (j l) ((1 lda) (1 *))) zero)
                   (/= (f2cl-lib:fref b (j l) ((1 ldb$) (1 *))) zero))
               (setf temp1
                       (* alpha
                          (f2cl-lib:fref b-%data%
                                         (j l)
                                         ((1 ldb$) (1 *))
                                         b-%offset%)))
               (setf temp2
                       (* alpha
                          (f2cl-lib:fref a-%data%
                                         (j l)
                                         ((1 lda) (1 *))
                                         a-%offset%)))
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i j) nil)
                 (tagbody
                   (setf (f2cl-lib:fref c-%data%
                                        (i j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                         (+
                          (f2cl-lib:fref c-%data%
                                         (i j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                          (*
                           (f2cl-lib:fref a-%data%
                                          (i l)
                                          ((1 lda) (1 *))
                                          a-%offset%)
                           temp1)
                          (*
                           (f2cl-lib:fref b-%data%
                                          (i l)
                                          ((1 ldb$) (1 *))
```

```
                                                  b-%offset%)
                                 temp2))))))))))))
                (t
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j n) nil)
                   (tagbody
                     (cond
                       ((= beta zero)
                        (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                      ((> i n) nil)
                          (tagbody
                            (setf (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  zero))))
                       ((/= beta one)
                        (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                      ((> i n) nil)
                          (tagbody
                            (setf (f2cl-lib:fref c-%data%
                                                 (i j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (* beta
                                     (f2cl-lib:fref c-%data%
                                                    (i j)
                                                    ((1 ldc) (1 *))
                                                    c-%offset%)))))))
                     (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                   ((> l k) nil)
                       (tagbody
                         (cond
                           ((or (/= (f2cl-lib:fref a (j l) ((1 lda) (1 *))) zero)
                                (/= (f2cl-lib:fref b (j l) ((1 ldb$) (1 *))) zero))
                            (setf temp1
                                    (* alpha
                                       (f2cl-lib:fref b-%data%
                                                      (j l)
                                                      ((1 ldb$) (1 *))
                                                      b-%offset%)))
                            (setf temp2
                                    (* alpha
                                       (f2cl-lib:fref a-%data%
                                                      (j l)
                                                      ((1 lda) (1 *))
```

```
                                        a-%offset%)))
                    (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                  ((> i n) nil)
                      (tagbody
                        (setf (f2cl-lib:fref c-%data%
                                             (i j)
                                             ((1 ldc) (1 *))
                                             c-%offset%)
                                      (+
                                       (f2cl-lib:fref c-%data%
                                                      (i j)
                                                      ((1 ldc) (1 *))
                                                      c-%offset%)
                                       (*
                                        (f2cl-lib:fref a-%data%
                                                       (i l)
                                                       ((1 lda) (1 *))
                                                       a-%offset%)
                                        temp1)
                                       (*
                                        (f2cl-lib:fref b-%data%
                                                       (i l)
                                                       ((1 ldb$) (1 *))
                                                       b-%offset%)
                                        temp2))))))))))))
            (t
             (cond
               (upper
                (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                              ((> j n) nil)
                  (tagbody
                    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                  ((> i j) nil)
                      (tagbody
                        (setf temp1 zero)
                        (setf temp2 zero)
                        (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                      ((> l k) nil)
                          (tagbody
                            (setf temp1
                                    (+ temp1
                                       (*
                                        (f2cl-lib:fref a-%data%
                                                       (l i)
                                                       ((1 lda) (1 *))
                                                       a-%offset%)
```

```
                                (f2cl-lib:fref b-%data%
                                               (l j)
                                               ((1 ldb$) (1 *))
                                               b-%offset%))))
                           (setf temp2
                                 (+ temp2
                                    (*
                                     (f2cl-lib:fref b-%data%
                                                    (l i)
                                                    ((1 ldb$) (1 *))
                                                    b-%offset%)
                                     (f2cl-lib:fref a-%data%
                                                    (l j)
                                                    ((1 lda) (1 *))
                                                    a-%offset%))))))
                    (cond
                      ((= beta zero)
                       (setf (f2cl-lib:fref c-%data%
                                            (i j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)
                             (+ (* alpha temp1) (* alpha temp2))))
                      (t
                       (setf (f2cl-lib:fref c-%data%
                                            (i j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)
                             (+
                              (* beta
                                 (f2cl-lib:fref c-%data%
                                                (i j)
                                                ((1 ldc) (1 *))
                                                c-%offset%))
                              (* alpha temp1)
                              (* alpha temp2)))))))))
            (t
             (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                           ((> j n) nil)
               (tagbody
                 (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                               ((> i n) nil)
                   (tagbody
                     (setf temp1 zero)
                     (setf temp2 zero)
                     (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                   ((> l k) nil)
```

```
                        (tagbody
                          (setf temp1
                                  (+ temp1
                                     (*
                                      (f2cl-lib:fref a-%data%
                                                     (l i)
                                                     ((1 lda) (1 *))
                                                     a-%offset%)
                                      (f2cl-lib:fref b-%data%
                                                     (l j)
                                                     ((1 ldb$) (1 *))
                                                     b-%offset%))))
                          (setf temp2
                                  (+ temp2
                                     (*
                                      (f2cl-lib:fref b-%data%
                                                     (l i)
                                                     ((1 ldb$) (1 *))
                                                     b-%offset%)
                                      (f2cl-lib:fref a-%data%
                                                     (l j)
                                                     ((1 lda) (1 *))
                                                     a-%offset%))))))
                        (cond
                          ((= beta zero)
                           (setf (f2cl-lib:fref c-%data%
                                                (i j)
                                                ((1 ldc) (1 *))
                                                c-%offset%)
                                 (+ (* alpha temp1) (* alpha temp2))))
                          (t
                           (setf (f2cl-lib:fref c-%data%
                                                (i j)
                                                ((1 ldc) (1 *))
                                                c-%offset%)
                                 (+
                                  (* beta
                                     (f2cl-lib:fref c-%data%
                                                    (i j)
                                                    ((1 ldc) (1 *))
                                                    c-%offset%))
                                  (* alpha temp1)
                                  (* alpha temp2)))))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil nil nil nil))))))
```

6.13 zsyrk BLAS

⟨*zsyrk.input*⟩≡

```
)set break resume
)sys rm -f zsyrk.output
)spool zsyrk.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zsyrk.help*⟩≡

```
  ======================================================================
  zsyrk examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        ZSYRK - perform one of the symmetric rank k operations   C
        := alpha*A*A' + beta*C,

   SYNOPSIS
        SUBROUTINE ZSYRK ( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA,
                           C, LDC )

            CHARACTER*1  UPLO, TRANS

            INTEGER      N, K, LDA, LDC

            COMPLEX*16   ALPHA, BETA

            COMPLEX*16   A( LDA, * ), C( LDC, * )

   PURPOSE
        ZSYRK  performs one of the symmetric rank k operations

        or

           C := alpha*A'*A + beta*C,

        where  alpha and beta  are scalars,  C is an  n by n sym-
        metric matrix and  A  is an  n by k  matrix in the first
        case and a  k by n  matrix in the second case.

   PARAMETERS
        UPLO   - CHARACTER*1.
                 On  entry,   UPLO  specifies  whether  the  upper  or
                 lower triangular  part  of the  array  C  is to be
                 referenced  as follows:

                 UPLO = 'U' or 'u'   Only the  upper triangular part
                 of  C is to be referenced.

                 UPLO = 'L' or 'l'   Only the  lower triangular part
```

```
              of  C is to be referenced.

              Unchanged on exit.

       TRANS  - CHARACTER*1.
              On entry,  TRANS  specifies the operation to be per-
              formed as follows:

              TRANS = 'N' or 'n'   C := alpha*A*A' + beta*C.

              TRANS = 'T' or 't'   C := alpha*A'*A + beta*C.

              Unchanged on exit.

       N      - INTEGER.
              On entry,  N specifies the order of the matrix C.  N
              must be at least zero.  Unchanged on exit.

       K      - INTEGER.
              On entry with  TRANS = 'N' or 'n',  K  specifies  the
              number of  columns   of  the   matrix   A,   and  on
              entry   with TRANS = 'T' or 't',  K  specifies  the
              number of rows of the matrix A.  K must be at least
              zero.  Unchanged on exit.

       ALPHA  - COMPLEX*16      .
              On entry, ALPHA specifies the scalar alpha.
              Unchanged on exit.

  ka is
       A      -
               COMPLEX*16       array of DIMENSION ( LDA, ka ), where
              k  when  TRANS = 'N' or 'n',  and is  n  otherwise.
              Before entry with  TRANS = 'N' or 'n',  the  leading
              n by k part of the array  A  must contain the matrix
              A,  otherwise the leading  k by n  part of the array
              A  must contain  the matrix A.  Unchanged on exit.

       LDA    - INTEGER.
              On entry, LDA specifies the first dimension of A as
              declared in  the  calling  (sub)  program.   When
              TRANS = 'N' or 'n' then  LDA must be at least  max(
              1, n ), otherwise  LDA must be at least  max( 1, k ).
              Unchanged on exit.

       BETA   - COMPLEX*16      .
```

On entry, BETA specifies the scalar beta. Unchanged on exit.

C - COMPLEX*16 array of DIMENSION (LDC, n). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

LDC - INTEGER.

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n). Unchanged on exit.

⟨*BLAS 3 zsyrk*⟩≡

```
(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
 (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zsyrk (uplo trans n k alpha a lda beta c ldc)
    (declare (type (simple-array (complex double-float) (*)) c a)
             (type (complex double-float) beta alpha)
             (type fixnum ldc lda k n)
             (type character trans uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (trans character trans-%data% trans-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (c (complex double-float) c-%data% c-%offset%))
      (prog ((temp #C(0.0 0.0)) (i 0) (info 0) (j 0) (l 0) (nrowa 0)
             (upper nil))
        (declare (type (complex double-float) temp)
                 (type fixnum i info j l nrowa)
                 (type (member t nil) upper))
        (cond
          ((char-equal trans #\N)
           (setf nrowa n))
          (t
           (setf nrowa k)))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not upper) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N)) (not (char-equal trans #\T)))
           (setf info 2))
          ((< n 0)
           (setf info 3))
          ((< k 0)
           (setf info 4))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))
           (setf info 7))
          ((< ldc (max (the fixnum 1) (the fixnum n)))
           (setf info 10)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZSYRK" info)
           (go end_label)))
        (if (or (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
            (go end_label))
```

```
(cond
  ((= alpha zero)
   (cond
     (upper
      (cond
        ((= beta zero)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i j) nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       zero))))))
        (t
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i j) nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       (* beta
                          (f2cl-lib:fref c-%data%
                                         (i j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)))))))))))
     (t
      (cond
        ((= beta zero)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                           ((> i n) nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
```

```
                                zero))))))
                       (t
                        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                      ((> j n) nil)
                          (tagbody
                            (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                          ((> i n) nil)
                              (tagbody
                                (setf (f2cl-lib:fref c-%data%
                                                     (i j)
                                                     ((1 ldc) (1 *))
                                                     c-%offset%)
                                      (* beta
                                         (f2cl-lib:fref c-%data%
                                                        (i j)
                                                        ((1 ldc) (1 *))
                                                        c-%offset%)))))))))
                   (go end_label)))
               (cond
                 ((char-equal trans #\N)
                  (cond
                    (upper
                     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                   ((> j n) nil)
                       (tagbody
                         (cond
                           ((= beta zero)
                            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                          ((> i j) nil)
                              (tagbody
                                (setf (f2cl-lib:fref c-%data%
                                                     (i j)
                                                     ((1 ldc) (1 *))
                                                     c-%offset%)
                                      zero))))
                           ((/= beta one)
                            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                          ((> i j) nil)
                              (tagbody
                                (setf (f2cl-lib:fref c-%data%
                                                     (i j)
                                                     ((1 ldc) (1 *))
                                                     c-%offset%)
                                      (* beta
                                         (f2cl-lib:fref c-%data%
                                                        (i j)
```

```
                                        ((1 ldc) (1 *))
                                        c-%offset%)))))))
          (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                        ((> l k) nil)
            (tagbody
              (cond
                ((/= (f2cl-lib:fref a (j l) ((1 lda) (1 *))) zero)
                 (setf temp
                         (* alpha
                            (f2cl-lib:fref a-%data%
                                           (j l)
                                           ((1 lda) (1 *))
                                           a-%offset%)))
                 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                               ((> i j) nil)
                   (tagbody
                     (setf (f2cl-lib:fref c-%data%
                                          (i j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                           (+
                            (f2cl-lib:fref c-%data%
                                           (i j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                            (* temp
                               (f2cl-lib:fref a-%data%
                                              (i l)
                                              ((1 lda) (1 *))
                                              a-%offset%)))))))))))
        (t
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (cond
               ((= beta zero)
                (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                              ((> i n) nil)
                  (tagbody
                    (setf (f2cl-lib:fref c-%data%
                                         (i j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                          zero))))
               ((/= beta one)
                (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
```

```
                                ((> i n) nil)
                            (tagbody
                              (setf (f2cl-lib:fref c-%data%
                                                   (i j)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%)
                                    (* beta
                                       (f2cl-lib:fref c-%data%
                                                      (i j)
                                                      ((1 ldc) (1 *))
                                                      c-%offset%)))))))
                        (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                      ((> l k) nil)
                          (tagbody
                            (cond
                              ((/= (f2cl-lib:fref a (j l) ((1 lda) (1 *))) zero)
                               (setf temp
                                       (* alpha
                                          (f2cl-lib:fref a-%data%
                                                         (j l)
                                                         ((1 lda) (1 *))
                                                         a-%offset%)))
                               (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                                             ((> i n) nil)
                                 (tagbody
                                   (setf (f2cl-lib:fref c-%data%
                                                        (i j)
                                                        ((1 ldc) (1 *))
                                                        c-%offset%)
                                         (+
                                          (f2cl-lib:fref c-%data%
                                                         (i j)
                                                         ((1 ldc) (1 *))
                                                         c-%offset%)
                                          (* temp
                                             (f2cl-lib:fref a-%data%
                                                            (i l)
                                                            ((1 lda) (1 *))
                                                            a-%offset%)))))))))))))
                    (t
                     (cond
                       (upper
                        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                      ((> j n) nil)
                          (tagbody
                            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
```

```
                    ((> i j) nil)
            (tagbody
              (setf temp zero)
              (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                            ((> l k) nil)
                (tagbody
                  (setf temp
                          (+ temp
                             (*
                              (f2cl-lib:fref a-%data%
                                             (l i)
                                             ((1 lda) (1 *))
                                             a-%offset%)
                              (f2cl-lib:fref a-%data%
                                             (l j)
                                             ((1 lda) (1 *))
                                             a-%offset%))))))
              (cond
                ((= beta zero)
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       (* alpha temp)))
                (t
                 (setf (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       (+ (* alpha temp)
                          (* beta
                             (f2cl-lib:fref c-%data%
                                            (i j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)))))))))
      (t
       (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                     ((> j n) nil)
         (tagbody
           (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                         ((> i n) nil)
             (tagbody
               (setf temp zero)
               (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                             ((> l k) nil)
                 (tagbody
```

```
                        (setf temp
                                (+ temp
                                   (*
                                    (f2cl-lib:fref a-%data%
                                                   (l i)
                                                   ((1 lda) (1 *))
                                                   a-%offset%)
                                    (f2cl-lib:fref a-%data%
                                                   (l j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%))))))
                    (cond
                      ((= beta zero)
                       (setf (f2cl-lib:fref c-%data%
                                            (i j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)
                             (* alpha temp)))
                      (t
                       (setf (f2cl-lib:fref c-%data%
                                            (i j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)
                             (+ (* alpha temp)
                                (* beta
                                   (f2cl-lib:fref c-%data%
                                                  (i j)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)))))))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil nil nil))))))
```

6.14 ztrmm BLAS

⟨*ztrmm.input*⟩≡

```
)set break resume
)sys rm -f ztrmm.output
)spool ztrmm.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*ztrmm.help*⟩≡

```
 ========================================================================
 ztrmm examples
 ========================================================================

 ========================================================================
 Man Page Details
 ========================================================================

  NAME
       ZTRMM - perform one of the matrix-matrix operations   B :=
       alpha*op( A )*B, or B := alpha*B*op( A )  where alpha is a
       scalar, B is an m by n matrix, A is a unit, or non-unit,
       upper or lower triangular matrix and op( A ) is one of   op(
       A ) = A or op( A ) = A' or op( A ) = conjg( A' )

  SYNOPSIS
       SUBROUTINE ZTRMM ( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A,
                          LDA, B, LDB )

           CHARACTER*1  SIDE, UPLO, TRANSA, DIAG

           INTEGER      M, N, LDA, LDB

           COMPLEX*16   ALPHA

           COMPLEX*16   A( LDA, * ), B( LDB, * )

  PURPOSE
       ZTRMM  performs one of the matrix-matrix operations

  PARAMETERS
       SIDE   - CHARACTER*1.
              On entry,  SIDE specifies whether  op( A ) multiplies
              B from the left or right as follows:

              SIDE = 'L' or 'l'   B := alpha*op( A )*B.

              SIDE = 'R' or 'r'   B := alpha*B*op( A ).

              Unchanged on exit.

       UPLO   - CHARACTER*1.
              On entry, UPLO specifies whether the matrix A is an
              upper or lower triangular matrix as follows:
```

```
              UPLO = 'U' or 'u'   A is an upper triangular matrix.

              UPLO = 'L' or 'l'   A is a lower triangular matrix.

              Unchanged on exit.

              TRANSA - CHARACTER*1.  On entry, TRANSA specifies the
              form of op( A ) to be used in the matrix multiplica-
              tion as follows:

              TRANSA = 'N' or 'n'   op( A ) = A.

              TRANSA = 'T' or 't'   op( A ) = A'.

              TRANSA = 'C' or 'c'   op( A ) = conjg( A' ).

              Unchanged on exit.

       DIAG   - CHARACTER*1.
              On entry, DIAG specifies whether or not A is unit
              triangular as follows:

              DIAG = 'U' or 'u'   A is assumed to be unit triangu-
              lar.

              DIAG = 'N' or 'n'   A is not assumed to be unit tri-
              angular.

              Unchanged on exit.

       M      - INTEGER.
              On entry, M specifies the number of rows of B. M must
              be at least zero.  Unchanged on exit.

       N      - INTEGER.
              On entry, N specifies the number of columns of B.  N
              must be at least zero.  Unchanged on exit.

       ALPHA  - COMPLEX*16      .
              On entry,  ALPHA specifies the scalar  alpha. When
              alpha is zero then  A is not referenced and  B need
              not be set before entry.  Unchanged on exit.

  is m
       A      -
               COMPLEX*16       array of DIMENSION ( LDA, k ), where k
```

```
          when  SIDE = 'L' or 'l'  and is  n  when  SIDE = 'R'
          or 'r'.  Before entry  with  UPLO = 'U' or 'u',  the
          leading  k by k upper triangular part of the array  A
          must contain the upper triangular matrix  and the
          strictly lower triangular part of A is not refer-
          enced.  Before entry  with  UPLO = 'L' or 'l',  the
          leading  k by k lower triangular part of the array  A
          must contain the lower triangular matrix  and the
          strictly upper triangular part of A is not refer-
          enced.  Note that when  DIAG = 'U' or 'u',  the diag-
          onal elements of A  are not referenced either,  but
          are assumed to be  unity.  Unchanged on exit.

  LDA     - INTEGER.
          On entry, LDA specifies the first dimension of A as
          declared in the calling (sub) program.  When  SIDE =
          'L' or 'l'  then LDA  must be at least  max( 1, m ),
          when  SIDE = 'R' or 'r' then LDA must be at least
          max( 1, n ).  Unchanged on exit.

  B       - COMPLEX*16       array of DIMENSION ( LDB, n ).
          Before entry,  the leading  m by n part of the array
          B must contain the matrix  B,  and  on exit  is
          overwritten  by the transformed matrix.

  LDB     - INTEGER.
          On entry, LDB specifies the first dimension of B as
          declared  in  the  calling  (sub)  program.   LDB
          must  be  at  least max( 1, m ).  Unchanged on exit.
```

⟨*BLAS 3 ztrmm*⟩≡

```
(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
 (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun ztrmm (side uplo transa diag m n alpha a lda b ldb$)
    (declare (type (simple-array (complex double-float) (*)) b a)
             (type (complex double-float) alpha)
             (type fixnum ldb$ lda n m)
             (type character diag transa uplo side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (uplo character uplo-%data% uplo-%offset%)
         (transa character transa-%data% transa-%offset%)
         (diag character diag-%data% diag-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (b (complex double-float) b-%data% b-%offset%))
      (prog ((temp #C(0.0 0.0)) (i 0) (info 0) (j 0) (k 0) (nrowa 0)
             (lside nil) (noconj nil) (nounit nil) (upper nil))
        (declare (type (complex double-float) temp)
                 (type fixnum i info j k nrowa)
                 (type (member t nil) lside noconj nounit upper))
        (setf lside (char-equal side #\L))
        (cond
          (lside
           (setf nrowa m))
          (t
           (setf nrowa n)))
        (setf noconj (char-equal transa #\T))
        (setf nounit (char-equal diag #\N))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not lside) (not (char-equal side #\R)))
           (setf info 1))
          ((and (not upper) (not (char-equal uplo #\L)))
           (setf info 2))
          ((and (not (char-equal transa #\N))
                (not (char-equal transa #\T))
                (not (char-equal transa #\C)))
           (setf info 3))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 4))
          ((< m 0)
           (setf info 5))
          ((< n 0)
           (setf info 6))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))
```

```
   (setf info 9))
  ((< ldb$ (max (the fixnum 1) (the fixnum m)))
   (setf info 11)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "ZTRMM" info)
   (go end_label)))
(if (= n 0) (go end_label))
(cond
  ((= alpha zero)
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                 ((> j n) nil)
     (tagbody
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i m) nil)
         (tagbody
           (setf (f2cl-lib:fref b-%data%
                                (i j)
                                ((1 ldb$) (1 *))
                                b-%offset%)
                 zero)))))
   (go end_label)))
(cond
  (lside
   (cond
     ((char-equal transa #\N)
      (cond
        (upper
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                           ((> k m) nil)
               (tagbody
                 (cond
                   ((/= (f2cl-lib:fref b (k j) ((1 ldb$) (1 *))) zero)
                    (setf temp
                            (* alpha
                               (f2cl-lib:fref b-%data%
                                              (k j)
                                              ((1 ldb$) (1 *))
                                              b-%offset%)))
                    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                  ((> i
```

```
                                        (f2cl-lib:int-add k
                                                         (f2cl-lib:int-sub
                                                          1)))
                                    nil)
                              (tagbody
                                (setf (f2cl-lib:fref b-%data%
                                                     (i j)
                                                     ((1 ldb$) (1 *))
                                                     b-%offset%)
                                      (+
                                       (f2cl-lib:fref b-%data%
                                                      (i j)
                                                      ((1 ldb$) (1 *))
                                                      b-%offset%)
                                       (* temp
                                          (f2cl-lib:fref a-%data%
                                                         (i k)
                                                         ((1 lda) (1 *))
                                                         a-%offset%))))))
                            (if nounit
                                (setf temp
                                        (* temp
                                           (f2cl-lib:fref a-%data%
                                                          (k k)
                                                          ((1 lda) (1 *))
                                                          a-%offset%))))
                            (setf (f2cl-lib:fref b-%data%
                                                 (k j)
                                                 ((1 ldb$) (1 *))
                                                 b-%offset%)
                                  temp))))))))
                    (t
                     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                   ((> j n) nil)
                       (tagbody
                         (f2cl-lib:fdo (k m
                                        (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
                                       ((> k 1) nil)
                           (tagbody
                             (cond
                               ((/= (f2cl-lib:fref b (k j) ((1 ldb$) (1 *))) zero)
                                (setf temp
                                        (* alpha
                                           (f2cl-lib:fref b-%data%
                                                          (k j)
                                                          ((1 ldb$) (1 *))
```

```
                                  b-%offset%)))
                        (setf (f2cl-lib:fref b-%data%
                                             (k j)
                                             ((1 ldb$) (1 *))
                                             b-%offset%)
                              temp)
                        (if nounit
                            (setf (f2cl-lib:fref b-%data%
                                                 (k j)
                                                 ((1 ldb$) (1 *))
                                                 b-%offset%)
                                  (*
                                   (f2cl-lib:fref b-%data%
                                                  (k j)
                                                  ((1 ldb$) (1 *))
                                                  b-%offset%)
                                   (f2cl-lib:fref a-%data%
                                                  (k k)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))))
                        (f2cl-lib:fdo (i (f2cl-lib:int-add k 1)
                                       (f2cl-lib:int-add i 1))
                                      ((> i m) nil)
                          (tagbody
                            (setf (f2cl-lib:fref b-%data%
                                                 (i j)
                                                 ((1 ldb$) (1 *))
                                                 b-%offset%)
                                  (+
                                   (f2cl-lib:fref b-%data%
                                                  (i j)
                                                  ((1 ldb$) (1 *))
                                                  b-%offset%)
                                   (* temp
                                      (f2cl-lib:fref a-%data%
                                                     (i k)
                                                     ((1 lda) (1 *))
                                                     a-%offset%)))))))))))
                (t
                 (cond
                   (upper
                    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                  ((> j n) nil)
                      (tagbody
                        (f2cl-lib:fdo (i m
                                       (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
```

```
            ((> i 1) nil)
(tagbody
  (setf temp
          (f2cl-lib:fref b-%data%
                         (i j)
                         ((1 ldb$) (1 *))
                         b-%offset%))
  (cond
    (noconj
     (if nounit
         (setf temp
                 (* temp
                    (f2cl-lib:fref a-%data%
                                   (i i)
                                   ((1 lda) (1 *))
                                   a-%offset%))))
     (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                   ((> k
                       (f2cl-lib:int-add i
                                         (f2cl-lib:int-sub
                                          1)))
                    nil)
       (tagbody
         (setf temp
                 (+ temp
                    (*
                     (f2cl-lib:fref a-%data%
                                    (k i)
                                    ((1 lda) (1 *))
                                    a-%offset%)
                     (f2cl-lib:fref b-%data%
                                    (k j)
                                    ((1 ldb$) (1 *))
                                    b-%offset%)))))))
    (t
     (if nounit
         (setf temp
                 (* temp
                    (f2cl-lib:dconjg
                     (f2cl-lib:fref a-%data%
                                    (i i)
                                    ((1 lda) (1 *))
                                    a-%offset%)))))
     (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                   ((> k
                       (f2cl-lib:int-add i
```

```
                                                    (f2cl-lib:int-sub
                                                     1)))
                                    nil)
                              (tagbody
                                (setf temp
                                      (+ temp
                                         (*
                                          (f2cl-lib:dconjg
                                           (f2cl-lib:fref a-%data%
                                                          (k i)
                                                          ((1 lda) (1 *))
                                                          a-%offset%))
                                          (f2cl-lib:fref b-%data%
                                                         (k j)
                                                         ((1 ldb$) (1 *))
                                                         b-%offset%))))))))
                      (setf (f2cl-lib:fref b-%data%
                                           (i j)
                                           ((1 ldb$) (1 *))
                                           b-%offset%)
                            (* alpha temp)))))))
            (t
             (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                           ((> j n) nil)
               (tagbody
                 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                               ((> i m) nil)
                   (tagbody
                     (setf temp
                             (f2cl-lib:fref b-%data%
                                            (i j)
                                            ((1 ldb$) (1 *))
                                            b-%offset%))
                     (cond
                       (noconj
                        (if nounit
                            (setf temp
                                  (* temp
                                     (f2cl-lib:fref a-%data%
                                                    (i i)
                                                    ((1 lda) (1 *))
                                                    a-%offset%))))
                        (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
                                       (f2cl-lib:int-add k 1))
                                      ((> k m) nil)
                          (tagbody
```

```
                        (setf temp
                                (+ temp
                                   (*
                                    (f2cl-lib:fref a-%data%
                                                   (k i)
                                                   ((1 lda) (1 *))
                                                   a-%offset%)
                                    (f2cl-lib:fref b-%data%
                                                   (k j)
                                                   ((1 ldb$) (1 *))
                                                   b-%offset%)))))))
                    (t
                     (if nounit
                         (setf temp
                                 (* temp
                                    (f2cl-lib:dconjg
                                     (f2cl-lib:fref a-%data%
                                                   (i i)
                                                   ((1 lda) (1 *))
                                                   a-%offset%)))))
                     (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
                                    (f2cl-lib:int-add k 1))
                                   ((> k m) nil)
                       (tagbody
                         (setf temp
                                 (+ temp
                                    (*
                                     (f2cl-lib:dconjg
                                      (f2cl-lib:fref a-%data%
                                                     (k i)
                                                     ((1 lda) (1 *))
                                                     a-%offset%))
                                     (f2cl-lib:fref b-%data%
                                                   (k j)
                                                   ((1 ldb$) (1 *))
                                                   b-%offset%)))))))
                     (setf (f2cl-lib:fref b-%data%
                                          (i j)
                                          ((1 ldb$) (1 *))
                                          b-%offset%)
                           (* alpha temp))))))))
          (t
           (cond
             ((char-equal transa #\N)
              (cond
                (upper
```

```
(f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
              ((> j 1) nil)
  (tagbody
    (setf temp alpha)
    (if nounit
        (setf temp
                (* temp
                   (f2cl-lib:fref a-%data%
                                  (j j)
                                  ((1 lda) (1 *))
                                  a-%offset%))))
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i m) nil)
      (tagbody
        (setf (f2cl-lib:fref b-%data%
                             (i j)
                             ((1 ldb$) (1 *))
                             b-%offset%)
                (* temp
                   (f2cl-lib:fref b-%data%
                                  (i j)
                                  ((1 ldb$) (1 *))
                                  b-%offset%)))))
    (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                  ((> k
                      (f2cl-lib:int-add j
                                        (f2cl-lib:int-sub 1)))
                   nil)
      (tagbody
        (cond
          ((/= (f2cl-lib:fref a (k j) ((1 lda) (1 *))) zero)
           (setf temp
                   (* alpha
                      (f2cl-lib:fref a-%data%
                                     (k j)
                                     ((1 lda) (1 *))
                                     a-%offset%)))
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                         ((> i m) nil)
             (tagbody
               (setf (f2cl-lib:fref b-%data%
                                    (i j)
                                    ((1 ldb$) (1 *))
                                    b-%offset%)
                       (+
                        (f2cl-lib:fref b-%data%
```

```
                                                (i j)
                                                ((1 ldb$) (1 *))
                                                b-%offset%)
                                      (* temp
                                         (f2cl-lib:fref b-%data%
                                                        (i k)
                                                        ((1 ldb$) (1 *))
                                                        b-%offset%)))))))))))))
                    (t
                     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                   ((> j n) nil)
                       (tagbody
                         (setf temp alpha)
                         (if nounit
                             (setf temp
                                     (* temp
                                        (f2cl-lib:fref a-%data%
                                                       (j j)
                                                       ((1 lda) (1 *))
                                                       a-%offset%))))
                         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                       ((> i m) nil)
                           (tagbody
                             (setf (f2cl-lib:fref b-%data%
                                                  (i j)
                                                  ((1 ldb$) (1 *))
                                                  b-%offset%)
                                     (* temp
                                        (f2cl-lib:fref b-%data%
                                                       (i j)
                                                       ((1 ldb$) (1 *))
                                                       b-%offset%)))))
                         (f2cl-lib:fdo (k (f2cl-lib:int-add j 1)
                                          (f2cl-lib:int-add k 1))
                                       ((> k n) nil)
                           (tagbody
                             (cond
                               ((/= (f2cl-lib:fref a (k j) ((1 lda) (1 *))) zero)
                                (setf temp
                                        (* alpha
                                           (f2cl-lib:fref a-%data%
                                                          (k j)
                                                          ((1 lda) (1 *))
                                                          a-%offset%)))
                                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                              ((> i m) nil)
```

```
                                (tagbody
                                  (setf (f2cl-lib:fref b-%data%
                                                       (i j)
                                                       ((1 ldb$) (1 *))
                                                       b-%offset%)
                                        (+
                                         (f2cl-lib:fref b-%data%
                                                        (i j)
                                                        ((1 ldb$) (1 *))
                                                        b-%offset%)
                                         (* temp
                                            (f2cl-lib:fref b-%data%
                                                           (i k)
                                                           ((1 ldb$) (1 *))
                                                           b-%offset%)))))))))))))))))
                        (t
                         (cond
                           (upper
                            (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                                          ((> k n) nil)
                              (tagbody
                                (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                              ((> j
                                                  (f2cl-lib:int-add k
                                                                    (f2cl-lib:int-sub 1)))
                                               nil)
                                  (tagbody
                                    (cond
                                      ((/= (f2cl-lib:fref a (j k) ((1 lda) (1 *))) zero)
                                       (cond
                                         (noconj
                                          (setf temp
                                                  (* alpha
                                                     (f2cl-lib:fref a-%data%
                                                                    (j k)
                                                                    ((1 lda) (1 *))
                                                                    a-%offset%))))
                                         (t
                                          (setf temp
                                                  (* alpha
                                                     (f2cl-lib:dconjg
                                                      (f2cl-lib:fref a-%data%
                                                                     (j k)
                                                                     ((1 lda) (1 *))
                                                                     a-%offset%))))))
                                       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
```

```
                          ((> i m) nil)
                (tagbody
                  (setf (f2cl-lib:fref b-%data%
                                       (i j)
                                       ((1 ldb$) (1 *))
                                       b-%offset%)
                        (+
                         (f2cl-lib:fref b-%data%
                                        (i j)
                                        ((1 ldb$) (1 *))
                                        b-%offset%)
                         (* temp
                            (f2cl-lib:fref b-%data%
                                           (i k)
                                           ((1 ldb$) (1 *))
                                           b-%offset%)))))))))))
(setf temp alpha)
(cond
  (nounit
   (cond
     (noconj
      (setf temp
              (* temp
                 (f2cl-lib:fref a-%data%
                                (k k)
                                ((1 lda) (1 *))
                                a-%offset%))))
     (t
      (setf temp
              (* temp
                 (f2cl-lib:dconjg
                  (f2cl-lib:fref a-%data%
                                 (k k)
                                 ((1 lda) (1 *))
                                 a-%offset%)))))))))
(cond
  ((/= temp one)
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                 ((> i m) nil)
     (tagbody
       (setf (f2cl-lib:fref b-%data%
                            (i k)
                            ((1 ldb$) (1 *))
                            b-%offset%)
             (* temp
                (f2cl-lib:fref b-%data%
```

```
                                           (i k)
                                           ((1 ldb$) (1 *))
                                           b-%offset%))))))))))
                  (t
                   (f2cl-lib:fdo (k n (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
                                 ((> k 1) nil)
                     (tagbody
                       (f2cl-lib:fdo (j (f2cl-lib:int-add k 1)
                                         (f2cl-lib:int-add j 1))
                                        ((> j n) nil)
                         (tagbody
                           (cond
                             ((/= (f2cl-lib:fref a (j k) ((1 lda) (1 *))) zero)
                              (cond
                                (noconj
                                 (setf temp
                                         (* alpha
                                            (f2cl-lib:fref a-%data%
                                                           (j k)
                                                           ((1 lda) (1 *))
                                                           a-%offset%))))
                                (t
                                 (setf temp
                                         (* alpha
                                            (f2cl-lib:dconjg
                                             (f2cl-lib:fref a-%data%
                                                            (j k)
                                                            ((1 lda) (1 *))
                                                            a-%offset%))))))
                              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                            ((> i m) nil)
                                (tagbody
                                  (setf (f2cl-lib:fref b-%data%
                                                       (i j)
                                                       ((1 ldb$) (1 *))
                                                       b-%offset%)
                                          (+
                                           (f2cl-lib:fref b-%data%
                                                          (i j)
                                                          ((1 ldb$) (1 *))
                                                          b-%offset%)
                                           (* temp
                                              (f2cl-lib:fref b-%data%
                                                             (i k)
                                                             ((1 ldb$) (1 *))
                                                             b-%offset%))))))))))
```

```
                        (setf temp alpha)
                        (cond
                          (nounit
                           (cond
                             (noconj
                              (setf temp
                                      (* temp
                                         (f2cl-lib:fref a-%data%
                                                        (k k)
                                                        ((1 lda) (1 *))
                                                        a-%offset%))))
                             (t
                              (setf temp
                                      (* temp
                                         (f2cl-lib:dconjg
                                          (f2cl-lib:fref a-%data%
                                                         (k k)
                                                         ((1 lda) (1 *))
                                                         a-%offset%))))))))
                        (cond
                          ((/= temp one)
                           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                         ((> i m) nil)
                             (tagbody
                               (setf (f2cl-lib:fref b-%data%
                                                    (i k)
                                                    ((1 ldb$) (1 *))
                                                    b-%offset%)
                                       (* temp
                                          (f2cl-lib:fref b-%data%
                                                         (i k)
                                                         ((1 ldb$) (1 *))
                                                         b-%offset%)))))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil nil nil nil))))))
```

6.15 ztrsm BLAS

⟨*ztrsm.input*⟩≡

```
)set break resume
)sys rm -f ztrsm.output
)spool ztrsm.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*ztrsm.help*⟩≡

```
  ======================================================================
  ztrsm examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

   NAME
        ZTRSM - solve one of the matrix equations   op( A )*X =
        alpha*B, or X*op( A ) = alpha*B,

   SYNOPSIS
        SUBROUTINE ZTRSM ( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A,
                           LDA, B, LDB )

            CHARACTER*1  SIDE, UPLO, TRANSA, DIAG

            INTEGER      M, N, LDA, LDB

            COMPLEX*16   ALPHA

            COMPLEX*16   A( LDA, * ), B( LDB, * )

   PURPOSE
        ZTRSM  solves one of the matrix equations

        where alpha is a scalar, X and B are m by n matrices, A is a
        unit, or non-unit,  upper or lower triangular matrix  and
        op( A )  is one  of

           op( A ) = A   or   op( A ) = A'   or   op( A ) = conjg(
        A' ).

        The matrix X is overwritten on B.

   PARAMETERS
        SIDE   - CHARACTER*1.
                 On entry, SIDE specifies whether op( A ) appears on
                 the left or right of X as follows:

                 SIDE = 'L' or 'l'   op( A )*X = alpha*B.

                 SIDE = 'R' or 'r'   X*op( A ) = alpha*B.
```

```
         Unchanged on exit.

UPLO   - CHARACTER*1.
         On entry, UPLO specifies whether the matrix A is an
         upper or lower triangular matrix as follows:

         UPLO = 'U' or 'u'   A is an upper triangular matrix.

         UPLO = 'L' or 'l'   A is a lower triangular matrix.

         Unchanged on exit.

         TRANSA - CHARACTER*1.  On entry, TRANSA specifies the
         form of op( A ) to be used in the matrix

         multiplication as follows:

         TRANSA = 'N' or 'n'   op( A ) = A.

         TRANSA = 'T' or 't'   op( A ) = A'.

         TRANSA = 'C' or 'c'   op( A ) = conjg( A' ).

         Unchanged on exit.

DIAG   - CHARACTER*1.
         On entry, DIAG specifies whether or not A is unit
         triangular as follows:

         DIAG = 'U' or 'u'   A is assumed to be unit triangu-
         lar.

         DIAG = 'N' or 'n'   A is not assumed to be unit tri-
         angular.

         Unchanged on exit.

M      - INTEGER.
         On entry, M specifies the number of rows of B. M must
         be at least zero.  Unchanged on exit.

N      - INTEGER.
         On entry, N specifies the number of columns of B.  N
         must be at least zero.  Unchanged on exit.

ALPHA  - COMPLEX*16      .
```

```
              On entry,  ALPHA specifies the scalar  alpha. When
              alpha is zero then  A is not referenced and  B need
              not be set before entry.  Unchanged on exit.

   is m
        A     -
               COMPLEX*16       array of DIMENSION ( LDA, k ), where k
              when  SIDE = 'L' or 'l'  and is  n  when  SIDE = 'R'
              or 'r'.  Before entry  with  UPLO = 'U' or 'u',  the
              leading  k by k upper triangular part of the array  A
              must contain the upper triangular matrix  and the
              strictly lower triangular part of A is not refer-
              enced.  Before entry  with  UPLO = 'L' or 'l',  the
              leading  k by k lower triangular part of the array  A
              must contain the lower triangular matrix  and the
              strictly upper triangular part of A is not refer-
              enced.  Note that when  DIAG = 'U' or 'u',  the diag-
              onal elements of A  are not referenced either,  but
              are assumed to be  unity.  Unchanged on exit.

        LDA   - INTEGER.

              On entry, LDA specifies the first dimension of A as
              declared in the calling (sub) program.  When  SIDE =
              'L' or 'l'  then LDA  must be at least  max( 1, m ),
              when  SIDE = 'R' or 'r' then LDA must be at least
              max( 1, n ).  Unchanged on exit.

        B     - COMPLEX*16       array of DIMENSION ( LDB, n ).
              Before entry,  the leading  m by n part of the array
              B must contain  the  right-hand  side  matrix  B,
              and  on exit  is overwritten by the solution matrix
              X.

        LDB   - INTEGER.
              On entry, LDB specifies the first dimension of B as
              declared  in  the  calling  (sub)  program.   LDB
              must  be  at  least max( 1, m ).  Unchanged on exit.
```

⟨*BLAS 3 ztrsm*⟩≡

```
(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
 (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun ztrsm (side uplo transa diag m n alpha a lda b ldb$)
    (declare (type (simple-array (complex double-float) (*)) b a)
             (type (complex double-float) alpha)
             (type fixnum ldb$ lda n m)
             (type character diag transa uplo side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (uplo character uplo-%data% uplo-%offset%)
         (transa character transa-%data% transa-%offset%)
         (diag character diag-%data% diag-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (b (complex double-float) b-%data% b-%offset%))
      (prog ((temp #C(0.0 0.0)) (i 0) (info 0) (j 0) (k 0) (nrowa 0)
             (lside nil) (noconj nil) (nounit nil) (upper nil))
        (declare (type (complex double-float) temp)
                 (type fixnum i info j k nrowa)
                 (type (member t nil) lside noconj nounit upper))
        (setf lside (char-equal side #\L))
        (cond
          (lside
           (setf nrowa m))
          (t
           (setf nrowa n)))
        (setf noconj (char-equal transa #\T))
        (setf nounit (char-equal diag #\N))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not lside) (not (char-equal side #\R)))
           (setf info 1))
          ((and (not upper) (not (char-equal uplo #\L)))
           (setf info 2))
          ((and (not (char-equal transa #\N))
                (not (char-equal transa #\T))
                (not (char-equal transa #\C)))
           (setf info 3))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
           (setf info 4))
          ((< m 0)
           (setf info 5))
          ((< n 0)
           (setf info 6))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))
```

```
   (setf info 9))
  ((< ldb$ (max (the fixnum 1) (the fixnum m)))
   (setf info 11)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "ZTRSM" info)
   (go end_label)))
(if (= n 0) (go end_label))
(cond
  ((= alpha zero)
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                 ((> j n) nil)
     (tagbody
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i m) nil)
         (tagbody
           (setf (f2cl-lib:fref b-%data%
                                (i j)
                                ((1 ldb$) (1 *))
                                b-%offset%)
                 zero)))))
   (go end_label)))
(cond
  (lside
   (cond
     ((char-equal transa #\N)
      (cond
        (upper
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (cond
               ((/= alpha one)
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                              ((> i m) nil)
                  (tagbody
                    (setf (f2cl-lib:fref b-%data%
                                         (i j)
                                         ((1 ldb$) (1 *))
                                         b-%offset%)
                          (* alpha
                             (f2cl-lib:fref b-%data%
                                            (i j)
                                            ((1 ldb$) (1 *))
```

```
                                        b-%offset%)))))))
(f2cl-lib:fdo (k m
               (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
              ((> k 1) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref b (k j) ((1 ldb$) (1 *))) zero)
       (if nounit
           (setf (f2cl-lib:fref b-%data%
                                (k j)
                                ((1 ldb$) (1 *))
                                b-%offset%)
                 (/
                  (f2cl-lib:fref b-%data%
                                 (k j)
                                 ((1 ldb$) (1 *))
                                 b-%offset%)
                  (f2cl-lib:fref a-%data%
                                 (k k)
                                 ((1 lda) (1 *))
                                 a-%offset%))))
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                         ((> i
                             (f2cl-lib:int-add k
                                               (f2cl-lib:int-sub
                                                1)))
                          nil)
             (tagbody
               (setf (f2cl-lib:fref b-%data%
                                    (i j)
                                    ((1 ldb$) (1 *))
                                    b-%offset%)
                     (-
                      (f2cl-lib:fref b-%data%
                                     (i j)
                                     ((1 ldb$) (1 *))
                                     b-%offset%)
                      (*
                       (f2cl-lib:fref b-%data%
                                      (k j)
                                      ((1 ldb$) (1 *))
                                      b-%offset%)
                       (f2cl-lib:fref a-%data%
                                      (i k)
                                      ((1 lda) (1 *))
                                      a-%offset%)))))))))))
```

```
(t
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j n) nil)
   (tagbody
     (cond
       ((/= alpha one)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i m) nil)
          (tagbody
            (setf (f2cl-lib:fref b-%data%
                                 (i j)
                                 ((1 ldb$) (1 *))
                                 b-%offset%)
                  (* alpha
                     (f2cl-lib:fref b-%data%
                                    (i j)
                                    ((1 ldb$) (1 *))
                                    b-%offset%)))))))
     (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                   ((> k m) nil)
       (tagbody
         (cond
           ((/= (f2cl-lib:fref b (k j) ((1 ldb$) (1 *))) zero)
            (if nounit
                (setf (f2cl-lib:fref b-%data%
                                     (k j)
                                     ((1 ldb$) (1 *))
                                     b-%offset%)
                      (/
                       (f2cl-lib:fref b-%data%
                                      (k j)
                                      ((1 ldb$) (1 *))
                                      b-%offset%)
                       (f2cl-lib:fref a-%data%
                                      (k k)
                                      ((1 lda) (1 *))
                                      a-%offset%))))
            (f2cl-lib:fdo (i (f2cl-lib:int-add k 1)
                             (f2cl-lib:int-add i 1))
                          ((> i m) nil)
              (tagbody
                (setf (f2cl-lib:fref b-%data%
                                     (i j)
                                     ((1 ldb$) (1 *))
                                     b-%offset%)
                      (-
```

```
                                    (f2cl-lib:fref b-%data%
                                                   (i j)
                                                   ((1 ldb$) (1 *))
                                                   b-%offset%)
                                    (*
                                     (f2cl-lib:fref b-%data%
                                                    (k j)
                                                    ((1 ldb$) (1 *))
                                                    b-%offset%)
                                     (f2cl-lib:fref a-%data%
                                                    (i k)
                                                    ((1 lda) (1 *))
                                                    a-%offset%)))))))))))))))
                (t
                 (cond
                   (upper
                    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                  ((> j n) nil)
                      (tagbody
                        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                      ((> i m) nil)
                          (tagbody
                            (setf temp
                                    (* alpha
                                       (f2cl-lib:fref b-%data%
                                                      (i j)
                                                      ((1 ldb$) (1 *))
                                                      b-%offset%)))
                            (cond
                              (noconj
                               (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                                             ((> k
                                                 (f2cl-lib:int-add i
                                                                   (f2cl-lib:int-sub
                                                                    1)))
                                              nil)
                                 (tagbody
                                   (setf temp
                                           (- temp
                                              (*
                                               (f2cl-lib:fref a-%data%
                                                              (k i)
                                                              ((1 lda) (1 *))
                                                              a-%offset%)
                                               (f2cl-lib:fref b-%data%
                                                              (k j)
```

```
                                                ((1 ldb$) (1 *))
                                                b-%offset%))))))
                        (if nounit
                            (setf temp
                                    (/ temp
                                       (f2cl-lib:fref a-%data%
                                                      (i i)
                                                      ((1 lda) (1 *))
                                                      a-%offset%)))))
                       (t
                        (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                                      ((> k
                                          (f2cl-lib:int-add i
                                                            (f2cl-lib:int-sub
                                                             1)))
                                       nil)
                          (tagbody
                            (setf temp
                                    (- temp
                                       (*
                                        (f2cl-lib:dconjg
                                         (f2cl-lib:fref a-%data%
                                                        (k i)
                                                        ((1 lda) (1 *))
                                                        a-%offset%))
                                        (f2cl-lib:fref b-%data%
                                                       (k j)
                                                       ((1 ldb$) (1 *))
                                                       b-%offset%))))))
                            (if nounit
                                (setf temp
                                        (/ temp
                                           (f2cl-lib:dconjg
                                            (f2cl-lib:fref a-%data%
                                                           (i i)
                                                           ((1 lda) (1 *))
                                                           a-%offset%)))))))
                        (setf (f2cl-lib:fref b-%data%
                                             (i j)
                                             ((1 ldb$) (1 *))
                                             b-%offset%)
                              temp))))))
                  (t
                   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                 ((> j n) nil)
                     (tagbody
```

```
(f2cl-lib:fdo (i m
               (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
              ((> i 1) nil)
  (tagbody
    (setf temp
            (* alpha
               (f2cl-lib:fref b-%data%
                              (i j)
                              ((1 ldb$) (1 *))
                              b-%offset%)))
    (cond
      (noconj
       (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
                        (f2cl-lib:int-add k 1))
                     ((> k m) nil)
         (tagbody
           (setf temp
                   (- temp
                      (*
                       (f2cl-lib:fref a-%data%
                                      (k i)
                                      ((1 lda) (1 *))
                                      a-%offset%)
                       (f2cl-lib:fref b-%data%
                                      (k j)
                                      ((1 ldb$) (1 *))
                                      b-%offset%))))))
       (if nounit
           (setf temp
                   (/ temp
                      (f2cl-lib:fref a-%data%
                                     (i i)
                                     ((1 lda) (1 *))
                                     a-%offset%)))))
      (t
       (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
                        (f2cl-lib:int-add k 1))
                     ((> k m) nil)
         (tagbody
           (setf temp
                   (- temp
                      (*
                       (f2cl-lib:dconjg
                        (f2cl-lib:fref a-%data%
                                       (k i)
                                       ((1 lda) (1 *))
```

```
                                                    a-%offset%))
                                          (f2cl-lib:fref b-%data%
                                                         (k j)
                                                         ((1 ldb$) (1 *))
                                                         b-%offset%))))))
                             (if nounit
                                 (setf temp
                                         (/ temp
                                            (f2cl-lib:dconjg
                                             (f2cl-lib:fref a-%data%
                                                            (i i)
                                                            ((1 lda) (1 *))
                                                            a-%offset%)))))))
                         (setf (f2cl-lib:fref b-%data%
                                              (i j)
                                              ((1 ldb$) (1 *))
                                              b-%offset%)
                               temp))))))))))
        (t
         (cond
           ((char-equal transa #\N)
            (cond
              (upper
               (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                             ((> j n) nil)
                 (tagbody
                   (cond
                     ((/= alpha one)
                      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                    ((> i m) nil)
                        (tagbody
                          (setf (f2cl-lib:fref b-%data%
                                               (i j)
                                               ((1 ldb$) (1 *))
                                               b-%offset%)
                                (* alpha
                                   (f2cl-lib:fref b-%data%
                                                  (i j)
                                                  ((1 ldb$) (1 *))
                                                  b-%offset%)))))))
                     (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                                   ((> k
                                       (f2cl-lib:int-add j
                                                         (f2cl-lib:int-sub 1)))
                                    nil)
                       (tagbody
```

```
                    (cond
                      ((/= (f2cl-lib:fref a (k j) ((1 lda) (1 *))) zero)
                       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                     ((> i m) nil)
                         (tagbody
                           (setf (f2cl-lib:fref b-%data%
                                                (i j)
                                                ((1 ldb$) (1 *))
                                                b-%offset%)
                                 (-
                                  (f2cl-lib:fref b-%data%
                                                 (i j)
                                                 ((1 ldb$) (1 *))
                                                 b-%offset%)
                                  (*
                                   (f2cl-lib:fref a-%data%
                                                  (k j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%)
                                   (f2cl-lib:fref b-%data%
                                                  (i k)
                                                  ((1 ldb$) (1 *))
                                                  b-%offset%)))))))))
                (cond
                  (nounit
                   (setf temp
                         (/ one
                            (f2cl-lib:fref a-%data%
                                           (j j)
                                           ((1 lda) (1 *))
                                           a-%offset%)))
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i m) nil)
                     (tagbody
                       (setf (f2cl-lib:fref b-%data%
                                            (i j)
                                            ((1 ldb$) (1 *))
                                            b-%offset%)
                             (* temp
                                (f2cl-lib:fref b-%data%
                                               (i j)
                                               ((1 ldb$) (1 *))
                                               b-%offset%)))))))))
          (t
           (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                         ((> j 1) nil)
```

```
(tagbody
  (cond
    ((/= alpha one)
     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                   ((> i m) nil)
       (tagbody
         (setf (f2cl-lib:fref b-%data%
                              (i j)
                              ((1 ldb$) (1 *))
                              b-%offset%)
                 (* alpha
                    (f2cl-lib:fref b-%data%
                                   (i j)
                                   ((1 ldb$) (1 *))
                                   b-%offset%)))))))
  (f2cl-lib:fdo (k (f2cl-lib:int-add j 1)
                 (f2cl-lib:int-add k 1))
                ((> k n) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref a (k j) ((1 lda) (1 *))) zero)
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i m) nil)
           (tagbody
             (setf (f2cl-lib:fref b-%data%
                                  (i j)
                                  ((1 ldb$) (1 *))
                                  b-%offset%)
                     (-
                      (f2cl-lib:fref b-%data%
                                     (i j)
                                     ((1 ldb$) (1 *))
                                     b-%offset%)
                      (*
                       (f2cl-lib:fref a-%data%
                                      (k j)
                                      ((1 lda) (1 *))
                                      a-%offset%)
                       (f2cl-lib:fref b-%data%
                                      (i k)
                                      ((1 ldb$) (1 *))
                                      b-%offset%)))))))))))
  (cond
    (nounit
     (setf temp
             (/ one
```

```
                                    (f2cl-lib:fref a-%data%
                                                   (j j)
                                                   ((1 lda) (1 *))
                                                   a-%offset%)))
                            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                          ((> i m) nil)
                              (tagbody
                                (setf (f2cl-lib:fref b-%data%
                                                     (i j)
                                                     ((1 ldb$) (1 *))
                                                     b-%offset%)
                                      (* temp
                                         (f2cl-lib:fref b-%data%
                                                        (i j)
                                                        ((1 ldb$) (1 *))
                                                        b-%offset%)))))))))))
                    (t
                     (cond
                       (upper
                        (f2cl-lib:fdo (k n (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
                                      ((> k 1) nil)
                          (tagbody
                            (cond
                              (nounit
                               (cond
                                 (noconj
                                  (setf temp
                                          (/ one
                                             (f2cl-lib:fref a-%data%
                                                            (k k)
                                                            ((1 lda) (1 *))
                                                            a-%offset%))))
                                 (t
                                  (setf temp
                                          (/ one
                                             (f2cl-lib:dconjg
                                              (f2cl-lib:fref a-%data%
                                                             (k k)
                                                             ((1 lda) (1 *))
                                                             a-%offset%))))))
                                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                              ((> i m) nil)
                                  (tagbody
                                    (setf (f2cl-lib:fref b-%data%
                                                         (i k)
                                                         ((1 ldb$) (1 *))
```

```
                                        b-%offset%)
                          (* temp
                             (f2cl-lib:fref b-%data%
                                            (i k)
                                            ((1 ldb$) (1 *))
                                            b-%offset%)))))))
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                      ((> j
                          (f2cl-lib:int-add k
                                            (f2cl-lib:int-sub 1)))
                       nil)
          (tagbody
            (cond
              ((/= (f2cl-lib:fref a (j k) ((1 lda) (1 *))) zero)
               (cond
                 (noconj
                  (setf temp
                          (f2cl-lib:fref a-%data%
                                         (j k)
                                         ((1 lda) (1 *))
                                         a-%offset%)))
                 (t
                  (setf temp
                          (coerce
                           (f2cl-lib:dconjg
                            (f2cl-lib:fref a-%data%
                                           (j k)
                                           ((1 lda) (1 *))
                                           a-%offset%))
                           '(complex double-float)))))
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i m) nil)
                 (tagbody
                   (setf (f2cl-lib:fref b-%data%
                                        (i j)
                                        ((1 ldb$) (1 *))
                                        b-%offset%)
                           (-
                            (f2cl-lib:fref b-%data%
                                           (i j)
                                           ((1 ldb$) (1 *))
                                           b-%offset%)
                            (* temp
                               (f2cl-lib:fref b-%data%
                                              (i k)
                                              ((1 ldb$) (1 *))
```

```
                                                  b-%offset%))))))))))
                (cond
                  ((/= alpha one)
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i m) nil)
                     (tagbody
                       (setf (f2cl-lib:fref b-%data%
                                            (i k)
                                            ((1 ldb$) (1 *))
                                            b-%offset%)
                             (* alpha
                                (f2cl-lib:fref b-%data%
                                               (i k)
                                               ((1 ldb$) (1 *))
                                               b-%offset%)))))))))
           (t
            (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                          ((> k n) nil)
              (tagbody
                (cond
                  (nounit
                   (cond
                     (noconj
                      (setf temp
                              (/ one
                                 (f2cl-lib:fref a-%data%
                                                (k k)
                                                ((1 lda) (1 *))
                                                a-%offset%))))
                     (t
                      (setf temp
                              (/ one
                                 (f2cl-lib:dconjg
                                  (f2cl-lib:fref a-%data%
                                                 (k k)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))))))
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i m) nil)
                     (tagbody
                       (setf (f2cl-lib:fref b-%data%
                                            (i k)
                                            ((1 ldb$) (1 *))
                                            b-%offset%)
                             (* temp
                                (f2cl-lib:fref b-%data%
```

```
                                    (i k)
                                    ((1 ldb$) (1 *))
                                    b-%offset%)))))))
(f2cl-lib:fdo (j (f2cl-lib:int-add k 1)
                 (f2cl-lib:int-add j 1))
               ((> j n) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref a (j k) ((1 lda) (1 *))) zero)
       (cond
         (noconj
          (setf temp
                  (f2cl-lib:fref a-%data%
                                 (j k)
                                 ((1 lda) (1 *))
                                 a-%offset%)))
         (t
          (setf temp
                  (coerce
                   (f2cl-lib:dconjg
                    (f2cl-lib:fref a-%data%
                                   (j k)
                                   ((1 lda) (1 *))
                                   a-%offset%))
                   '(complex double-float)))))
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i m) nil)
         (tagbody
           (setf (f2cl-lib:fref b-%data%
                                (i j)
                                ((1 ldb$) (1 *))
                                b-%offset%)
                   (-
                    (f2cl-lib:fref b-%data%
                                   (i j)
                                   ((1 ldb$) (1 *))
                                   b-%offset%)
                    (* temp
                       (f2cl-lib:fref b-%data%
                                      (i k)
                                      ((1 ldb$) (1 *))
                                      b-%offset%)))))))))
(cond
  ((/= alpha one)
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                 ((> i m) nil)
```

```
                              (tagbody
                                (setf (f2cl-lib:fref b-%data%
                                                     (i k)
                                                     ((1 ldb$) (1 *))
                                                     b-%offset%)
                                      (* alpha
                                         (f2cl-lib:fref b-%data%
                                                        (i k)
                                                        ((1 ldb$) (1 *))
                                                        b-%offset%)))))))))))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil nil nil nil))))))
```

Chapter 7

LAPACK

7.1 dbdsdc LAPACK

⟨*dbdsdc.input*⟩≡
```
)set break resume
)sys rm -f dbdsdc.output
)spool dbdsdc.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dbdsdc.help*⟩≡

```
======================================================================
dbdsdc examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DBDSDC - the singular value decomposition (SVD) of a real N-by-N (upper
       or lower) bidiagonal matrix B

SYNOPSIS
       SUBROUTINE DBDSDC( UPLO, COMPQ, N, D, E, U, LDU, VT, LDVT, Q, IQ, WORK,
                          IWORK, INFO )

           CHARACTER      COMPQ, UPLO

           INTEGER        INFO, LDU, LDVT, N

           INTEGER        IQ( * ), IWORK( * )

           DOUBLE         PRECISION  D(  * ), E( * ), Q( * ), U( LDU, * ), VT(
                          LDVT, * ), WORK( * )

PURPOSE
       DBDSDC computes the singular value decomposition (SVD) of a real N-by-N
       (upper  or  lower) bidiagonal matrix B:  B = U * S * VT, using a divide
       and conquer method, where S is  a  diagonal  matrix  with  non-negative
       diagonal elements (the singular values of B), and U and VT are orthogo-
       nal matrices of left and right singular vectors,  respectively.  DBDSDC
       can  be  used  to compute all singular values, and optionally, singular
       vectors or singular vectors in compact form.

       This code makes very mild assumptions about floating point  arithmetic.
       It  will  work  on  machines  with a guard digit in add/subtract, or on
       those binary machines without guard digits which subtract like the Cray
       X-MP,  Cray  Y-MP,  Cray C-90, or Cray-2.  It could conceivably fail on
       hexadecimal or decimal machines without guard digits, but  we  know  of
       none.  See DLASD3 for details.

       The  code  currently  calls DLASDQ if singular values only are desired.
       However, it can be slightly modified to compute singular  values  using
       the divide and conquer method.
```

```
ARGUMENTS
       UPLO    (input) CHARACTER*1
               = 'U':  B is upper bidiagonal.
               = 'L':  B is lower bidiagonal.

       COMPQ   (input) CHARACTER*1
               Specifies  whether  singular vectors are to be computed as fol-
               lows:
               = 'N':  Compute singular values only;
               = 'P':  Compute singular values and compute singular vectors in
               compact form; = 'I':  Compute singular values and singular vec-
               tors.

       N       (input) INTEGER
               The order of the matrix B.  N >= 0.

       D       (input/output) DOUBLE PRECISION array, dimension (N)
               On entry, the n diagonal elements of the bidiagonal  matrix  B.
               On exit, if INFO=0, the singular values of B.

       E       (input/output) DOUBLE PRECISION array, dimension (N-1)
               On entry, the elements of E contain the offdiagonal elements of
               the bidiagonal matrix whose SVD is desired.   On  exit,  E  has
               been destroyed.

       U       (output) DOUBLE PRECISION array, dimension (LDU,N)
               If   COMPQ  =  'I',  then: On exit, if INFO = 0, U contains the
               left singular vectors of the bidiagonal matrix.  For other val-
               ues of COMPQ, U is not referenced.

       LDU     (input) INTEGER
               The  leading  dimension of the array U.  LDU >= 1.  If singular
               vectors are desired, then LDU >= max( 1, N ).

       VT      (output) DOUBLE PRECISION array, dimension (LDVT,N)
               If  COMPQ = 'I', then: On exit, if INFO = 0, VT'  contains  the
               right  singular  vectors  of  the bidiagonal matrix.  For other
               values of COMPQ, VT is not referenced.

       LDVT    (input) INTEGER
               The leading dimension of the array VT.  LDVT >= 1.  If singular
               vectors are desired, then LDVT >= max( 1, N ).

       Q       (output) DOUBLE PRECISION array, dimension (LDQ)
               If   COMPQ  = 'P', then: On exit, if INFO = 0, Q and IQ contain
```

```
          the left and right singular vectors in a compact form,  requir-
          ing  O(N log N) space instead of 2*N**2.  In particular, Q con-
          tains all the DOUBLE PRECISION data in LDQ >= N*(11 +  2*SMLSIZ
          +  8*INT(LOG_2(N/(SMLSIZ+1)))) words of memory, where SMLSIZ is
          returned by ILAENV and is equal to the maximum size of the sub-
          problems  at  the bottom of the computation tree (usually about
          25).  For other values of COMPQ, Q is not referenced.

IQ        (output) INTEGER array, dimension (LDIQ)
          If  COMPQ = 'P', then: On exit, if INFO = 0, Q and  IQ  contain
          the  left and right singular vectors in a compact form, requir-
          ing O(N log N) space instead of 2*N**2.  In particular, IQ con-
          tains  all  INTEGER  data in LDIQ >= N*(3 + 3*INT(LOG_2(N/(SML-
          SIZ+1)))) words of memory, where SMLSIZ is returned  by  ILAENV
          and is equal to the maximum size of the subproblems at the bot-
          tom of the computation tree (usually about 25).  For other val-
          ues of COMPQ, IQ is not referenced.

WORK      (workspace) DOUBLE PRECISION array, dimension (MAX(1,LWORK))
          If  COMPQ  =  'N'  then  LWORK >= (4 * N).  If COMPQ = 'P' then
          LWORK >= (6 * N).  If COMPQ = 'I' then LWORK >= (3 * N**2 + 4 *
          N).

IWORK     (workspace) INTEGER array, dimension (8*N)

INFO      (output) INTEGER
          = 0:  successful exit.
          < 0:  if INFO = -i, the i-th argument had an illegal value.
          >  0:   The algorithm failed to compute an singular value.  The
          update process of divide and conquer failed.
```

The input arguments are:

- uplo - simple-array character (1)
- compq - (simple-array character (1)
- n - fixnum
- d - array doublefloat
- e - array doublefloat
- u - array doublefloat
- ldu - fixnum
- vt - doublefloat
- ldvt - fixnum
- q - array doublefloat
- iq - array fixnum
- work - array doublefloat
- iwork - array fixnum
- info - fixnum

The return values are:

- uplo - nil
- compq - nil
- n - nil
- d - nil
- e - nil
- u - nil
- ldu - nil
- vt - nil
- ldvt - nil
- q - nil
- iq - nil

- work - nil
- iwork - nil
- info - info

[dlasr p??]
[dswap p??]
[dlasda p??]
[dlasd0 p??]
[dlamch p??]
[dlascl p??]
[dlanst p??]
[dlaset p??]
[dlasdq p??]
[dlartg p??]
[dcopy p??]
[ilaenv p??]
[xerbla p??]
[char-equal p??]

⟨*LAPACK dbdsdc*⟩≡

```
(let* ((zero 0.0) (one 1.0) (two 2.0))
 (declare (type (double-float 0.0 0.0) zero)
          (type (double-float 1.0 1.0) one)
          (type (double-float 2.0 2.0) two))
 (defun dbdsdc (uplo compq n d e u ldu vt ldvt q iq work iwork info)
  (declare (type (simple-array fixnum (*)) iwork iq)
           (type (simple-array double-float (*)) work q vt u e d)
           (type fixnum info ldvt ldu n)
           (type character compq uplo))
   (f2cl-lib:with-multi-array-data
    ((uplo character uplo-%data% uplo-%offset%)
     (compq character compq-%data% compq-%offset%)
     (d double-float d-%data% d-%offset%)
     (e double-float e-%data% e-%offset%)
     (u double-float u-%data% u-%offset%)
     (vt double-float vt-%data% vt-%offset%)
     (q double-float q-%data% q-%offset%)
     (work double-float work-%data% work-%offset%)
     (iq fixnum iq-%data% iq-%offset%)
     (iwork fixnum iwork-%data% iwork-%offset%))
    (prog ((cs 0.0) (eps 0.0) (orgnrm 0.0) (p 0.0) (r 0.0) (sn 0.0) (difl 0)
           (difr 0) (givcol 0) (givnum 0) (givptr 0) (i 0) (ic 0) (icompq 0)
           (ierr 0) (ii 0) (is 0) (iu 0) (iuplo 0) (ivt 0) (j 0) (k 0) (kk 0)
           (mlvl 0) (nm1 0) (nsize 0) (perm 0) (poles 0) (qstart 0)
```

```
      (smlsiz 0) (smlszp 0) (sqre 0) (start 0) (wstart 0) (z 0))
  (declare (type (double-float) cs eps orgnrm p r sn)
           (type fixnum difl difr givcol givnum givptr i ic
                                  icompq ierr ii is iu iuplo ivt j k
                                  kk mlvl nm1 nsize perm poles qstart
                                  smlsiz smlszp sqre start wstart z))
  (setf info 0)
  (setf iuplo 0)
  (if (char-equal uplo #\U) (setf iuplo 1))
  (if (char-equal uplo #\L) (setf iuplo 2))
  (cond
   ((char-equal compq #\N) (setf icompq 0))
   ((char-equal compq #\P) (setf icompq 1))
   ((char-equal compq #\I) (setf icompq 2))
   (t                 (setf icompq -1)))
  (cond
   ((= iuplo 0)                                      (setf info -1))
   ((< icompq 0)                                     (setf info -2))
   ((< n 0)                                          (setf info -3))
   ((or (< ldu 1) (and (= icompq 2) (< ldu n)))   (setf info -7))
   ((or (< ldvt 1) (and (= icompq 2) (< ldvt n))) (setf info -9)))
  (cond
   ((/= info 0)
    (error
     " ** On entry to ~a parameter number ~a had an illegal value~%"
     "DBDSDC" (f2cl-lib:int-sub info))
    (go end_label)))
  (if (= n 0) (go end_label))
  (setf smlsiz (ilaenv 9 "DBDSDC" " " 0 0 0 0))
  (cond
   ((= n 1)
    (cond
     ((= icompq 1)
      (setf
       (f2cl-lib:fref q-%data% (1) ((1 *)) q-%offset%)
       (f2cl-lib:sign one
        (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)))
      (setf
       (f2cl-lib:fref q-%data%
        ((f2cl-lib:int-add 1
          (f2cl-lib:int-mul smlsiz n))) ((1 *)) q-%offset%)
       one))
     ((= icompq 2)
      (setf
       (f2cl-lib:fref u-%data% (1 1) ((1 ldu) (1 *)) u-%offset%)
       (f2cl-lib:sign one (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)))
```

```
    (setf
      (f2cl-lib:fref vt-%data% (1 1) ((1 ldvt) (1 *)) vt-%offset%)
      one)))
  (setf (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)
        (abs (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)))
  (go end_label)))
(setf nm1 (f2cl-lib:int-sub n 1))
(setf wstart 1)
(setf qstart 3)
(cond
 ((= icompq 1)
  (dcopy n d 1 (f2cl-lib:array-slice q double-float (1) ((1 *))) 1)
  (dcopy (f2cl-lib:int-sub n 1) e 1
    (f2cl-lib:array-slice q double-float ((+ n 1)) ((1 *))) 1)))
(cond
 ((= iuplo 2)
  (setf qstart 5)
  (setf wstart (f2cl-lib:int-sub (f2cl-lib:int-mul 2 n) 1))
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
   (tagbody
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
     (dlartg (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
             (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%) cs sn r)
      (declare (ignore var-0 var-1))
      (setf cs var-2)
      (setf sn var-3)
      (setf r var-4))
      (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) r)
      (setf
       (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
       (* sn (f2cl-lib:fref
              d-%data% ((f2cl-lib:int-add i 1)) ((1 *)) d-%offset%)))
      (setf
       (f2cl-lib:fref d-%data%
         ((f2cl-lib:int-add i 1)) ((1 *)) d-%offset%)
       (* cs (f2cl-lib:fref
              d-%data% ((f2cl-lib:int-add i 1)) ((1 *)) d-%offset%)))
      (cond
       ((= icompq 1)
        (setf
         (f2cl-lib:fref q-%data%
          ((f2cl-lib:int-add i (f2cl-lib:int-mul 2 n)))
           ((1 *)) q-%offset%)
         cs)
        (setf
```

```
            (f2cl-lib:fref q-%data%
             ((f2cl-lib:int-add i (f2cl-lib:int-mul 3 n)))
              ((1 *)) q-%offset%)
            sn))
          ((= icompq 2)
           (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%) cs)
           (setf
            (f2cl-lib:fref work-%data%
             ((f2cl-lib:int-add nm1 i)) ((1 *)) work-%offset%)
            (- sn))))))))
(cond
 ((= icompq 0)
  (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
    var-10 var-11 var-12 var-13 var-14 var-15)
   (dlasdq "U" 0 n 0 0 0 d e vt ldvt u ldu u ldu
    (f2cl-lib:array-slice work double-float (wstart) ((1 *))) info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                    var-8 var-9 var-10 var-11 var-12 var-13 var-14))
   (setf info var-15))
  (go label40)))
(cond
 ((<= n smlsiz)
  (cond
   ((= icompq 2)
    (dlaset "A" n n zero one u ldu)
    (dlaset "A" n n zero one vt ldvt)
    (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
      var-9 var-10 var-11 var-12 var-13 var-14 var-15)
     (dlasdq "U" 0 n n n 0 d e vt ldvt u ldu u ldu
      (f2cl-lib:array-slice work double-float (wstart) ((1 *)))
      info)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7 var-8 var-9 var-10 var-11 var-12 var-13 var-14))
     (setf info var-15)))
   ((= icompq 1)
    (setf iu 1)
    (setf ivt (f2cl-lib:int-add iu n))
    (dlaset "A" n n zero one
     (f2cl-lib:array-slice q double-float
      ((+ iu
          (f2cl-lib:int-mul
           (f2cl-lib:int-add qstart (f2cl-lib:int-sub 1)) n)))
      ((1 *)))
     n)
```

```
          (dlaset "A" n n zero one
           (f2cl-lib:array-slice q double-float
            ((+ ivt
                (f2cl-lib:int-mul
                 (f2cl-lib:int-add qstart (f2cl-lib:int-sub 1)) n)))
            ((1 *)))
           n)
          (multiple-value-bind
           (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
            var-9 var-10 var-11 var-12 var-13 var-14 var-15)
           (dlasdq "U" 0 n n n 0 d e
            (f2cl-lib:array-slice q double-float
             ((+ ivt
                 (f2cl-lib:int-mul
                  (f2cl-lib:int-add qstart (f2cl-lib:int-sub 1))
                  n)))
             ((1 *)))
            n
            (f2cl-lib:array-slice q double-float
             ((+ iu
                 (f2cl-lib:int-mul
                  (f2cl-lib:int-add qstart (f2cl-lib:int-sub 1))
                  n)))
             ((1 *)))
            n
            (f2cl-lib:array-slice q double-float
             ((+ iu
                 (f2cl-lib:int-mul
                  (f2cl-lib:int-add qstart (f2cl-lib:int-sub 1))
                  n)))
             ((1 *)))
            n
            (f2cl-lib:array-slice work double-float (wstart) ((1 *)))
            info)
           (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                     var-7 var-8 var-9 var-10 var-11 var-12 var-13 var-14))
           (setf info var-15))))
          (go label40)))
     (cond
      ((= icompq 2)
       (dlaset "A" n n zero one u ldu)
       (dlaset "A" n n zero one vt ldvt)))
     (setf orgnrm (dlanst "M" n d e))
     (if (= orgnrm zero) (go end_label))
     (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
```

```
  (dlascl "G" 0 0 orgnrm one n 1 d n ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
            var-8))
  (setf ierr var-9))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
 (dlascl "G" 0 0 orgnrm one nm1 1 e nm1 ierr)
 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
           var-8))
 (setf ierr var-9))
(setf eps (dlamch "Epsilon"))
(setf mlvl
 (f2cl-lib:int-add
  (f2cl-lib:int
   (/
    (f2cl-lib:flog
     (/ (coerce (realpart n) 'double-float)
        (coerce (realpart (f2cl-lib:int-add smlsiz 1)) 'double-float)))
    (f2cl-lib:flog two)))
   1))
(setf smlszp (f2cl-lib:int-add smlsiz 1))
(cond
 ((= icompq 1)
  (setf iu 1)
  (setf ivt (f2cl-lib:int-add 1 smlsiz))
  (setf difl (f2cl-lib:int-add ivt smlszp))
  (setf difr (f2cl-lib:int-add difl mlvl))
  (setf z (f2cl-lib:int-add difr (f2cl-lib:int-mul mlvl 2)))
  (setf ic (f2cl-lib:int-add z mlvl))
  (setf is (f2cl-lib:int-add ic 1))
  (setf poles (f2cl-lib:int-add is 1))
  (setf givnum (f2cl-lib:int-add poles (f2cl-lib:int-mul 2 mlvl)))
  (setf k 1)
  (setf givptr 2)
  (setf perm 3)
   (setf givcol (f2cl-lib:int-add perm mlvl))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i n) nil)
 (tagbody
  (cond
   ((< (abs (f2cl-lib:fref d (i) ((1 *)))) eps)
    (setf
     (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
     (f2cl-lib:sign eps
      (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)))))))
(setf start 1)
```

```
(setf sqre 0)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i nm1) nil)
 (tagbody
  (cond
   ((or (< (abs (f2cl-lib:fref e (i) ((1 *)))) eps) (= i nm1))
    (cond
     ((< i nm1)
      (setf nsize (f2cl-lib:int-add (f2cl-lib:int-sub i start) 1)))
     ((>= (abs (f2cl-lib:fref e (i) ((1 *)))) eps)
      (setf nsize (f2cl-lib:int-add (f2cl-lib:int-sub n start) 1)))
     (t
      (setf nsize (f2cl-lib:int-add (f2cl-lib:int-sub i start) 1))
      (cond
       ((= icompq 2)
        (setf
         (f2cl-lib:fref u-%data% (n n) ((1 ldu) (1 *)) u-%offset%)
         (f2cl-lib:sign one
          (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)))
        (setf
         (f2cl-lib:fref vt-%data% (n n) ((1 ldvt) (1 *)) vt-%offset%)
         one))
       ((= icompq 1)
        (setf
         (f2cl-lib:fref q-%data%
          ((f2cl-lib:int-add n
            (f2cl-lib:int-mul (f2cl-lib:int-sub qstart 1) n)))
           ((1 *)) q-%offset%)
         (f2cl-lib:sign one
          (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)))
        (setf
         (f2cl-lib:fref q-%data%
          ((f2cl-lib:int-add n
            (f2cl-lib:int-mul
             (f2cl-lib:int-sub
              (f2cl-lib:int-add smlsiz qstart) 1) n)))
          ((1 *)) q-%offset%)
         one)))
        (setf
         (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
         (abs (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)))))
    (cond
     ((= icompq 2)
      (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11)
```

```
  (dlasd0 nsize sqre
   (f2cl-lib:array-slice d double-float (start) ((1 *)))
   (f2cl-lib:array-slice e double-float (start) ((1 *)))
   (f2cl-lib:array-slice u double-float
    (start start) ((1 ldu) (1 *)))
   ldu
   (f2cl-lib:array-slice vt double-float
    (start start) ((1 ldvt) (1 *)))
   ldvt
   smlsiz
   iwork
   (f2cl-lib:array-slice work double-float (wstart) ((1 *))) info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
            var-7 var-8 var-9 var-10))
  (setf info var-11)))
(t
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13 var-14 var-15 var-16
   var-17 var-18 var-19 var-20 var-21 var-22 var-23)
  (dlasda icompq smlsiz nsize sqre
   (f2cl-lib:array-slice d double-float (start) ((1 *)))
   (f2cl-lib:array-slice e double-float (start) ((1 *)))
   (f2cl-lib:array-slice q double-float
    ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add iu qstart
      (f2cl-lib:int-sub 2)) n))) ((1 *)))
   n
   (f2cl-lib:array-slice q double-float
    ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add ivt qstart
      (f2cl-lib:int-sub 2)) n))) ((1 *)))
   (f2cl-lib:array-slice iq fixnum
    ((+ start (f2cl-lib:int-mul k n))) ((1 *)))
   (f2cl-lib:array-slice q double-float
    ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add difl qstart
               (f2cl-lib:int-sub 2)) n))) ((1 *)))
   (f2cl-lib:array-slice q double-float
    ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add difr qstart
               (f2cl-lib:int-sub 2)) n))) ((1 *)))
   (f2cl-lib:array-slice q double-float
    ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add z qstart
               (f2cl-lib:int-sub 2)) n))) ((1 *)))
   (f2cl-lib:array-slice q double-float
    ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add poles qstart
               (f2cl-lib:int-sub 2)) n))) ((1 *)))
   (f2cl-lib:array-slice iq fixnum
    ((+ start (f2cl-lib:int-mul givptr n))) ((1 *)))
```

```
              (f2cl-lib:array-slice iq fixnum
               ((+ start (f2cl-lib:int-mul givcol n))) ((1 *)))
              n
              (f2cl-lib:array-slice iq fixnum
               ((+ start (f2cl-lib:int-mul perm n))) ((1 *)))
              (f2cl-lib:array-slice q double-float
               ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add givnum qstart
                          (f2cl-lib:int-sub 2)) n))) ((1 *)))
              (f2cl-lib:array-slice q double-float
               ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add ic qstart
                          (f2cl-lib:int-sub 2)) n))) ((1 *)))
              (f2cl-lib:array-slice q double-float
               ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add is qstart
                          (f2cl-lib:int-sub 2)) n))) ((1 *)))
              (f2cl-lib:array-slice work double-float (wstart) ((1 *)))
              iwork
              info)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                               var-7 var-8 var-9 var-10 var-11 var-12
                               var-13 var-14 var-15 var-16 var-17 var-18
                               var-19 var-20 var-21 var-22))
              (setf info var-23))
            (cond
             ((/= info 0) (go end_label)))))
          (setf start (f2cl-lib:int-add i 1))))))
     (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
      (dlascl "G" 0 0 one orgnrm n 1 d n ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8))
      (setf ierr var-9))
label40
     (f2cl-lib:fdo (ii 2 (f2cl-lib:int-add ii 1))
                   ((> ii n) nil)
      (tagbody
       (setf i (f2cl-lib:int-sub ii 1))
       (setf kk i)
       (setf p (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
       (f2cl-lib:fdo (j ii (f2cl-lib:int-add j 1))
                     ((> j n) nil)
        (tagbody
         (cond
          ((> (f2cl-lib:fref d (j) ((1 *))) p)
           (setf kk j)
           (setf p (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))))))
         (cond
          ((/= kk i)
```

```
              (setf (f2cl-lib:fref d-%data% (kk) ((1 *)) d-%offset%)
                    (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
              (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) p)
              (cond
               ((= icompq 1)
                (setf (f2cl-lib:fref iq-%data% (i) ((1 *)) iq-%offset%) kk))
               ((= icompq 2)
                (dswap n
                 (f2cl-lib:array-slice u double-float (1 i) ((1 ldu) (1 *)))
                 1
                 (f2cl-lib:array-slice u double-float (1 kk) ((1 ldu) (1 *)))
                 1)
                (dswap n
                 (f2cl-lib:array-slice vt double-float (i 1) ((1 ldvt) (1 *)))
                 ldvt
                 (f2cl-lib:array-slice vt double-float (kk 1) ((1 ldvt) (1 *)))
                 ldvt))))
             ((= icompq 1)
              (setf (f2cl-lib:fref iq-%data% (i) ((1 *)) iq-%offset%) i)))))
      (cond
       ((= icompq 1)
        (cond
         ((= iuplo 1)
          (setf (f2cl-lib:fref iq-%data% (n) ((1 *)) iq-%offset%) 1))
         (t
          (setf (f2cl-lib:fref iq-%data% (n) ((1 *)) iq-%offset%) 0)))))
      (if (and (= iuplo 2) (= icompq 2))
       (dlasr "L" "V" "B" n n
        (f2cl-lib:array-slice work double-float (1) ((1 *)))
        (f2cl-lib:array-slice work double-float (n) ((1 *))) u ldu))
  end_label
      (return
       (values nil nil nil nil nil nil nil nil nil nil nil nil nil info))))))
```

7.2 dbdsqr LAPACK

⟨*dbdsqr.input*⟩≡

```
)set break resume
)sys rm -f dbdsqr.output
)spool dbdsqr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dbdsqr.help*⟩≡

```
  ======================================================================
  dbdsqr examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DBDSQR  -  the  singular  values and, optionally, the right and/or left
         singular vectors from the singular value decomposition (SVD) of a  real
         N-by-N  (upper  or  lower) bidiagonal matrix B using the implicit zero-
         shift QR algorithm

  SYNOPSIS
         SUBROUTINE DBDSQR( UPLO, N, NCVT, NRU, NCC, D, E, VT, LDVT, U, LDU,  C,
                            LDC, WORK, INFO )

             CHARACTER      UPLO

             INTEGER        INFO, LDC, LDU, LDVT, N, NCC, NCVT, NRU

             DOUBLE         PRECISION  C( LDC, * ), D( * ), E( * ), U( LDU, * ),
                            VT( LDVT, * ), WORK( * )

  PURPOSE
         DBDSQR computes the singular values and, optionally, the  right  and/or
         left  singular vectors from the singular value decomposition (SVD) of a
         real N-by-N (upper or lower) bidiagonal matrix  B  using  the  implicit
         zero-shift QR algorithm.  The SVD of B has the form

            B = Q * S * P**T

         where  S  is the diagonal matrix of singular values, Q is an orthogonal
         matrix of left singular vectors, and P is an orthogonal matrix of right
         singular vectors.  If left singular vectors are requested, this subrou-
         tine actually returns U*Q instead of Q, and, if right singular  vectors
         are  requested,  this  subroutine  returns P**T*VT instead of P**T, for
         given real input matrices U and VT.  When U and VT are  the  orthogonal
         matrices  that  reduce  a  general  matrix  A  to bidiagonal form:  A =
         U*B*VT, as computed by DGEBRD, then

            A = (U*Q) * S * (P**T*VT)

         is the SVD of A.  Optionally, the subroutine may  also  compute  Q**T*C
```

```
       for a given real input matrix C.

       See "Computing  Small Singular Values of Bidiagonal Matrices With Guar-
       anteed High Relative Accuracy," by J. Demmel and W. Kahan, LAPACK Work-
       ing  Note  #3  (or  SIAM  J.  Sci. Statist. Comput. vol. 11, no. 5, pp.
       873-912, Sept 1990) and
       "Accurate singular values and differential qd algorithms," by  B.  Par-
       lett  and  V.  Fernando, Technical Report CPAM-554, Mathematics Depart-
       ment, University of California at Berkeley, July 1992  for  a  detailed
       description of the algorithm.


 ARGUMENTS
       UPLO    (input) CHARACTER*1
               = 'U':  B is upper bidiagonal;
               = 'L':  B is lower bidiagonal.

       N       (input) INTEGER
               The order of the matrix B.  N >= 0.

       NCVT    (input) INTEGER
               The number of columns of the matrix VT. NCVT >= 0.

       NRU     (input) INTEGER
               The number of rows of the matrix U. NRU >= 0.

       NCC     (input) INTEGER
               The number of columns of the matrix C. NCC >= 0.

       D       (input/output) DOUBLE PRECISION array, dimension (N)
               On  entry,  the n diagonal elements of the bidiagonal matrix B.
               On exit, if INFO=0, the singular  values  of  B  in  decreasing
               order.

       E       (input/output) DOUBLE PRECISION array, dimension (N-1)
               On entry, the N-1 offdiagonal elements of the bidiagonal matrix
               B.  On exit, if INFO = 0, E is destroyed; if INFO > 0, D and  E
               will contain the diagonal and superdiagonal elements of a bidi-
               agonal matrix orthogonally  equivalent  to  the  one  given  as
               input.

       VT      (input/output) DOUBLE PRECISION array, dimension (LDVT, NCVT)
               On  entry,  an N-by-NCVT matrix VT.  On exit, VT is overwritten
               by P**T * VT.  Not referenced if NCVT = 0.

       LDVT    (input) INTEGER
```

```
          The leading dimension of the array VT.  LDVT  >=  max(1,N)  if
          NCVT > 0; LDVT >= 1 if NCVT = 0.

      U   (input/output) DOUBLE PRECISION array, dimension (LDU, N)
          On entry, an NRU-by-N matrix U.  On exit, U is overwritten by U
          * Q.  Not referenced if NRU = 0.

      LDU (input) INTEGER
          The leading dimension of the array U.  LDU >= max(1,NRU).

      C   (input/output) DOUBLE PRECISION array, dimension (LDC, NCC)
          On entry, an N-by-NCC matrix C.  On exit, C is  overwritten  by
          Q**T * C.  Not referenced if NCC = 0.

      LDC (input) INTEGER
          The leading dimension of the array C.  LDC >= max(1,N) if NCC >
          0; LDC >=1 if NCC = 0.

      WORK (workspace) DOUBLE PRECISION array, dimension (2*N)
          if NCVT = NRU = NCC = 0, (max(1, 4*N)) otherwise

      INFO (output) INTEGER
          = 0:  successful exit
          < 0:  If INFO = -i, the i-th argument had an illegal value
          > 0:  the algorithm did not converge; D and E contain the  ele-
          ments  of  a bidiagonal matrix which is orthogonally similar to
          the input matrix B;  if INFO = i, i elements of E have not con-
          verged to zero.

PARAMETERS
      TOLMUL  DOUBLE PRECISION, default = max(10,min(100,EPS**(-1/8)))
          TOLMUL  controls  the convergence criterion of the QR loop.  If
          it is positive, TOLMUL*EPS is the desired relative precision in
          the  computed  singular  values.  If  it is negative, abs(TOL-
          MUL*EPS*sigma_max) is the desired absolute accuracy in the com-
          puted  singular  values  (corresponds  to  relative  accuracy
          abs(TOLMUL*EPS) in the  largest  singular  value.  abs(TOLMUL)
          should  be  between 1 and 1/EPS, and preferably between 10 (for
          fast convergence) and .1/EPS (for there to be some accuracy  in
          the  results).  Default is to lose at either one eighth or 2 of
          the available decimal digits in each  computed  singular  value
          (whichever is smaller).

      MAXITR  INTEGER, default = 6
          MAXITR  controls  the maximum number of passes of the algorithm
          through its inner loop. The algorithms stops (and so  fails  to
```

```
converge) if the number of passes through the inner loop
exceeds MAXITR*N**2.
```

⟨*LAPACK dbdsqr*⟩≡

```
(let* ((zero 0.0)
       (one 1.0)
       (negone (- 1.0))
       (hndrth 0.01)
       (ten 10.0)
       (hndrd 100.0)
       (meigth (- 0.125))
       (maxitr 6))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float) negone)
           (type (double-float 0.01 0.01) hndrth)
           (type (double-float 10.0 10.0) ten)
           (type (double-float 100.0 100.0) hndrd)
           (type (double-float) meigth)
           (type (fixnum 6 6) maxitr))
  (defun dbdsqr (uplo n ncvt nru ncc d e vt ldvt u ldu c ldc work info)
    (declare (type (simple-array double-float (*)) work c u vt e d)
             (type fixnum info ldc ldu ldvt ncc nru ncvt n)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (d double-float d-%data% d-%offset%)
         (e double-float e-%data% e-%offset%)
         (vt double-float vt-%data% vt-%offset%)
         (u double-float u-%data% u-%offset%)
         (c double-float c-%data% c-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((abse 0.0) (abss 0.0) (cosl 0.0) (cosr 0.0) (cs 0.0) (eps 0.0)
             (f 0.0) (g 0.0) (h 0.0) (mu 0.0) (oldcs 0.0) (oldsn 0.0) (r 0.0)
             (shift 0.0) (sigmn 0.0) (sigmx 0.0) (sinl 0.0) (sinr 0.0)
             (sll 0.0) (smax 0.0) (smin 0.0) (sminl 0.0) (sminlo 0.0)
             (sminoa 0.0) (sn 0.0) (thresh 0.0) (tol 0.0) (tolmul 0.0)
             (unfl 0.0) (i 0) (idir 0) (isub 0) (iter 0) (j 0) (ll 0) (lll 0)
             (m 0) (maxit 0) (nm1 0) (nm12 0) (nm13 0) (oldll 0) (oldm 0)
             (lower nil) (rotate nil))
        (declare (type (double-float) abse abss cosl cosr cs eps f g h mu oldcs
                                      oldsn r shift sigmn sigmx sinl sinr sll
                                      smax smin sminl sminlo sminoa sn thresh
                                      tol tolmul unfl)
                 (type fixnum i idir isub iter j ll lll m maxit
                                        nm1 nm12 nm13 oldll oldm)
                 (type (member t nil) lower rotate))
        (setf info 0)
        (setf lower (char-equal uplo #\L))
```

```
(cond
  ((and (not (char-equal uplo #\U)) (not lower))
   (setf info -1))
  ((< n 0)
   (setf info -2))
  ((< ncvt 0)
   (setf info -3))
  ((< nru 0)
   (setf info -4))
  ((< ncc 0)
   (setf info -5))
  ((or (and (= ncvt 0) (< ldvt 1))
       (and (> ncvt 0)
            (< ldvt
               (max (the fixnum 1)
                    (the fixnum n)))))
   (setf info -9))
  ((< ldu (max (the fixnum 1) (the fixnum nru)))
   (setf info -11))
  ((or (and (= ncc 0) (< ldc 1))
       (and (> ncc 0)
            (< ldc
               (max (the fixnum 1)
                    (the fixnum n)))))
   (setf info -13)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DBDSQR" (f2cl-lib:int-sub info))
   (go end_label)))
(if (= n 0) (go end_label))
(if (= n 1) (go label160))
(setf rotate (or (> ncvt 0) (> nru 0) (> ncc 0)))
(cond
  ((not rotate)
   (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
       (dlasq1 n d e work info)
     (declare (ignore var-0 var-1 var-2 var-3))
     (setf info var-4))
   (go end_label)))
(setf nm1 (f2cl-lib:int-sub n 1))
(setf nm12 (f2cl-lib:int-add nm1 nm1))
(setf nm13 (f2cl-lib:int-add nm12 nm1))
(setf idir 0)
(setf eps (dlamch "Epsilon"))
```

```
(setf unfl (dlamch "Safe minimum"))
(cond
  (lower
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                 ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
     (tagbody
       (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
           (dlartg (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
            (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%) cs sn r)
         (declare (ignore var-0 var-1))
         (setf cs var-2)
         (setf sn var-3)
         (setf r var-4))
       (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) r)
       (setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
               (* sn
                  (f2cl-lib:fref d-%data%
                                 ((f2cl-lib:int-add i 1))
                                 ((1 *))
                                 d-%offset%)))
       (setf (f2cl-lib:fref d-%data%
                            ((f2cl-lib:int-add i 1))
                            ((1 *))
                            d-%offset%)
               (* cs
                  (f2cl-lib:fref d-%data%
                                 ((f2cl-lib:int-add i 1))
                                 ((1 *))
                                 d-%offset%)))
       (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%) cs)
       (setf (f2cl-lib:fref work-%data%
                            ((f2cl-lib:int-add nm1 i))
                            ((1 *))
                            work-%offset%)
               sn)))
   (if (> nru 0)
       (dlasr "R" "V" "F" nru n
        (f2cl-lib:array-slice work double-float (1) ((1 *)))
        (f2cl-lib:array-slice work double-float (n) ((1 *))) u ldu))
   (if (> ncc 0)
       (dlasr "L" "V" "F" n ncc
        (f2cl-lib:array-slice work double-float (1) ((1 *)))
        (f2cl-lib:array-slice work double-float (n) ((1 *))) c ldc))))
(setf tolmul (max ten (min hndrd (expt eps meigth))))
(setf tol (* tolmul eps))
(setf smax zero)
```

```
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i n) nil)
            (tagbody
              (setf smax
                      (max smax
                           (abs
                            (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))))))
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
            (tagbody
              (setf smax
                      (max smax
                           (abs
                            (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%))))))
          (setf sminl zero)
          (cond
            ((>= tol zero)
             (tagbody
               (setf sminoa
                       (abs (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)))
               (if (= sminoa zero) (go label50))
               (setf mu sminoa)
               (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                             ((> i n) nil)
                 (tagbody
                   (setf mu
                           (*
                            (abs (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
                            (/ mu
                               (+ mu
                                  (abs
                                   (f2cl-lib:fref e-%data%
                                                  ((f2cl-lib:int-sub i 1))
                                                  ((1 *))
                                                  e-%offset%))))))
                   (setf sminoa (min sminoa mu))
                   (if (= sminoa zero) (go label50))))
        label50
               (setf sminoa
                (/ sminoa (f2cl-lib:fsqrt (coerce (realpart n) 'double-float))))
               (setf thresh (max (* tol sminoa) (* maxitr n n unfl)))))
            (t
             (setf thresh (max (* (abs tol) smax) (* maxitr n n unfl)))))
          (setf maxit (f2cl-lib:int-mul maxitr n n))
          (setf iter 0)
          (setf oldll -1)
```

```
        (setf oldm -1)
        (setf m n)
  label60
        (if (<= m 1) (go label160))
        (if (> iter maxit) (go label200))
        (if
         (and (< tol zero)
              (<= (abs (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%))
                  thresh))
         (setf (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%) zero))
        (setf smax (abs (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%)))
        (setf smin smax)
        (f2cl-lib:fdo (lll 1 (f2cl-lib:int-add lll 1))
                      ((> lll (f2cl-lib:int-add m (f2cl-lib:int-sub 1))) nil)
          (tagbody
            (setf ll (f2cl-lib:int-sub m lll))
            (setf abss (abs (f2cl-lib:fref d-%data% (ll) ((1 *)) d-%offset%)))
            (setf abse (abs (f2cl-lib:fref e-%data% (ll) ((1 *)) e-%offset%)))
            (if (and (< tol zero) (<= abss thresh))
                (setf (f2cl-lib:fref d-%data% (ll) ((1 *)) d-%offset%) zero))
            (if (<= abse thresh) (go label80))
            (setf smin (min smin abss))
            (setf smax (max smax abss abse))))
        (setf ll 0)
        (go label90)
  label80
        (setf (f2cl-lib:fref e-%data% (ll) ((1 *)) e-%offset%) zero)
        (cond
          ((= ll (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
           (setf m (f2cl-lib:int-sub m 1))
           (go label60)))
  label90
        (setf ll (f2cl-lib:int-add ll 1))
        (cond
          ((= ll (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
           (multiple-value-bind
                 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
               (dlasv2
                (f2cl-lib:fref d-%data%
                               ((f2cl-lib:int-sub m 1))
                               ((1 *))
                               d-%offset%)
                (f2cl-lib:fref e-%data%
                               ((f2cl-lib:int-sub m 1))
                               ((1 *))
                               e-%offset%)
```

```
     (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%) sigmn sigmx
     sinr cosr sinl cosl)
  (declare (ignore var-0 var-1 var-2))
  (setf sigmn var-3)
  (setf sigmx var-4)
  (setf sinr var-5)
  (setf cosr var-6)
  (setf sinl var-7)
  (setf cosl var-8))
(setf (f2cl-lib:fref d-%data%
                     ((f2cl-lib:int-sub m 1))
                     ((1 *))
                     d-%offset%)
        sigmx)
(setf (f2cl-lib:fref e-%data%
                     ((f2cl-lib:int-sub m 1))
                     ((1 *))
                     e-%offset%)
        zero)
(setf (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%) sigmn)
(if (> ncvt 0)
    (drot ncvt
     (f2cl-lib:array-slice vt
                           double-float
                           ((+ m (f2cl-lib:int-sub 1)) 1)
                           ((1 ldvt) (1 *)))
     ldvt
     (f2cl-lib:array-slice vt double-float (m 1) ((1 ldvt) (1 *)))
     ldvt cosr sinr))
(if (> nru 0)
    (drot nru
     (f2cl-lib:array-slice u
                           double-float
                           (1 (f2cl-lib:int-sub m 1))
                           ((1 ldu) (1 *)))
     1 (f2cl-lib:array-slice u double-float (1 m) ((1 ldu) (1 *))) 1
     cosl sinl))
(if (> ncc 0)
    (drot ncc
     (f2cl-lib:array-slice c
                           double-float
                           ((+ m (f2cl-lib:int-sub 1)) 1)
                           ((1 ldc) (1 *)))
     ldc (f2cl-lib:array-slice c double-float (m 1) ((1 ldc) (1 *)))
     ldc cosl sinl))
(setf m (f2cl-lib:int-sub m 2))
```

```
   (go label60)))
(cond
  ((or (> ll oldm) (< m oldll))
   (cond
     ((>= (abs (f2cl-lib:fref d (ll) ((1 *))))
          (abs (f2cl-lib:fref d (m) ((1 *)))))
      (setf idir 1))
     (t
      (setf idir 2)))))
(cond
  ((= idir 1)
   (cond
     ((or
       (<=
        (abs
         (f2cl-lib:fref e
                        ((f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
                        ((1 *))))
        (* (abs tol) (abs (f2cl-lib:fref d (m) ((1 *))))))
       (and (< tol zero)
            (<=
             (abs
              (f2cl-lib:fref e
                             ((f2cl-lib:int-add m
                                                (f2cl-lib:int-sub 1)))
                             ((1 *))))
             thresh)))
      (setf (f2cl-lib:fref e-%data%
                           ((f2cl-lib:int-sub m 1))
                           ((1 *))
                           e-%offset%)
            zero)
      (go label60)))
   (cond
     ((>= tol zero)
      (setf mu (abs (f2cl-lib:fref d-%data% (ll) ((1 *)) d-%offset%)))
      (setf sminl mu)
      (f2cl-lib:fdo (lll ll (f2cl-lib:int-add lll 1))
                    ((> lll (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
                     nil)
        (tagbody
          (cond
            ((<= (abs (f2cl-lib:fref e (lll) ((1 *)))) (* tol mu))
             (setf (f2cl-lib:fref e-%data% (lll) ((1 *)) e-%offset%)
                   zero)
             (go label60)))
```

```
                (setf sminlo sminl)
                (setf mu
                        (*
                         (abs
                          (f2cl-lib:fref d-%data%
                                         ((f2cl-lib:int-add lll 1))
                                         ((1 *))
                                         d-%offset%))
                         (/ mu
                            (+ mu
                               (abs
                                (f2cl-lib:fref e-%data%
                                               (lll)
                                               ((1 *))
                                               e-%offset%))))))
                (setf sminl (min sminl mu)))))))
          (t
           (cond
             ((or
               (<= (abs (f2cl-lib:fref e (ll) ((1 *))))
                   (* (abs tol) (abs (f2cl-lib:fref d (ll) ((1 *))))))
               (and (< tol zero)
                    (<= (abs (f2cl-lib:fref e (ll) ((1 *)))) thresh)))
              (setf (f2cl-lib:fref e-%data% (ll) ((1 *)) e-%offset%) zero)
              (go label60)))
           (cond
             ((>= tol zero)
              (setf mu (abs (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%)))
              (setf sminl mu)
              (f2cl-lib:fdo (lll (f2cl-lib:int-add m (f2cl-lib:int-sub 1))
                                 (f2cl-lib:int-add lll (f2cl-lib:int-sub 1)))
                            ((> lll ll) nil)
                (tagbody
                  (cond
                    ((<= (abs (f2cl-lib:fref e (lll) ((1 *)))) (* tol mu))
                     (setf (f2cl-lib:fref e-%data% (lll) ((1 *)) e-%offset%)
                             zero)
                     (go label60)))
                  (setf sminlo sminl)
                  (setf mu
                          (*
                           (abs
                            (f2cl-lib:fref d-%data% (lll) ((1 *)) d-%offset%))
                           (/ mu
                              (+ mu
                                 (abs
```

```
                              (f2cl-lib:fref e-%data%
                                             (lll)
                                             ((1 *))
                                             e-%offset%))))))
            (setf sminl (min sminl mu))))))))
    (setf oldll ll)
    (setf oldm m)
    (cond
      ((and (>= tol zero)
            (<= (* n tol (f2cl-lib:f2cl/ sminl smax))
                (max eps (* hndrth tol))))
       (setf shift zero))
      (t
       (cond
         ((= idir 1)
          (setf sll (abs (f2cl-lib:fref d-%data% (ll) ((1 *)) d-%offset%)))
          (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
              (dlas2
               (f2cl-lib:fref d-%data%
                              ((f2cl-lib:int-sub m 1))
                              ((1 *))
                              d-%offset%)
               (f2cl-lib:fref e-%data%
                              ((f2cl-lib:int-sub m 1))
                              ((1 *))
                              e-%offset%)
               (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%) shift r)
            (declare (ignore var-0 var-1 var-2))
            (setf shift var-3)
            (setf r var-4)))
         (t
          (setf sll (abs (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%)))
          (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
              (dlas2 (f2cl-lib:fref d-%data% (ll) ((1 *)) d-%offset%)
               (f2cl-lib:fref e-%data% (ll) ((1 *)) e-%offset%)
               (f2cl-lib:fref d-%data%
                              ((f2cl-lib:int-add ll 1))
                              ((1 *))
                              d-%offset%)
               shift r)
            (declare (ignore var-0 var-1 var-2))
            (setf shift var-3)
            (setf r var-4))))
       (cond
         ((> sll zero)
          (if (< (expt (/ shift sll) 2) eps) (setf shift zero))))))
```

```
(setf iter (f2cl-lib:int-sub (f2cl-lib:int-add iter m) ll))
(cond
  ((= shift zero)
   (cond
     ((= idir 1)
      (setf cs one)
      (setf oldcs one)
      (f2cl-lib:fdo (i ll (f2cl-lib:int-add i 1))
                    ((> i (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
                     nil)
        (tagbody
          (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
              (dlartg
               (* (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) cs)
               (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%) cs sn r)
            (declare (ignore var-0 var-1))
            (setf cs var-2)
            (setf sn var-3)
            (setf r var-4))
          (if (> i ll)
              (setf (f2cl-lib:fref e-%data%
                                   ((f2cl-lib:int-sub i 1))
                                   ((1 *))
                                   e-%offset%)
                    (* oldsn r)))
          (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
              (dlartg (* oldcs r)
               (*
                (f2cl-lib:fref d-%data%
                               ((f2cl-lib:int-add i 1))
                               ((1 *))
                               d-%offset%)
                sn)
               oldcs oldsn
               (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
            (declare (ignore var-0 var-1))
            (setf oldcs var-2)
            (setf oldsn var-3)
            (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                  var-4))
          (setf (f2cl-lib:fref work-%data%
                               ((f2cl-lib:int-add
                                 (f2cl-lib:int-sub i ll)
                                 1))
                               ((1 *))
                               work-%offset%)
```

```
                cs)
            (setf (f2cl-lib:fref work-%data%
                                 ((f2cl-lib:int-add
                                   (f2cl-lib:int-sub i ll)
                                   1
                                   nm1))
                                 ((1 *))
                                 work-%offset%)
                sn)
            (setf (f2cl-lib:fref work-%data%
                                 ((f2cl-lib:int-add
                                   (f2cl-lib:int-sub i ll)
                                   1
                                   nm12))
                                 ((1 *))
                                 work-%offset%)
                oldcs)
            (setf (f2cl-lib:fref work-%data%
                                 ((f2cl-lib:int-add
                                   (f2cl-lib:int-sub i ll)
                                   1
                                   nm13))
                                 ((1 *))
                                 work-%offset%)
                oldsn)))
        (setf h (* (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%) cs))
        (setf (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%) (* h oldcs))
        (setf (f2cl-lib:fref e-%data%
                             ((f2cl-lib:int-sub m 1))
                             ((1 *))
                             e-%offset%)
            (* h oldsn))
        (if (> ncvt 0)
            (dlasr "L" "V" "F"
             (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1) ncvt
             (f2cl-lib:array-slice work double-float (1) ((1 *)))
             (f2cl-lib:array-slice work double-float (n) ((1 *)))
             (f2cl-lib:array-slice vt
                                   double-float
                                   (ll 1)
                                   ((1 ldvt) (1 *)))
             ldvt))
        (if (> nru 0)
            (dlasr "R" "V" "F" nru
             (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1)
             (f2cl-lib:array-slice work
```

```
                                double-float
                                ((+ nm12 1))
                                ((1 *)))
          (f2cl-lib:array-slice work
                                double-float
                                ((+ nm13 1))
                                ((1 *)))
          (f2cl-lib:array-slice u double-float (1 ll) ((1 ldu) (1 *)))
          ldu))
     (if (> ncc 0)
         (dlasr "L" "V" "F"
          (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1) ncc
          (f2cl-lib:array-slice work
                                double-float
                                ((+ nm12 1))
                                ((1 *)))
          (f2cl-lib:array-slice work
                                double-float
                                ((+ nm13 1))
                                ((1 *)))
          (f2cl-lib:array-slice c double-float (ll 1) ((1 ldc) (1 *)))
          ldc))
     (if
      (<=
       (abs
        (f2cl-lib:fref e-%data%
                       ((f2cl-lib:int-sub m 1))
                       ((1 *))
                       e-%offset%))
       thresh)
      (setf (f2cl-lib:fref e-%data%
                           ((f2cl-lib:int-sub m 1))
                           ((1 *))
                           e-%offset%)
            zero)))
     (t
      (setf cs one)
      (setf oldcs one)
      (f2cl-lib:fdo (i m (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                    ((> i (f2cl-lib:int-add ll 1)) nil)
        (tagbody
          (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
              (dlartg
               (* (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) cs)
               (f2cl-lib:fref e-%data%
                              ((f2cl-lib:int-sub i 1))
```

```
                    ((1 *))
                    e-%offset%)
     cs sn r)
  (declare (ignore var-0 var-1))
  (setf cs var-2)
  (setf sn var-3)
  (setf r var-4))
(if (< i m)
    (setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
            (* oldsn r)))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlartg (* oldcs r)
     (*
      (f2cl-lib:fref d-%data%
                     ((f2cl-lib:int-sub i 1))
                     ((1 *))
                     d-%offset%)
      sn)
     oldcs oldsn
     (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
  (declare (ignore var-0 var-1))
  (setf oldcs var-2)
  (setf oldsn var-3)
  (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
          var-4))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-sub i ll))
                     ((1 *))
                     work-%offset%)
        cs)
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add
                       (f2cl-lib:int-sub i ll)
                       nm1))
                     ((1 *))
                     work-%offset%)
        (- sn))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add
                       (f2cl-lib:int-sub i ll)
                       nm12))
                     ((1 *))
                     work-%offset%)
        oldcs)
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add
```

```
                                  (f2cl-lib:int-sub i ll)
                                  nm13))
                                ((1 *))
                                work-%offset%)
                    (- oldsn))))
          (setf h (* (f2cl-lib:fref d-%data% (ll) ((1 *)) d-%offset%) cs))
          (setf (f2cl-lib:fref d-%data% (ll) ((1 *)) d-%offset%)
                  (* h oldcs))
          (setf (f2cl-lib:fref e-%data% (ll) ((1 *)) e-%offset%)
                  (* h oldsn))
          (if (> ncvt 0)
              (dlasr "L" "V" "B"
               (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1) ncvt
               (f2cl-lib:array-slice work
                                     double-float
                                     ((+ nm12 1))
                                     ((1 *)))
               (f2cl-lib:array-slice work
                                     double-float
                                     ((+ nm13 1))
                                     ((1 *)))
               (f2cl-lib:array-slice vt
                                     double-float
                                     (ll 1)
                                     ((1 ldvt) (1 *)))
               ldvt))
          (if (> nru 0)
              (dlasr "R" "V" "B" nru
               (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1)
               (f2cl-lib:array-slice work double-float (1) ((1 *)))
               (f2cl-lib:array-slice work double-float (n) ((1 *)))
               (f2cl-lib:array-slice u double-float (1 ll) ((1 ldu) (1 *)))
               ldu))
          (if (> ncc 0)
              (dlasr "L" "V" "B"
               (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1) ncc
               (f2cl-lib:array-slice work double-float (1) ((1 *)))
               (f2cl-lib:array-slice work double-float (n) ((1 *)))
               (f2cl-lib:array-slice c double-float (ll 1) ((1 ldc) (1 *)))
               ldc))
          (if
           (<= (abs (f2cl-lib:fref e-%data% (ll) ((1 *)) e-%offset%))
               thresh)
           (setf (f2cl-lib:fref e-%data% (ll) ((1 *)) e-%offset%) zero)))))
        (t
         (cond
```

```
((= idir 1)
 (setf f
        (*
         (-
          (abs (f2cl-lib:fref d-%data% (ll) ((1 *)) d-%offset%))
          shift)
         (+
          (f2cl-lib:sign one
                         (f2cl-lib:fref d-%data%
                                        (ll)
                                        ((1 *))
                                        d-%offset%))
          (/ shift
             (f2cl-lib:fref d-%data% (ll) ((1 *)) d-%offset%)))))
 (setf g (f2cl-lib:fref e-%data% (ll) ((1 *)) e-%offset%))
 (f2cl-lib:fdo (i ll (f2cl-lib:int-add i 1))
               ((> i (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
                nil)
   (tagbody
     (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
         (dlartg f g cosr sinr r)
       (declare (ignore var-0 var-1))
       (setf cosr var-2)
       (setf sinr var-3)
       (setf r var-4))
     (if (> i ll)
         (setf (f2cl-lib:fref e-%data%
                              ((f2cl-lib:int-sub i 1))
                              ((1 *))
                              e-%offset%)
               r))
     (setf f
             (+
              (* cosr
                 (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
              (* sinr
                 (f2cl-lib:fref e-%data%
                                (i)
                                ((1 *))
                                e-%offset%))))
     (setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
             (-
              (* cosr
                 (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%))
              (* sinr
                 (f2cl-lib:fref d-%data%
```

```
                              (i)
                              ((1 *))
                              d-%offset%))))
(setf g
        (* sinr
           (f2cl-lib:fref d-%data%
                         ((f2cl-lib:int-add i 1))
                         ((1 *))
                         d-%offset%)))
(setf (f2cl-lib:fref d-%data%
                     ((f2cl-lib:int-add i 1))
                     ((1 *))
                     d-%offset%)
        (* cosr
           (f2cl-lib:fref d-%data%
                         ((f2cl-lib:int-add i 1))
                         ((1 *))
                         d-%offset%)))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlartg f g cosl sinl r)
  (declare (ignore var-0 var-1))
  (setf cosl var-2)
  (setf sinl var-3)
  (setf r var-4))
(setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) r)
(setf f
        (+
         (* cosl
            (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%))
         (* sinl
            (f2cl-lib:fref d-%data%
                          ((f2cl-lib:int-add i 1))
                          ((1 *))
                          d-%offset%))))
(setf (f2cl-lib:fref d-%data%
                     ((f2cl-lib:int-add i 1))
                     ((1 *))
                     d-%offset%)
        (-
         (* cosl
            (f2cl-lib:fref d-%data%
                          ((f2cl-lib:int-add i 1))
                          ((1 *))
                          d-%offset%))
         (* sinl
            (f2cl-lib:fref e-%data%
```

```
                                (i)
                                ((1 *))
                                e-%offset%))))
                (cond
                  ((< i (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
                   (setf g
                           (* sinl
                              (f2cl-lib:fref e-%data%
                                             ((f2cl-lib:int-add i 1))
                                             ((1 *))
                                             e-%offset%)))
                   (setf (f2cl-lib:fref e-%data%
                                        ((f2cl-lib:int-add i 1))
                                        ((1 *))
                                        e-%offset%)
                           (* cosl
                              (f2cl-lib:fref e-%data%
                                             ((f2cl-lib:int-add i 1))
                                             ((1 *))
                                             e-%offset%)))))
                (setf (f2cl-lib:fref work-%data%
                                     ((f2cl-lib:int-add
                                       (f2cl-lib:int-sub i ll)
                                       1))
                                     ((1 *))
                                     work-%offset%)
                        cosr)
                (setf (f2cl-lib:fref work-%data%
                                     ((f2cl-lib:int-add
                                       (f2cl-lib:int-sub i ll)
                                       1
                                       nm1))
                                     ((1 *))
                                     work-%offset%)
                        sinr)
                (setf (f2cl-lib:fref work-%data%
                                     ((f2cl-lib:int-add
                                       (f2cl-lib:int-sub i ll)
                                       1
                                       nm12))
                                     ((1 *))
                                     work-%offset%)
                        cosl)
                (setf (f2cl-lib:fref work-%data%
                                     ((f2cl-lib:int-add
                                       (f2cl-lib:int-sub i ll)
```

```
                                1
                                nm13))
                              ((1 *))
                              work-%offset%)
                  sinl)))
      (setf (f2cl-lib:fref e-%data%
                           ((f2cl-lib:int-sub m 1))
                           ((1 *))
                           e-%offset%)
            f)
      (if (> ncvt 0)
          (dlasr "L" "V" "F"
           (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1) ncvt
           (f2cl-lib:array-slice work double-float (1) ((1 *)))
           (f2cl-lib:array-slice work double-float (n) ((1 *)))
           (f2cl-lib:array-slice vt
                                 double-float
                                 (ll 1)
                                 ((1 ldvt) (1 *)))
           ldvt))
      (if (> nru 0)
          (dlasr "R" "V" "F" nru
           (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1)
           (f2cl-lib:array-slice work
                                 double-float
                                 ((+ nm12 1))
                                 ((1 *)))
           (f2cl-lib:array-slice work
                                 double-float
                                 ((+ nm13 1))
                                 ((1 *)))
           (f2cl-lib:array-slice u double-float (1 ll) ((1 ldu) (1 *)))
           ldu))
      (if (> ncc 0)
          (dlasr "L" "V" "F"
           (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1) ncc
           (f2cl-lib:array-slice work
                                 double-float
                                 ((+ nm12 1))
                                 ((1 *)))
           (f2cl-lib:array-slice work
                                 double-float
                                 ((+ nm13 1))
                                 ((1 *)))
           (f2cl-lib:array-slice c double-float (ll 1) ((1 ldc) (1 *)))
           ldc))
```

```
                (if
                 (<=
                  (abs
                   (f2cl-lib:fref e-%data%
                                  ((f2cl-lib:int-sub m 1))
                                  ((1 *))
                                  e-%offset%))
                  thresh)
                 (setf (f2cl-lib:fref e-%data%
                                      ((f2cl-lib:int-sub m 1))
                                      ((1 *))
                                      e-%offset%)
                       zero)))
               (t
                (setf f
                      (*
                       (- (abs (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%))
                          shift)
                       (+
                        (f2cl-lib:sign one
                                       (f2cl-lib:fref d-%data%
                                                      (m)
                                                      ((1 *))
                                                      d-%offset%))
                        (/ shift
                           (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%)))))
                (setf g
                      (f2cl-lib:fref e-%data%
                                     ((f2cl-lib:int-sub m 1))
                                     ((1 *))
                                     e-%offset%))
                (f2cl-lib:fdo (i m (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                              ((> i (f2cl-lib:int-add ll 1)) nil)
                  (tagbody
                    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                        (dlartg f g cosr sinr r)
                      (declare (ignore var-0 var-1))
                      (setf cosr var-2)
                      (setf sinr var-3)
                      (setf r var-4))
                    (if (< i m)
                        (setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%) r))
                    (setf f
                          (+
                           (* cosr
                              (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
```

```
          (* sinr
             (f2cl-lib:fref e-%data%
                            ((f2cl-lib:int-sub i 1))
                            ((1 *))
                            e-%offset%))))
(setf (f2cl-lib:fref e-%data%
                     ((f2cl-lib:int-sub i 1))
                     ((1 *))
                     e-%offset%)
        (-
         (* cosr
            (f2cl-lib:fref e-%data%
                           ((f2cl-lib:int-sub i 1))
                           ((1 *))
                           e-%offset%))
         (* sinr
            (f2cl-lib:fref d-%data%
                           (i)
                           ((1 *))
                           d-%offset%))))
(setf g
        (* sinr
           (f2cl-lib:fref d-%data%
                          ((f2cl-lib:int-sub i 1))
                          ((1 *))
                          d-%offset%)))
(setf (f2cl-lib:fref d-%data%
                     ((f2cl-lib:int-sub i 1))
                     ((1 *))
                     d-%offset%)
        (* cosr
           (f2cl-lib:fref d-%data%
                          ((f2cl-lib:int-sub i 1))
                          ((1 *))
                          d-%offset%)))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlartg f g cosl sinl r)
  (declare (ignore var-0 var-1))
  (setf cosl var-2)
  (setf sinl var-3)
  (setf r var-4))
(setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) r)
(setf f
        (+
         (* cosl
            (f2cl-lib:fref e-%data%
```

```
                                ((f2cl-lib:int-sub i 1))
                                ((1 *))
                                e-%offset%))
                        (* sinl
                           (f2cl-lib:fref d-%data%
                                ((f2cl-lib:int-sub i 1))
                                ((1 *))
                                d-%offset%))))
                (setf (f2cl-lib:fref d-%data%
                                     ((f2cl-lib:int-sub i 1))
                                     ((1 *))
                                     d-%offset%)
                      (-
                       (* cosl
                          (f2cl-lib:fref d-%data%
                                ((f2cl-lib:int-sub i 1))
                                ((1 *))
                                d-%offset%))
                       (* sinl
                          (f2cl-lib:fref e-%data%
                                ((f2cl-lib:int-sub i 1))
                                ((1 *))
                                e-%offset%))))
                (cond
                  ((> i (f2cl-lib:int-add ll 1))
                   (setf g
                           (* sinl
                              (f2cl-lib:fref e-%data%
                                   ((f2cl-lib:int-sub i 2))
                                   ((1 *))
                                   e-%offset%)))
                   (setf (f2cl-lib:fref e-%data%
                                        ((f2cl-lib:int-sub i 2))
                                        ((1 *))
                                        e-%offset%)
                           (* cosl
                              (f2cl-lib:fref e-%data%
                                   ((f2cl-lib:int-sub i 2))
                                   ((1 *))
                                   e-%offset%)))))
                (setf (f2cl-lib:fref work-%data%
                                     ((f2cl-lib:int-sub i ll))
                                     ((1 *))
                                     work-%offset%)
                      cosr)
                (setf (f2cl-lib:fref work-%data%
```

```
                                ((f2cl-lib:int-add
                                  (f2cl-lib:int-sub i ll)
                                  nm1))
                                ((1 *))
                                work-%offset%)
                        (- sinr))
                    (setf (f2cl-lib:fref work-%data%
                                ((f2cl-lib:int-add
                                  (f2cl-lib:int-sub i ll)
                                  nm12))
                                ((1 *))
                                work-%offset%)
                        cosl)
                    (setf (f2cl-lib:fref work-%data%
                                ((f2cl-lib:int-add
                                  (f2cl-lib:int-sub i ll)
                                  nm13))
                                ((1 *))
                                work-%offset%)
                        (- sinl))))
                (setf (f2cl-lib:fref e-%data% (ll) ((1 *)) e-%offset%) f)
                (if
                 (<= (abs (f2cl-lib:fref e-%data% (ll) ((1 *)) e-%offset%))
                     thresh)
                 (setf (f2cl-lib:fref e-%data% (ll) ((1 *)) e-%offset%) zero))
                (if (> ncvt 0)
                    (dlasr "L" "V" "B"
                     (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1) ncvt
                     (f2cl-lib:array-slice work
                                           double-float
                                           ((+ nm12 1))
                                           ((1 *)))
                     (f2cl-lib:array-slice work
                                           double-float
                                           ((+ nm13 1))
                                           ((1 *)))
                     (f2cl-lib:array-slice vt
                                           double-float
                                           (ll 1)
                                           ((1 ldvt) (1 *)))
                     ldvt))
                (if (> nru 0)
                    (dlasr "R" "V" "B" nru
                     (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1)
                     (f2cl-lib:array-slice work double-float (1) ((1 *)))
                     (f2cl-lib:array-slice work double-float (n) ((1 *)))
```

```
                    (f2cl-lib:array-slice u double-float (1 ll) ((1 ldu) (1 *)))
                    ldu))
                (if (> ncc 0)
                    (dlasr "L" "V" "B"
                     (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1) ncc
                     (f2cl-lib:array-slice work double-float (1) ((1 *)))
                     (f2cl-lib:array-slice work double-float (n) ((1 *)))
                     (f2cl-lib:array-slice c double-float (ll 1) ((1 ldc) (1 *)))
                     ldc))))))
          (go label60)
  label160
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i n) nil)
            (tagbody
              (cond
                ((< (f2cl-lib:fref d (i) ((1 *))) zero)
                 (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                         (- (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)))
                 (if (> ncvt 0)
                     (dscal ncvt negone
                      (f2cl-lib:array-slice vt
                                            double-float
                                            (i 1)
                                            ((1 ldvt) (1 *)))
                      ldvt))))))
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
            (tagbody
              (setf isub 1)
              (setf smin (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))
              (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                            ((> j (f2cl-lib:int-add n 1 (f2cl-lib:int-sub i)))
                             nil)
                (tagbody
                  (cond
                    ((<= (f2cl-lib:fref d (j) ((1 *))) smin)
                     (setf isub j)
                     (setf smin
                             (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))))))
              (cond
                ((/= isub (f2cl-lib:int-add n 1 (f2cl-lib:int-sub i)))
                 (setf (f2cl-lib:fref d-%data% (isub) ((1 *)) d-%offset%)
                         (f2cl-lib:fref d-%data%
                                        ((f2cl-lib:int-sub (f2cl-lib:int-add n 1)
                                                           i))
                                        ((1 *))
```

```
                          d-%offset%))
(setf (f2cl-lib:fref d-%data%
                     ((f2cl-lib:int-sub (f2cl-lib:int-add n 1)
                                        i))
                     ((1 *))
                     d-%offset%)
        smin)
(if (> ncvt 0)
    (dswap ncvt
     (f2cl-lib:array-slice vt
                           double-float
                           (isub 1)
                           ((1 ldvt) (1 *)))
     ldvt
     (f2cl-lib:array-slice vt
                           double-float
                           ((+ n 1 (f2cl-lib:int-sub i)) 1)
                           ((1 ldvt) (1 *)))
     ldvt))
(if (> nru 0)
    (dswap nru
     (f2cl-lib:array-slice u
                           double-float
                           (1 isub)
                           ((1 ldu) (1 *)))
     1
     (f2cl-lib:array-slice u
                           double-float
                           (1
                            (f2cl-lib:int-sub
                             (f2cl-lib:int-add n 1)
                             i))
                           ((1 ldu) (1 *)))
     1))
(if (> ncc 0)
    (dswap ncc
     (f2cl-lib:array-slice c
                           double-float
                           (isub 1)
                           ((1 ldc) (1 *)))
     ldc
     (f2cl-lib:array-slice c
                           double-float
                           ((+ n 1 (f2cl-lib:int-sub i)) 1)
                           ((1 ldc) (1 *)))
     ldc))))))
```

```
        (go end_label)
   label200
        (setf info 0)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
          (tagbody
            (if (/= (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%) zero)
                (setf info (f2cl-lib:int-add info 1)))))
   end_label
        (return
         (values nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 info))))))
```

7.3 ddisna LAPACK

⟨*ddisna.input*⟩≡

```
  )set break resume
  )sys rm -f ddisna.output
  )spool ddisna.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*ddisna.help*⟩≡

```
==========================================================================
ddisna examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

NAME
       DDISNA  -  the  reciprocal  condition numbers for the eigenvectors of a
       real symmetric or complex Hermitian matrix or for  the  left  or  right
       singular vectors of a general m-by-n matrix

SYNOPSIS
       SUBROUTINE DDISNA( JOB, M, N, D, SEP, INFO )

           CHARACTER      JOB

           INTEGER        INFO, M, N

           DOUBLE         PRECISION D( * ), SEP( * )

PURPOSE
       DDISNA  computes  the reciprocal condition numbers for the eigenvectors
       of a real symmetric or complex Hermitian matrix  or  for  the  left  or
       right  singular vectors of a general m-by-n matrix. The reciprocal con-
       dition number is the 'gap' between the corresponding eigenvalue or sin-
       gular value and the nearest other one.

       The  bound on the error, measured by angle in radians, in the I-th com-
       puted vector is given by

              DLAMCH( 'E' ) * ( ANORM / SEP( I ) )

       where ANORM = 2-norm(A) = max( abs( D(j) ) ).  SEP(I) is not allowed to
       be  smaller  than DLAMCH( 'E' )*ANORM in order to limit the size of the
       error bound.

       DDISNA may also be used to compute error bounds for eigenvectors of the
       generalized symmetric definite eigenproblem.


ARGUMENTS
       JOB     (input) CHARACTER*1
               Specifies  for  which  problem the reciprocal condition numbers
```

```
          should be computed:
          = 'E':  the eigenvectors of a symmetric/Hermitian matrix;
          = 'L':  the left singular vectors of a general matrix;
          = 'R':  the right singular vectors of a general matrix.

M         (input) INTEGER
          The number of rows of the matrix. M >= 0.

N         (input) INTEGER
          If JOB = 'L' or 'R', the number of columns of  the  matrix,  in
          which case N >= 0. Ignored if JOB = 'E'.

D         (input) DOUBLE PRECISION array, dimension (M) if JOB = 'E'
          dimension  (min(M,N))  if  JOB = 'L' or 'R' The eigenvalues (if
          JOB = 'E') or singular values (if JOB =  'L'  or  'R')  of  the
          matrix,  in  either increasing or decreasing order. If singular
          values, they must be non-negative.

SEP       (output) DOUBLE PRECISION array, dimension (M) if JOB = 'E'
          dimension (min(M,N)) if JOB = 'L' or 'R' The reciprocal  condi-
          tion numbers of the vectors.

INFO      (output) INTEGER
          = 0:  successful exit.
          < 0:  if INFO = -i, the i-th argument had an illegal value.
```

⟨*LAPACK ddisna*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun ddisna (job m n d sep info)
    (declare (type (simple-array double-float (*)) sep d)
             (type fixnum info n m)
             (type character job))
    (f2cl-lib:with-multi-array-data
        ((job character job-%data% job-%offset%)
         (d double-float d-%data% d-%offset%)
         (sep double-float sep-%data% sep-%offset%))
      (prog ((anorm 0.0) (eps 0.0) (newgap 0.0) (oldgap 0.0) (safmin 0.0)
             (thresh 0.0) (i 0) (k 0) (decr nil) (eigen nil) (incr nil)
             (left nil) (right nil) (sing nil))
        (declare (type (double-float) anorm eps newgap oldgap safmin thresh)
                 (type fixnum i k)
                 (type (member t nil) decr eigen incr left right sing))
        (setf info 0)
        (setf eigen (char-equal job #\E))
        (setf left (char-equal job #\L))
        (setf right (char-equal job #\R))
        (setf sing (or left right))
        (cond
          (eigen
           (setf k m))
          (sing
           (setf k (min (the fixnum m) (the fixnum n)))))
        (cond
          ((and (not eigen) (not sing))
           (setf info -1))
          ((< m 0)
           (setf info -2))
          ((< k 0)
           (setf info -3))
          (t
           (setf incr t)
           (setf decr t)
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                         ((> i (f2cl-lib:int-add k (f2cl-lib:int-sub 1))) nil)
             (tagbody
               (if incr
                   (setf incr
                           (and incr
                                (<=
                                 (f2cl-lib:fref d-%data%
                                                (i)
```

```
                                        ((1 *))
                                        d-%offset%)
                          (f2cl-lib:fref d-%data%
                                        ((f2cl-lib:int-add i 1))
                                        ((1 *))
                                        d-%offset%)))))
          (if decr
              (setf decr
                      (and decr
                           (>=
                            (f2cl-lib:fref d-%data%
                                          (i)
                                          ((1 *))
                                          d-%offset%)
                            (f2cl-lib:fref d-%data%
                                          ((f2cl-lib:int-add i 1))
                                          ((1 *))
                                          d-%offset%)))))))
     (cond
       ((and sing (> k 0))
        (if incr
            (setf incr
                    (and incr
                         (<= zero
                             (f2cl-lib:fref d-%data%
                                           (1)
                                           ((1 *))
                                           d-%offset%)))))
        (if decr
            (setf decr
                    (and decr
                         (>=
                          (f2cl-lib:fref d-%data% (k) ((1 *)) d-%offset%)
                          zero))))))
     (if (not (or incr decr)) (setf info -4))))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DDISNA" (f2cl-lib:int-sub info))
   (go end_label)))
(if (= k 0) (go end_label))
(cond
  ((= k 1)
   (setf (f2cl-lib:fref sep-%data% (1) ((1 *)) sep-%offset%)
           (dlamch "O")))
```

```
  (t
   (setf oldgap
           (abs
            (- (f2cl-lib:fref d-%data% (2) ((1 *)) d-%offset%)
               (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))))
   (setf (f2cl-lib:fref sep-%data% (1) ((1 *)) sep-%offset%) oldgap)
   (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                 ((> i (f2cl-lib:int-add k (f2cl-lib:int-sub 1))) nil)
     (tagbody
       (setf newgap
               (abs
                (-
                 (f2cl-lib:fref d-%data%
                                ((f2cl-lib:int-add i 1))
                                ((1 *))
                                d-%offset%)
                 (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))))
       (setf (f2cl-lib:fref sep-%data% (i) ((1 *)) sep-%offset%)
               (min oldgap newgap))
       (setf oldgap newgap)))
   (setf (f2cl-lib:fref sep-%data% (k) ((1 *)) sep-%offset%) oldgap)))
(cond
  (sing
   (cond
     ((or (and left (> m n)) (and right (< m n)))
      (if incr
          (setf (f2cl-lib:fref sep-%data% (1) ((1 *)) sep-%offset%)
                  (min
                   (f2cl-lib:fref sep-%data% (1) ((1 *)) sep-%offset%)
                   (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))))
      (if decr
          (setf (f2cl-lib:fref sep-%data% (k) ((1 *)) sep-%offset%)
                  (min
                   (f2cl-lib:fref sep-%data% (k) ((1 *)) sep-%offset%)
                   (f2cl-lib:fref d-%data%
                                  (k)
                                  ((1 *))
                                  d-%offset%))))))))
(setf eps (dlamch "E"))
(setf safmin (dlamch "S"))
(setf anorm
        (max (abs (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))
             (abs (f2cl-lib:fref d-%data% (k) ((1 *)) d-%offset%))))
(cond
  ((= anorm zero)
   (setf thresh eps))
```

```
            (t
             (setf thresh (max (* eps anorm) safmin))))
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i k) nil)
            (tagbody
              (setf (f2cl-lib:fref sep-%data% (i) ((1 *)) sep-%offset%)
                      (max (f2cl-lib:fref sep-%data% (i) ((1 *)) sep-%offset%)
                           thresh))))
   end_label
          (return (values nil nil nil nil nil info))))))
```

7.4 dgebak LAPACK

⟨*dgebak.input*⟩≡

```
  )set break resume
  )sys rm -f dgebak.output
  )spool dgebak.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dgebak.help*⟩≡

```
======================================================================
dgebak examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DGEBAK  -  the  right  or left eigenvectors of a real general matrix by
       backward transformation on the computed eigenvectors  of  the  balanced
       matrix output by DGEBAL

SYNOPSIS
       SUBROUTINE DGEBAK( JOB, SIDE, N, ILO, IHI, SCALE, M, V, LDV, INFO )

           CHARACTER      JOB, SIDE

           INTEGER        IHI, ILO, INFO, LDV, M, N

           DOUBLE         PRECISION SCALE( * ), V( LDV, * )

PURPOSE
       DGEBAK forms the right or left eigenvectors of a real general matrix by
       backward transformation on the computed eigenvectors  of  the  balanced
       matrix output by DGEBAL.


ARGUMENTS
       JOB     (input) CHARACTER*1
               Specifies  the type of backward transformation required: = 'N',
               do nothing, return immediately; = 'P', do backward  transforma-
               tion  for  permutation  only; = 'S', do backward transformation
               for scaling only; = 'B', do backward transformations  for  both
               permutation  and scaling.  JOB must be the same as the argument
               JOB supplied to DGEBAL.

       SIDE    (input) CHARACTER*1
               = 'R':  V contains right eigenvectors;
               = 'L':  V contains left eigenvectors.

       N       (input) INTEGER
               The number of rows of the matrix V.  N >= 0.

       ILO     (input) INTEGER
```

```
          IHI     (input) INTEGER The integers ILO and IHI determined  by
          DGEBAL.   1  <=  ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if
          N=0.

     SCALE  (input) DOUBLE PRECISION array, dimension (N)
          Details of the permutation and scaling factors, as returned  by
          DGEBAL.

     M       (input) INTEGER
          The number of columns of the matrix V.  M >= 0.

     V       (input/output) DOUBLE PRECISION array, dimension (LDV,M)
          On entry, the matrix of right or left eigenvectors to be trans-
          formed, as returned by DHSEIN or DTREVC.  On exit, V  is  over-
          written by the transformed eigenvectors.

     LDV     (input) INTEGER
          The leading dimension of the array V. LDV >= max(1,N).

     INFO    (output) INTEGER
          = 0:  successful exit
          < 0:  if INFO = -i, the i-th argument had an illegal value.
```

⟨*LAPACK dgebak*⟩≡

```
(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dgebak (job side n ilo ihi scale m v ldv info)
    (declare (type (simple-array double-float (*)) v scale)
             (type fixnum info ldv m ihi ilo n)
             (type character side job))
    (f2cl-lib:with-multi-array-data
        ((job character job-%data% job-%offset%)
         (side character side-%data% side-%offset%)
         (scale double-float scale-%data% scale-%offset%)
         (v double-float v-%data% v-%offset%))
      (prog ((s 0.0) (i 0) (ii 0) (k 0) (leftv nil) (rightv nil))
        (declare (type (double-float) s)
                 (type fixnum i ii k)
                 (type (member t nil) leftv rightv))
        (setf rightv (char-equal side #\R))
        (setf leftv (char-equal side #\L))
        (setf info 0)
        (cond
          ((and (not (char-equal job #\N))
                (not (char-equal job #\P))
                (not (char-equal job #\S))
                (not (char-equal job #\B)))
           (setf info -1))
          ((and (not rightv) (not leftv))
           (setf info -2))
          ((< n 0)
           (setf info -3))
          ((or (< ilo 1)
               (> ilo
                  (max (the fixnum 1) (the fixnum n))))
           (setf info -4))
          ((or
            (< ihi (min (the fixnum ilo) (the fixnum n)))
            (> ihi n))
           (setf info -5))
          ((< m 0)
           (setf info -7))
          ((< ldv (max (the fixnum 1) (the fixnum n)))
           (setf info -9)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DGEBAK" (f2cl-lib:int-sub info))
```

```
           (go end_label)))
         (if (= n 0) (go end_label))
         (if (= m 0) (go end_label))
         (if (char-equal job #\N) (go end_label))
         (if (= ilo ihi) (go label30))
         (cond
           ((or (char-equal job #\S) (char-equal job #\B))
            (cond
              (rightv
               (f2cl-lib:fdo (i ilo (f2cl-lib:int-add i 1))
                             ((> i ihi) nil)
                 (tagbody
                   (setf s
                           (f2cl-lib:fref scale-%data%
                                         (i)
                                         ((1 *))
                                         scale-%offset%))
                   (dscal m s
                    (f2cl-lib:array-slice v double-float (i 1) ((1 ldv) (1 *)))
                    ldv)))))
              (cond
                (leftv
                 (f2cl-lib:fdo (i ilo (f2cl-lib:int-add i 1))
                               ((> i ihi) nil)
                   (tagbody
                     (setf s
                             (/ one
                                (f2cl-lib:fref scale-%data%
                                               (i)
                                               ((1 *))
                                               scale-%offset%)))
                     (dscal m s
                      (f2cl-lib:array-slice v double-float (i 1) ((1 ldv) (1 *)))
                      ldv)))))))
  label30
         (cond
           ((or (char-equal job #\P) (char-equal job #\B))
            (cond
              (rightv
               (f2cl-lib:fdo (ii 1 (f2cl-lib:int-add ii 1))
                             ((> ii n) nil)
                 (tagbody
                   (setf i ii)
                   (if (and (>= i ilo) (<= i ihi)) (go label40))
                   (if (< i ilo) (setf i (f2cl-lib:int-sub ilo ii)))
                   (setf k
```

```
                          (f2cl-lib:int
                           (f2cl-lib:fref scale-%data%
                                          (i)
                                          ((1 *))
                                          scale-%offset%)))
                    (if (= k i) (go label40))
                    (dswap m
                     (f2cl-lib:array-slice v double-float (i 1) ((1 ldv) (1 *)))
                     ldv
                     (f2cl-lib:array-slice v double-float (k 1) ((1 ldv) (1 *)))
                     ldv)
  label40))))
            (cond
              (leftv
               (f2cl-lib:fdo (ii 1 (f2cl-lib:int-add ii 1))
                             ((> ii n) nil)
                 (tagbody
                   (setf i ii)
                   (if (and (>= i ilo) (<= i ihi)) (go label50))
                   (if (< i ilo) (setf i (f2cl-lib:int-sub ilo ii)))
                   (setf k
                           (f2cl-lib:int
                            (f2cl-lib:fref scale-%data%
                                           (i)
                                           ((1 *))
                                           scale-%offset%)))
                   (if (= k i) (go label50))
                   (dswap m
                    (f2cl-lib:array-slice v double-float (i 1) ((1 ldv) (1 *)))
                    ldv
                    (f2cl-lib:array-slice v double-float (k 1) ((1 ldv) (1 *)))
                    ldv)
  label50))))))
  end_label
         (return (values nil nil nil nil nil nil nil nil nil info))))))
```

7.5 dgebal LAPACK

⟨*dgebal.input*⟩≡

```
)set break resume
)sys rm -f dgebal.output
)spool dgebal.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgebal.help*⟩≡

```
======================================================================
dgebal examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DGEBAL - a general real matrix A

SYNOPSIS
       SUBROUTINE DGEBAL( JOB, N, A, LDA, ILO, IHI, SCALE, INFO )

           CHARACTER      JOB

           INTEGER        IHI, ILO, INFO, LDA, N

           DOUBLE         PRECISION A( LDA, * ), SCALE( * )

PURPOSE
       DGEBAL balances a general real matrix A.  This involves, first, permut-
       ing A by a similarity transformation  to  isolate  eigenvalues  in  the
       first 1 to ILO-1 and last IHI+1 to N elements on the diagonal; and sec-
       ond, applying a diagonal similarity transformation to rows and  columns
       ILO  to  IHI to make the rows and columns as close in norm as possible.
       Both steps are optional.

       Balancing may reduce the 1-norm of the matrix, and improve the accuracy
       of the computed eigenvalues and/or eigenvectors.


ARGUMENTS
       JOB     (input) CHARACTER*1
               Specifies the operations to be performed on A:
               =  'N':  none:  simply set ILO = 1, IHI = N, SCALE(I) = 1.0 for
               i = 1,...,N; = 'P':  permute only;
               = 'S':  scale only;
               = 'B':  both permute and scale.

       N       (input) INTEGER
               The order of the matrix A.  N >= 0.

       A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
               On entry, the input matrix A.  On exit,  A  is  overwritten  by
```

the balanced matrix. If JOB = 'N', A is not referenced. See Further Details. LDA (input) INTEGER The leading dimension of the array A. LDA >= max(1,N).

ILO (output) INTEGER
IHI (output) INTEGER ILO and IHI are set to integers such that on exit A(i,j) = 0 if i > j and j = 1,...,ILO-1 or I = IHI+1,...,N. If JOB = 'N' or 'S', ILO = 1 and IHI = N.

SCALE (output) DOUBLE PRECISION array, dimension (N)
Details of the permutations and scaling factors applied to A. If P(j) is the index of the row and column interchanged with row and column j and D(j) is the scaling factor applied to row and column j, then SCALE(j) = P(j) for j = 1,...,ILO-1 = D(j) for j = ILO,...,IHI = P(j) for j = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

INFO (output) INTEGER
= 0: successful exit.
< 0: if INFO = -i, the i-th argument had an illegal value.

FURTHER DETAILS

The permutations consist of row and column interchanges which put the matrix in the form

$$P\,A\,P = \begin{pmatrix} T1 & X & Y \\ 0 & B & Z \\ 0 & 0 & T2 \end{pmatrix}$$

where T1 and T2 are upper triangular matrices whose eigenvalues lie along the diagonal. The column indices ILO and IHI mark the starting and ending columns of the submatrix B. Balancing consists of applying a diagonal similarity transformation inv(D) * B * D to make the 1-norms of each row of B and its corresponding column nearly equal. The output matrix is

$$\begin{pmatrix} T1 & X*D & Y \\ 0 & inv(D)*B*D & inv(D)*Z \\ 0 & 0 & T2 \end{pmatrix}.$$

Information about the permutations P and the diagonal matrix D is returned in the vector SCALE.

This subroutine is based on the EISPACK routine BALANC.

⟨*LAPACK dgebal*⟩≡

```
(let* ((zero 0.0) (one 1.0) (sclfac 8.0) (factor 0.95))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 8.0 8.0) sclfac)
           (type (double-float 0.95 0.95) factor))
  (defun dgebal (job n a lda ilo ihi scale info)
    (declare (type (simple-array double-float (*)) scale a)
             (type fixnum info ihi ilo lda n)
             (type character job))
    (f2cl-lib:with-multi-array-data
        ((job character job-%data% job-%offset%)
         (a double-float a-%data% a-%offset%)
         (scale double-float scale-%data% scale-%offset%))
      (prog ((c 0.0) (ca 0.0) (f 0.0) (g 0.0) (r 0.0) (ra 0.0) (s 0.0)
             (sfmax1 0.0) (sfmax2 0.0) (sfmin1 0.0) (sfmin2 0.0) (i 0) (ica 0)
             (iexc 0) (ira 0) (j 0) (k 0) (l 0) (m 0) (noconv nil))
        (declare (type (double-float) c ca f g r ra s sfmax1 sfmax2 sfmin1
                                      sfmin2)
                 (type fixnum i ica iexc ira j k l m)
                 (type (member t nil) noconv))
        (setf info 0)
        (cond
          ((and (not (char-equal job #\N))
                (not (char-equal job #\P))
                (not (char-equal job #\S))
                (not (char-equal job #\B)))
           (setf info -1))
          ((< n 0)
           (setf info -2))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info -4)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DGEBAL" (f2cl-lib:int-sub info))
           (go end_label)))
        (setf k 1)
        (setf l n)
        (if (= n 0) (go label210))
        (cond
          ((char-equal job #\N)
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                         ((> i n) nil)
             (tagbody
```

```
               (setf (f2cl-lib:fref scale-%data% (i) ((1 *)) scale-%offset%)
                     one)))
          (go label210)))
       (if (char-equal job #\S) (go label120))
       (go label50)
label20
       (setf (f2cl-lib:fref scale-%data% (m) ((1 *)) scale-%offset%)
             (coerce (the fixnum j) 'double-float))
       (if (= j m) (go label30))
       (dswap l (f2cl-lib:array-slice a double-float (1 j) ((1 lda) (1 *))) 1
        (f2cl-lib:array-slice a double-float (1 m) ((1 lda) (1 *))) 1)
       (dswap (f2cl-lib:int-add (f2cl-lib:int-sub n k) 1)
        (f2cl-lib:array-slice a double-float (j k) ((1 lda) (1 *))) lda
        (f2cl-lib:array-slice a double-float (m k) ((1 lda) (1 *))) lda)
label30
       (f2cl-lib:computed-goto (label40 label80) iexc)
label40
       (if (= l 1) (go label210))
       (setf l (f2cl-lib:int-sub l 1))
label50
       (f2cl-lib:fdo (j l (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                     ((> j 1) nil)
         (tagbody
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                         ((> i l) nil)
             (tagbody
               (if (= i j) (go label60))
               (if
                (/= (f2cl-lib:fref a-%data% (j i) ((1 lda) (1 *)) a-%offset%)
                    zero)
                (go label70))
label60))
           (setf m l)
           (setf iexc 1)
           (go label20)
label70))
       (go label90)
label80
       (setf k (f2cl-lib:int-add k 1))
label90
       (f2cl-lib:fdo (j k (f2cl-lib:int-add j 1))
                     ((> j l) nil)
         (tagbody
           (f2cl-lib:fdo (i k (f2cl-lib:int-add i 1))
                         ((> i l) nil)
             (tagbody
```

```
                (if (= i j) (go label100))
                (if
                 (/= (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *)) a-%offset%)
                     zero)
                 (go label110))
label100))
            (setf m k)
            (setf iexc 2)
            (go label20)
label110))
label120
       (f2cl-lib:fdo (i k (f2cl-lib:int-add i 1))
                     ((> i l) nil)
         (tagbody
          (setf (f2cl-lib:fref scale-%data% (i) ((1 *)) scale-%offset%) one)))
       (if (char-equal job #\P) (go label210))
       (setf sfmin1 (/ (dlamch "S") (dlamch "P")))
       (setf sfmax1 (/ one sfmin1))
       (setf sfmin2 (* sfmin1 sclfac))
       (setf sfmax2 (/ one sfmin2))
label140
       (setf noconv nil)
       (f2cl-lib:fdo (i k (f2cl-lib:int-add i 1))
                     ((> i l) nil)
         (tagbody
           (setf c zero)
           (setf r zero)
           (f2cl-lib:fdo (j k (f2cl-lib:int-add j 1))
                         ((> j l) nil)
             (tagbody
               (if (= j i) (go label150))
               (setf c
                       (+ c
                          (abs
                           (f2cl-lib:fref a-%data%
                                          (j i)
                                          ((1 lda) (1 *))
                                          a-%offset%))))
               (setf r
                       (+ r
                          (abs
                           (f2cl-lib:fref a-%data%
                                          (i j)
                                          ((1 lda) (1 *))
                                          a-%offset%))))
label150))
```

```
            (setf ica
                  (idamax l
                   (f2cl-lib:array-slice a
                                         double-float
                                         (1 i)
                                         ((1 lda) (1 *)))
                   1))
            (setf ca
                  (abs
                   (f2cl-lib:fref a-%data%
                                  (ica i)
                                  ((1 lda) (1 *))
                                  a-%offset%)))
            (setf ira
                  (idamax (f2cl-lib:int-add (f2cl-lib:int-sub n k) 1)
                   (f2cl-lib:array-slice a
                                         double-float
                                         (i k)
                                         ((1 lda) (1 *)))
                   lda))
            (setf ra
                  (abs
                   (f2cl-lib:fref a-%data%
                                  (i
                                   (f2cl-lib:int-sub (f2cl-lib:int-add ira k)
                                                     1))
                                  ((1 lda) (1 *))
                                  a-%offset%)))
            (if (or (= c zero) (= r zero)) (go label200))
            (setf g (/ r sclfac))
            (setf f one)
            (setf s (+ c r))
label160
            (if (or (>= c g) (>= (max f c ca) sfmax2) (<= (min r g ra) sfmin2))
                (go label170))
            (setf f (* f sclfac))
            (setf c (* c sclfac))
            (setf ca (* ca sclfac))
            (setf r (/ r sclfac))
            (setf g (/ g sclfac))
            (setf ra (/ ra sclfac))
            (go label160)
label170
            (setf g (/ c sclfac))
label180
            (if (or (< g r) (>= (max r ra) sfmax2) (<= (min f c g ca) sfmin2))
```

```
                (go label190))
              (setf f (/ f sclfac))
              (setf c (/ c sclfac))
              (setf g (/ g sclfac))
              (setf ca (/ ca sclfac))
              (setf r (* r sclfac))
              (setf ra (* ra sclfac))
              (go label180)
label190
              (if (>= (+ c r) (* factor s)) (go label200))
              (cond
                ((and (< f one) (< (f2cl-lib:fref scale (i) ((1 *))) one))
                 (if
                  (<=
                   (* f (f2cl-lib:fref scale-%data% (i) ((1 *)) scale-%offset%))
                   sfmin1)
                  (go label200))))
              (cond
                ((and (> f one) (> (f2cl-lib:fref scale (i) ((1 *))) one))
                 (if
                  (>= (f2cl-lib:fref scale-%data% (i) ((1 *)) scale-%offset%)
                      (/ sfmax1 f))
                  (go label200))))
              (setf g (/ one f))
              (setf (f2cl-lib:fref scale-%data% (i) ((1 *)) scale-%offset%)
                      (* (f2cl-lib:fref scale-%data% (i) ((1 *)) scale-%offset%)
                         f))
              (setf noconv t)
              (dscal (f2cl-lib:int-add (f2cl-lib:int-sub n k) 1) g
               (f2cl-lib:array-slice a double-float (i k) ((1 lda) (1 *))) lda)
              (dscal l f
               (f2cl-lib:array-slice a double-float (1 i) ((1 lda) (1 *))) 1)
label200))
       (if noconv (go label140))
label210
       (setf ilo k)
       (setf ihi l)
end_label
       (return (values nil nil nil nil ilo ihi nil info))))))
```

7.6 dgebd2 LAPACK

⟨*dgebd2.input*⟩≡

```
)set break resume
)sys rm -f dgebd2.output
)spool dgebd2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgebd2.help*⟩≡

```
======================================================================
dgebd2 examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DGEBD2  -  a  real general m by n matrix A to upper or lower bidiagonal
       form B by an orthogonal transformation

SYNOPSIS
       SUBROUTINE DGEBD2( M, N, A, LDA, D, E, TAUQ, TAUP, WORK, INFO )

           INTEGER        INFO, LDA, M, N

           DOUBLE         PRECISION A( LDA, * ), D( * ), E( * ),  TAUP(  *  ),
                          TAUQ( * ), WORK( * )

PURPOSE
       DGEBD2 reduces a real general m by n matrix A to upper or lower bidiag-
       onal form B by an orthogonal transformation: Q' * A * P = B.

       If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal.


ARGUMENTS
       M       (input) INTEGER
               The number of rows in the matrix A.  M >= 0.

       N       (input) INTEGER
               The number of columns in the matrix A.  N >= 0.

       A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
               On entry, the m by n general matrix to be reduced.  On exit, if
               m  >= n, the diagonal and the first superdiagonal are overwrit-
               ten with the upper bidiagonal matrix B; the elements below  the
               diagonal,  with the array TAUQ, represent the orthogonal matrix
               Q as a product of elementary reflectors, and the elements above
               the  first  superdiagonal,  with  the array TAUP, represent the
               orthogonal matrix P as a product of elementary reflectors; if m
               <  n,  the  diagonal  and the first subdiagonal are overwritten
               with the lower bidiagonal matrix  B;  the  elements  below  the
               first  subdiagonal, with the array TAUQ, represent the orthogo-
```

```
            nal matrix Q as a product of  elementary  reflectors,  and  the
            elements above the diagonal, with the array TAUP, represent the
            orthogonal matrix P as a product of elementary reflectors.  See
            Further Details.  LDA     (input) INTEGER The leading dimension
            of the array A.  LDA >= max(1,M).

    D       (output) DOUBLE PRECISION array, dimension (min(M,N))
            The diagonal elements  of  the  bidiagonal  matrix  B:  D(i)  =
            A(i,i).

    E       (output) DOUBLE PRECISION array, dimension (min(M,N)-1)
            The  off-diagonal  elements of the bidiagonal matrix B: if m >=
            n, E(i) = A(i,i+1) for i =  1,2,...,n-1;  if  m  <  n,  E(i)  =
            A(i+1,i) for i = 1,2,...,m-1.

    TAUQ    (output) DOUBLE PRECISION array dimension (min(M,N))
            The scalar factors of the elementary reflectors which represent
            the orthogonal matrix Q. See Further Details.  TAUP    (output)
            DOUBLE PRECISION array, dimension (min(M,N)) The scalar factors
            of the elementary reflectors  which  represent  the  orthogonal
            matrix P. See Further Details.  WORK    (workspace) DOUBLE PRE-
            CISION array, dimension (max(M,N))

    INFO    (output) INTEGER
            = 0: successful exit.
            < 0: if INFO = -i, the i-th argument had an illegal value.

FURTHER DETAILS
       The matrices Q and P are represented as products of elementary  reflec-
       tors:

       If m >= n,

          Q = H(1) H(2) . . . H(n)  and  P = G(1) G(2) . . . G(n-1)

       Each H(i) and G(i) has the form:

          H(i) = I - tauq * v * v'  and G(i) = I - taup * u * u'

       where  tauq  and  taup  are real scalars, and v and u are real vectors;
       v(1:i-1) = 0, v(i) = 1, and v(i+1:m) is stored on exit  in  A(i+1:m,i);
       u(1:i)  =  0, u(i+1) = 1, and u(i+2:n) is stored on exit in A(i,i+2:n);
       tauq is stored in TAUQ(i) and taup in TAUP(i).

       If m < n,
```

```
   Q = H(1) H(2) . . . H(m-1)  and  P = G(1) G(2) . . . G(m)

Each H(i) and G(i) has the form:

   H(i) = I - tauq * v * v'  and G(i) = I - taup * u * u'

where tauq and taup are real scalars, and v and  u  are  real  vectors;
v(1:i)  =  0, v(i+1) = 1, and v(i+2:m) is stored on exit in A(i+2:m,i);
u(1:i-1) = 0, u(i) = 1, and u(i+1:n) is stored on exit  in  A(i,i+1:n);
tauq is stored in TAUQ(i) and taup in TAUP(i).

The contents of A on exit are illustrated by the following examples:

m = 6 and n = 5 (m > n):          m = 5 and n = 6 (m < n):

  (  d   e   u1  u1  u1 )           (  d   u1  u1  u1  u1  u1 )
  (  v1  d   e   u2  u2 )           (  e   d   u2  u2  u2  u2 )
  (  v1  v2  d   e   u3 )           (  v1  e   d   u3  u3  u3 )
  (  v1  v2  v3  d   e  )           (  v1  v2  e   d   u4  u4 )
  (  v1  v2  v3  v4  d  )           (  v1  v2  v3  e   d   u5 )
  (  v1  v2  v3  v4  v5 )

where  d  and  e  denote  diagonal  and  off-diagonal elements of B, vi
denotes an element of the vector defining H(i), and ui  an  element  of
the vector defining G(i).
```

⟨*LAPACK dgebd2*⟩≡

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dgebd2 (m n a lda d e tauq taup work info)
    (declare (type (simple-array double-float (*)) work taup tauq e d a)
             (type fixnum info lda n m))
    (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (d double-float d-%data% d-%offset%)
         (e double-float e-%data% e-%offset%)
         (tauq double-float tauq-%data% tauq-%offset%)
         (taup double-float taup-%data% taup-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((i 0))
        (declare (type fixnum i))
        (setf info 0)
        (cond
          ((< m 0)
           (setf info -1))
          ((< n 0)
           (setf info -2))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info -4)))
        (cond
          ((< info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DGEBD2" (f2cl-lib:int-sub info))
           (go end_label)))
        (cond
          ((>= m n)
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                         ((> i n) nil)
             (tagbody
               (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                   (dlarfg (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
                    (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                    (f2cl-lib:array-slice a
                                          double-float
                                          ((min (f2cl-lib:int-add i 1) m) i)
                                          ((1 lda) (1 *)))
                    1 (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%))
                 (declare (ignore var-0 var-2 var-3))
                 (setf (f2cl-lib:fref a-%data%
                                      (i i)
```

```
                        ((1 lda) (1 *))
                        a-%offset%)
          var-1)
  (setf (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
          var-4))
(setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
        (f2cl-lib:fref a-%data%
                        (i i)
                        ((1 lda) (1 *))
                        a-%offset%))
(setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
        one)
(dlarf "Left" (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
 (f2cl-lib:int-sub n i)
 (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) 1
 (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
 (f2cl-lib:array-slice a
                        double-float
                        (i (f2cl-lib:int-add i 1))
                        ((1 lda) (1 *)))
 lda work)
(setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
        (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
(cond
  ((< i n)
   (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
       (dlarfg (f2cl-lib:int-sub n i)
        (f2cl-lib:fref a-%data%
                        (i (f2cl-lib:int-add i 1))
                        ((1 lda) (1 *))
                        a-%offset%)
        (f2cl-lib:array-slice a
                              double-float
                              (i
                               (min
                                (the fixnum
                                     (f2cl-lib:int-add i 2))
                                (the fixnum n)))
                              ((1 lda) (1 *)))
        lda
        (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%))
     (declare (ignore var-0 var-2 var-3))
     (setf (f2cl-lib:fref a-%data%
                          (i (f2cl-lib:int-add i 1))
                          ((1 lda) (1 *))
                          a-%offset%)
```

```
                    var-1)
              (setf (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
                    var-4))
             (setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
                   (f2cl-lib:fref a-%data%
                                  (i (f2cl-lib:int-add i 1))
                                  ((1 lda) (1 *))
                                  a-%offset%))
             (setf (f2cl-lib:fref a-%data%
                                  (i (f2cl-lib:int-add i 1))
                                  ((1 lda) (1 *))
                                  a-%offset%)
                   one)
             (dlarf "Right" (f2cl-lib:int-sub m i) (f2cl-lib:int-sub n i)
              (f2cl-lib:array-slice a
                                    double-float
                                    (i (f2cl-lib:int-add i 1))
                                    ((1 lda) (1 *)))
              lda (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
              (f2cl-lib:array-slice a
                                    double-float
                                    ((+ i 1) (f2cl-lib:int-add i 1))
                                    ((1 lda) (1 *)))
              lda work)
             (setf (f2cl-lib:fref a-%data%
                                  (i (f2cl-lib:int-add i 1))
                                  ((1 lda) (1 *))
                                  a-%offset%)
                   (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)))
            (t
             (setf (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
                   zero))))))
(t
 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
               ((> i m) nil)
   (tagbody
     (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
         (dlarfg (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
          (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
          (f2cl-lib:array-slice a
                                double-float
                                (i
                                 (min
                                  (the fixnum
                                       (f2cl-lib:int-add i 1))
                                  (the fixnum n)))
```

```
                          ((1 lda) (1 *)))
     lda (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%))
  (declare (ignore var-0 var-2 var-3))
  (setf (f2cl-lib:fref a-%data%
                       (i i)
                       ((1 lda) (1 *))
                       a-%offset%)
          var-1)
  (setf (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
          var-4))
(setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
        (f2cl-lib:fref a-%data%
                       (i i)
                       ((1 lda) (1 *))
                       a-%offset%))
(setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
        one)
(dlarf "Right" (f2cl-lib:int-sub m i)
 (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
 (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
 (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
 (f2cl-lib:array-slice a
                       double-float
                       ((min (f2cl-lib:int-add i 1) m) i)
                       ((1 lda) (1 *)))
 lda work)
(setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
        (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
(cond
  ((< i m)
   (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
       (dlarfg (f2cl-lib:int-sub m i)
        (f2cl-lib:fref a-%data%
                       ((f2cl-lib:int-add i 1) i)
                       ((1 lda) (1 *))
                       a-%offset%)
        (f2cl-lib:array-slice a
                              double-float
                              ((min (f2cl-lib:int-add i 2) m) i)
                              ((1 lda) (1 *)))
        1 (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%))
     (declare (ignore var-0 var-2 var-3))
     (setf (f2cl-lib:fref a-%data%
                          ((f2cl-lib:int-add i 1) i)
                          ((1 lda) (1 *))
                          a-%offset%)
```

```
                          var-1)
                    (setf (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
                          var-4))
                   (setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
                         (f2cl-lib:fref a-%data%
                                        ((f2cl-lib:int-add i 1) i)
                                        ((1 lda) (1 *))
                                        a-%offset%))
                   (setf (f2cl-lib:fref a-%data%
                                        ((f2cl-lib:int-add i 1) i)
                                        ((1 lda) (1 *))
                                        a-%offset%)
                         one)
                   (dlarf "Left" (f2cl-lib:int-sub m i) (f2cl-lib:int-sub n i)
                    (f2cl-lib:array-slice a
                                          double-float
                                          ((+ i 1) i)
                                          ((1 lda) (1 *)))
                    1 (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
                    (f2cl-lib:array-slice a
                                          double-float
                                          ((+ i 1) (f2cl-lib:int-add i 1))
                                          ((1 lda) (1 *)))
                    lda work)
                   (setf (f2cl-lib:fref a-%data%
                                        ((f2cl-lib:int-add i 1) i)
                                        ((1 lda) (1 *))
                                        a-%offset%)
                         (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)))
                  (t
                   (setf (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
                         zero)))))))
  end_label
       (return (values nil nil nil nil nil nil nil nil nil info))))))
```

7.7 dgebrd LAPACK

⟨*dgebrd.input*⟩≡

```
)set break resume
)sys rm -f dgebrd.output
)spool dgebrd.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgebrd.help*⟩≡

```
  ======================================================================
  dgebrd examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DGEBRD  -  a  general real M-by-N matrix A to upper or lower bidiagonal
         form B by an orthogonal transformation

  SYNOPSIS
         SUBROUTINE DGEBRD( M, N, A, LDA, D, E, TAUQ, TAUP, WORK, LWORK, INFO )

             INTEGER        INFO, LDA, LWORK, M, N

             DOUBLE         PRECISION A( LDA, * ), D( * ), E( * ),  TAUP(  *  ),
                            TAUQ( * ), WORK( * )

  PURPOSE
         DGEBRD reduces a general real M-by-N matrix A to upper or lower bidiag-
         onal form B by an orthogonal transformation: Q**T * A * P = B.

         If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal.


  ARGUMENTS
         M       (input) INTEGER
                 The number of rows in the matrix A.  M >= 0.

         N       (input) INTEGER
                 The number of columns in the matrix A.  N >= 0.

         A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
                 On entry, the M-by-N general matrix to be reduced.  On exit, if
                 m  >= n, the diagonal and the first superdiagonal are overwrit-
                 ten with the upper bidiagonal matrix B; the elements below  the
                 diagonal,  with the array TAUQ, represent the orthogonal matrix
                 Q as a product of elementary reflectors, and the elements above
                 the  first  superdiagonal,  with  the array TAUP, represent the
                 orthogonal matrix P as a product of elementary reflectors; if m
                 <  n,  the  diagonal  and the first subdiagonal are overwritten
                 with the lower bidiagonal matrix  B;  the  elements  below  the
                 first  subdiagonal, with the array TAUQ, represent the orthogo-
```

```
nal matrix Q as a product of elementary reflectors, and the
elements above the diagonal, with the array TAUP, represent the
orthogonal matrix P as a product of elementary reflectors. See
Further Details. LDA     (input) INTEGER The leading dimension
of the array A.  LDA >= max(1,M).

D       (output) DOUBLE PRECISION array, dimension (min(M,N))
        The diagonal elements of the bidiagonal matrix B: D(i) =
        A(i,i).

E       (output) DOUBLE PRECISION array, dimension (min(M,N)-1)
        The off-diagonal elements of the bidiagonal matrix B: if m >=
        n, E(i) = A(i,i+1) for i = 1,2,...,n-1; if m < n, E(i) =
        A(i+1,i) for i = 1,2,...,m-1.

TAUQ    (output) DOUBLE PRECISION array dimension (min(M,N))
        The scalar factors of the elementary reflectors which represent
        the orthogonal matrix Q. See Further Details. TAUP    (output)
        DOUBLE PRECISION array, dimension (min(M,N)) The scalar factors
        of the elementary reflectors which represent the orthogonal
        matrix P. See Further Details. WORK    (workspace/output) DOU-
        BLE PRECISION array, dimension (MAX(1,LWORK)) On exit, if INFO
        = 0, WORK(1) returns the optimal LWORK.

LWORK   (input) INTEGER
        The length of the array WORK. LWORK >= max(1,M,N). For opti-
        mum performance LWORK >= (M+N)*NB, where NB is the optimal
        blocksize.

        If LWORK = -1, then a workspace query is assumed; the routine
        only calculates the optimal size of the WORK array, returns
        this value as the first entry of the WORK array, and no error
        message related to LWORK is issued by XERBLA.

INFO    (output) INTEGER
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value.

FURTHER DETAILS
      The matrices Q and P are represented as products of elementary reflec-
      tors:

      If m >= n,

         Q = H(1) H(2) . . . H(n)  and  P = G(1) G(2) . . . G(n-1)
```

```
Each H(i) and G(i) has the form:

   H(i) = I - tauq * v * v'  and G(i) = I - taup * u * u'

where  tauq  and  taup  are real scalars, and v and u are real vectors;
v(1:i-1) = 0, v(i) = 1, and v(i+1:m) is stored on exit  in  A(i+1:m,i);
u(1:i)  =  0, u(i+1) = 1, and u(i+2:n) is stored on exit in A(i,i+2:n);
tauq is stored in TAUQ(i) and taup in TAUP(i).

If m < n,

   Q = H(1) H(2) . . . H(m-1)  and  P = G(1) G(2) . . . G(m)

Each H(i) and G(i) has the form:

   H(i) = I - tauq * v * v'  and G(i) = I - taup * u * u'

where tauq and taup are real scalars, and v and  u  are  real  vectors;
v(1:i)  =  0, v(i+1) = 1, and v(i+2:m) is stored on exit in A(i+2:m,i);
u(1:i-1) = 0, u(i) = 1, and u(i+1:n) is stored on exit  in  A(i,i+1:n);
tauq is stored in TAUQ(i) and taup in TAUP(i).

The contents of A on exit are illustrated by the following examples:

m = 6 and n = 5 (m > n):          m = 5 and n = 6 (m < n):

  (  d   e   u1  u1  u1 )           (  d   u1  u1  u1  u1  u1 )
  (  v1  d   e   u2  u2 )           (  e   d   u2  u2  u2  u2 )
  (  v1  v2  d   e   u3 )           (  v1  e   d   u3  u3  u3 )
  (  v1  v2  v3  d   e  )           (  v1  v2  e   d   u4  u4 )
  (  v1  v2  v3  v4  d  )           (  v1  v2  v3  e   d   u5 )
  (  v1  v2  v3  v4  v5 )

where  d  and  e  denote  diagonal  and  off-diagonal elements of B, vi
denotes an element of the vector defining H(i), and ui  an  element  of
the vector defining G(i).
```

⟨*LAPACK dgebrd*⟩≡

```
(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dgebrd (m n a lda d e tauq taup work lwork info)
    (declare (type (simple-array double-float (*)) work taup tauq e d a)
             (type fixnum info lwork lda n m))
    (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (d double-float d-%data% d-%offset%)
         (e double-float e-%data% e-%offset%)
         (tauq double-float tauq-%data% tauq-%offset%)
         (taup double-float taup-%data% taup-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((ws 0.0) (i 0) (iinfo 0) (j 0) (ldwrkx 0) (ldwrky 0) (lwkopt 0)
             (minmn 0) (nb 0) (nbmin 0) (nx 0) (lquery nil))
        (declare (type (double-float) ws)
                 (type fixnum i iinfo j ldwrkx ldwrky lwkopt minmn
                                       nb nbmin nx)
                 (type (member t nil) lquery))
        (setf info 0)
        (setf nb
                (max (the fixnum 1)
                     (the fixnum
                          (ilaenv 1 "DGEBRD" " " m n -1 -1))))
        (setf lwkopt (f2cl-lib:int-mul (f2cl-lib:int-add m n) nb))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                (coerce (realpart lwkopt) 'double-float))
        (setf lquery (coerce (= lwork -1) '(member t nil)))
        (cond
          ((< m 0)
           (setf info -1))
          ((< n 0)
           (setf info -2))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info -4))
          ((and
            (< lwork
               (max (the fixnum 1)
                    (the fixnum m)
                    (the fixnum n)))
            (not lquery))
           (setf info -10)))
        (cond
          ((< info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
```

```
    "DGEBRD" (f2cl-lib:int-sub info))
   (go end_label))
  (lquery
   (go end_label)))
(setf minmn (min (the fixnum m) (the fixnum n)))
(cond
  ((= minmn 0)
   (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
           (coerce (the fixnum 1) 'double-float))
   (go end_label)))
(setf ws
        (coerce
         (the fixnum
              (max (the fixnum m)
                   (the fixnum n)))
         'double-float))
(setf ldwrkx m)
(setf ldwrky n)
(cond
  ((and (> nb 1) (< nb minmn))
   (setf nx
           (max (the fixnum nb)
                (the fixnum
                     (ilaenv 3 "DGEBRD" " " m n -1 -1))))
   (cond
     ((< nx minmn)
      (setf ws
              (coerce
               (the fixnum
                    (f2cl-lib:int-mul (f2cl-lib:int-add m n) nb))
               'double-float))
      (cond
        ((< lwork ws)
         (setf nbmin (ilaenv 2 "DGEBRD" " " m n -1 -1))
         (cond
           ((>= lwork (f2cl-lib:int-mul (f2cl-lib:int-add m n) nbmin))
            (setf nb (the fixnum (truncate lwork (+ m n)))))
           (t
            (setf nb 1)
            (setf nx minmn))))))))
  (t
   (setf nx minmn)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i nb))
              ((> i (f2cl-lib:int-add minmn (f2cl-lib:int-sub nx))) nil)
  (tagbody
    (dlabrd (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
```

```
 (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) nb
 (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
 (f2cl-lib:array-slice d double-float (i) ((1 *)))
 (f2cl-lib:array-slice e double-float (i) ((1 *)))
 (f2cl-lib:array-slice tauq double-float (i) ((1 *)))
 (f2cl-lib:array-slice taup double-float (i) ((1 *))) work ldwrkx
 (f2cl-lib:array-slice work
                       double-float
                       ((+ (f2cl-lib:int-mul ldwrkx nb) 1))
                       ((1 *)))
 ldwrky)
(dgemm "No transpose" "Transpose"
 (f2cl-lib:int-add (f2cl-lib:int-sub m i nb) 1)
 (f2cl-lib:int-add (f2cl-lib:int-sub n i nb) 1) nb (- one)
 (f2cl-lib:array-slice a double-float ((+ i nb) i) ((1 lda) (1 *)))
 lda
 (f2cl-lib:array-slice work
                       double-float
                       ((+ (f2cl-lib:int-mul ldwrkx nb) nb 1))
                       ((1 *)))
 ldwrky one
 (f2cl-lib:array-slice a
                       double-float
                       ((+ i nb) (f2cl-lib:int-add i nb))
                       ((1 lda) (1 *)))
 lda)
(dgemm "No transpose" "No transpose"
 (f2cl-lib:int-add (f2cl-lib:int-sub m i nb) 1)
 (f2cl-lib:int-add (f2cl-lib:int-sub n i nb) 1) nb (- one)
 (f2cl-lib:array-slice work double-float ((+ nb 1)) ((1 *))) ldwrkx
 (f2cl-lib:array-slice a
                       double-float
                       (i (f2cl-lib:int-add i nb))
                       ((1 lda) (1 *)))
 lda one
 (f2cl-lib:array-slice a
                       double-float
                       ((+ i nb) (f2cl-lib:int-add i nb))
                       ((1 lda) (1 *)))
 lda)
(cond
  ((>= m n)
   (f2cl-lib:fdo (j i (f2cl-lib:int-add j 1))
                 ((> j
                     (f2cl-lib:int-add i nb (f2cl-lib:int-sub 1)))
                  nil)
```

```
                  (tagbody
                    (setf (f2cl-lib:fref a-%data%
                                         (j j)
                                         ((1 lda) (1 *))
                                         a-%offset%)
                            (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))
                    (setf (f2cl-lib:fref a-%data%
                                         (j (f2cl-lib:int-add j 1))
                                         ((1 lda) (1 *))
                                         a-%offset%)
                            (f2cl-lib:fref e-%data% (j) ((1 *)) e-%offset%)))))
                 (t
                  (f2cl-lib:fdo (j i (f2cl-lib:int-add j 1))
                                ((> j
                                    (f2cl-lib:int-add i nb (f2cl-lib:int-sub 1)))
                                 nil)
                    (tagbody
                      (setf (f2cl-lib:fref a-%data%
                                           (j j)
                                           ((1 lda) (1 *))
                                           a-%offset%)
                              (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))
                      (setf (f2cl-lib:fref a-%data%
                                           ((f2cl-lib:int-add j 1) j)
                                           ((1 lda) (1 *))
                                           a-%offset%)
                            (f2cl-lib:fref e-%data% (j) ((1 *)) e-%offset%)))))))
          (multiple-value-bind
                (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
              (dgebd2 (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
               (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
               (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
               (f2cl-lib:array-slice d double-float (i) ((1 *)))
               (f2cl-lib:array-slice e double-float (i) ((1 *)))
               (f2cl-lib:array-slice tauq double-float (i) ((1 *)))
               (f2cl-lib:array-slice taup double-float (i) ((1 *))) work iinfo)
            (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                             var-8))
            (setf iinfo var-9))
          (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) ws)
  end_label
          (return (values nil nil nil nil nil nil nil nil nil nil info))))))
```

7.8 dgeev LAPACK

⟨*dgeev.input*⟩≡

```
)set break resume
)sys rm -f dgeev.output
)spool dgeev.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgeev.help*⟩≡

```
  ======================================================================
  dgeev examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DGEEV  - for an N-by-N real nonsymmetric matrix A, the eigenvalues and,
         optionally, the left and/or right eigenvectors

  SYNOPSIS
         SUBROUTINE DGEEV( JOBVL, JOBVR, N, A, LDA, WR, WI, VL, LDVL, VR,  LDVR,
                           WORK, LWORK, INFO )

             CHARACTER     JOBVL, JOBVR

             INTEGER       INFO, LDA, LDVL, LDVR, LWORK, N

             DOUBLE        PRECISION  A( LDA, * ), VL( LDVL, * ), VR( LDVR, * ),
                           WI( * ), WORK( * ), WR( * )

  PURPOSE
         DGEEV computes for an N-by-N real nonsymmetric matrix A, the  eigenval-
         ues and, optionally, the left and/or right eigenvectors.

         The right eigenvector v(j) of A satisfies
                          A * v(j) = lambda(j) * v(j)
         where lambda(j) is its eigenvalue.
         The left eigenvector u(j) of A satisfies
                       u(j)**H * A = lambda(j) * u(j)**H
         where u(j)**H denotes the conjugate transpose of u(j).

         The  computed  eigenvectors are normalized to have Euclidean norm equal
         to 1 and largest component real.


  ARGUMENTS
         JOBVL   (input) CHARACTER*1
                 = 'N': left eigenvectors of A are not computed;
                 = 'V': left eigenvectors of A are computed.

         JOBVR   (input) CHARACTER*1
                 = 'N': right eigenvectors of A are not computed;
```

```
          = 'V': right eigenvectors of A are computed.

N         (input) INTEGER
          The order of the matrix A. N >= 0.

A         (input/output) DOUBLE PRECISION array, dimension (LDA,N)
          On entry, the N-by-N matrix A.  On exit, A has  been  overwrit-
          ten.

LDA       (input) INTEGER
          The leading dimension of the array A.  LDA >= max(1,N).

WR        (output) DOUBLE PRECISION array, dimension (N)
          WI        (output)  DOUBLE PRECISION array, dimension (N) WR and
          WI contain the real and imaginary parts, respectively,  of  the
          computed  eigenvalues.   Complex conjugate pairs of eigenvalues
          appear consecutively with the eigenvalue  having  the  positive
          imaginary part first.

VL        (output) DOUBLE PRECISION array, dimension (LDVL,N)
          If JOBVL = 'V', the left eigenvectors u(j) are stored one after
          another in the columns of VL, in the same order as their eigen-
          values.   If  JOBVL  =  'N', VL is not referenced.  If the j-th
          eigenvalue is real, then u(j) = VL(:,j), the j-th column of VL.
          If  the  j-th and (j+1)-st eigenvalues form a complex conjugate
          pair, then u(j) = VL(:,j) + i*VL(:,j+1) and
          u(j+1) = VL(:,j) - i*VL(:,j+1).

LDVL      (input) INTEGER
          The leading dimension of the array VL.  LDVL >= 1; if  JOBVL  =
          'V', LDVL >= N.

VR        (output) DOUBLE PRECISION array, dimension (LDVR,N)
          If  JOBVR  =  'V',  the  right eigenvectors v(j) are stored one
          after another in the columns of VR, in the same order as  their
          eigenvalues.   If JOBVR = 'N', VR is not referenced.  If the j-
          th eigenvalue is real, then v(j) = VR(:,j), the j-th column  of
          VR.  If the j-th and (j+1)-st eigenvalues form a complex conju-
          gate pair, then v(j) = VR(:,j) + i*VR(:,j+1) and
          v(j+1) = VR(:,j) - i*VR(:,j+1).

LDVR      (input) INTEGER
          The leading dimension of the array VR.  LDVR >= 1; if  JOBVR  =
          'V', LDVR >= N.

WORK        (workspace/output)   DOUBLE   PRECISION   array,   dimension
```

```
(MAX(1,LWORK))
        On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK   (input) INTEGER
        The  dimension  of the array WORK.  LWORK >= max(1,3*N), and if
        JOBVL = 'V' or JOBVR = 'V', LWORK >=  4*N.   For  good  perfor-
        mance, LWORK must generally be larger.

        If  LWORK  = -1, then a workspace query is assumed; the routine
        only calculates the optimal size of  the  WORK  array,  returns
        this  value  as the first entry of the WORK array, and no error
        message related to LWORK is issued by XERBLA.

INFO    (output) INTEGER
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value.
        > 0:  if INFO = i, the QR algorithm failed to compute  all  the
        eigenvalues,  and  no eigenvectors have been computed; elements
        i+1:N of WR and WI contain eigenvalues which have converged.
```

⟨*LAPACK dgeev*⟩≡

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dgeev (jobvl jobvr n a lda wr wi vl ldvl vr ldvr work lwork info)
    (declare (type (simple-array double-float (*)) work vr vl wi wr a)
             (type fixnum info lwork ldvr ldvl lda n)
             (type character jobvr jobvl))
    (f2cl-lib:with-multi-array-data
        ((jobvl character jobvl-%data% jobvl-%offset%)
         (jobvr character jobvr-%data% jobvr-%offset%)
         (a double-float a-%data% a-%offset%)
         (wr double-float wr-%data% wr-%offset%)
         (wi double-float wi-%data% wi-%offset%)
         (vl double-float vl-%data% vl-%offset%)
         (vr double-float vr-%data% vr-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((dum (make-array 1 :element-type 'double-float))
             (select (make-array 1 :element-type 't)) (anrm 0.0) (bignum 0.0)
             (cs 0.0) (cscale 0.0) (eps 0.0) (r 0.0) (scl 0.0) (smlnum 0.0)
             (sn 0.0) (hswork 0) (i 0) (ibal 0) (ierr 0) (ihi 0) (ilo 0)
             (itau 0) (iwrk 0) (k 0) (maxb 0) (maxwrk 0) (minwrk 0) (nout 0)
             (side
              (make-array '(1) :element-type 'character :initial-element #\ ))
             (lquery nil) (scalea nil) (wantvl nil) (wantvr nil))
        (declare (type (simple-array double-float (1)) dum)
                 (type (simple-array (member t nil) (1)) select)
                 (type (double-float) anrm bignum cs cscale eps r scl smlnum
                                      sn)
                 (type fixnum hswork i ibal ierr ihi ilo itau iwrk
                                          k maxb maxwrk minwrk nout)
                 (type (simple-array character (1)) side)
                 (type (member t nil) lquery scalea wantvl wantvr))
        (setf info 0)
        (setf lquery (coerce (= lwork -1) '(member t nil)))
        (setf wantvl (char-equal jobvl #\V))
        (setf wantvr (char-equal jobvr #\V))
        (cond
          ((and (not wantvl) (not (char-equal jobvl #\N)))
           (setf info -1))
          ((and (not wantvr) (not (char-equal jobvr #\N)))
           (setf info -2))
          ((< n 0)
           (setf info -3))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info -5))
```

```
          ((or (< ldvl 1) (and wantvl (< ldvl n)))
           (setf info -9))
          ((or (< ldvr 1) (and wantvr (< ldvr n)))
           (setf info -11)))
        (setf minwrk 1)
        (cond
          ((and (= info 0) (or (>= lwork 1) lquery))
           (setf maxwrk
                 (f2cl-lib:int-add (f2cl-lib:int-mul 2 n)
                                   (f2cl-lib:int-mul n
                                                     (ilaenv 1 "DGEHRD" " " n
                                                      1 n 0))))
           (cond
             ((and (not wantvl) (not wantvr))
              (setf minwrk
                    (max (the fixnum 1)
                         (the fixnum (f2cl-lib:int-mul 3 n))))
              (setf maxb
                    (max
                     (the fixnum
                          (ilaenv 8 "DHSEQR" "EN" n 1 n -1))
                     (the fixnum 2)))
              (setf k
                    (min (the fixnum maxb)
                         (the fixnum n)
                         (the fixnum
                              (max (the fixnum 2)
                                   (the fixnum
                                        (ilaenv 4 "DHSEQR" "EN" n 1 n -1))))))
              (setf hswork
                    (max
                     (the fixnum
                          (f2cl-lib:int-mul k (f2cl-lib:int-add k 2)))
                     (the fixnum (f2cl-lib:int-mul 2 n))))
              (setf maxwrk
                    (max (the fixnum maxwrk)
                         (the fixnum (f2cl-lib:int-add n 1))
                         (the fixnum
                              (f2cl-lib:int-add n hswork)))))
             (t
              (setf minwrk
                    (max (the fixnum 1)
                         (the fixnum (f2cl-lib:int-mul 4 n))))
              (setf maxwrk
                    (max (the fixnum maxwrk)
                         (the fixnum
```

```
                                (f2cl-lib:int-add (f2cl-lib:int-mul 2 n)
                                                  (f2cl-lib:int-mul
                                                   (f2cl-lib:int-sub n 1)
                                                   (ilaenv 1 "DORGHR" " " n 1 n
                                                    -1))))))
              (setf maxb
                      (max
                       (the fixnum
                            (ilaenv 8 "DHSEQR" "SV" n 1 n -1))
                       (the fixnum 2)))
              (setf k
                      (min (the fixnum maxb)
                           (the fixnum n)
                           (the fixnum
                                (max (the fixnum 2)
                                     (the fixnum
                                          (ilaenv 4 "DHSEQR" "SV" n 1 n -1))))))
              (setf hswork
                      (max
                       (the fixnum
                            (f2cl-lib:int-mul k (f2cl-lib:int-add k 2)))
                       (the fixnum (f2cl-lib:int-mul 2 n))))
              (setf maxwrk
                      (max (the fixnum maxwrk)
                           (the fixnum (f2cl-lib:int-add n 1))
                           (the fixnum (f2cl-lib:int-add n hswork))))
              (setf maxwrk
                      (max (the fixnum maxwrk)
                           (the fixnum (f2cl-lib:int-mul 4 n))))))
         (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                 (coerce (the fixnum maxwrk) 'double-float))))
      (cond
        ((and (< lwork minwrk) (not lquery))
         (setf info -13)))
      (cond
        ((/= info 0)
         (error
          " ** On entry to ~a parameter number ~a had an illegal value~%"
          "DGEEV " (f2cl-lib:int-sub info))
         (go end_label))
        (lquery
         (go end_label)))
      (if (= n 0) (go end_label))
      (setf eps (dlamch "P"))
      (setf smlnum (dlamch "S"))
      (setf bignum (/ one smlnum))
```

```
(multiple-value-bind (var-0 var-1)
    (dlabad smlnum bignum)
  (declare (ignore))
  (setf smlnum var-0)
  (setf bignum var-1))
(setf smlnum (/ (f2cl-lib:fsqrt smlnum) eps))
(setf bignum (/ one smlnum))
(setf anrm (dlange "M" n n a lda dum))
(setf scalea nil)
(cond
  ((and (> anrm zero) (< anrm smlnum))
   (setf scalea t)
   (setf cscale smlnum))
  ((> anrm bignum)
   (setf scalea t)
   (setf cscale bignum)))
(if scalea
    (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
        (dlascl "G" 0 0 anrm cscale n n a lda ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                       var-8))
      (setf ierr var-9)))
(setf ibal 1)
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgebal "B" n a lda ilo ihi
     (f2cl-lib:array-slice work double-float (ibal) ((1 *))) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-6))
  (setf ilo var-4)
  (setf ihi var-5)
  (setf ierr var-7))
(setf itau (f2cl-lib:int-add ibal n))
(setf iwrk (f2cl-lib:int-add itau n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
    (dgehrd n ilo ihi a lda
     (f2cl-lib:array-slice work double-float (itau) ((1 *)))
     (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7))
  (setf ierr var-8))
(cond
  (wantvl
   (f2cl-lib:f2cl-set-string side "L" (string 1))
   (dlacpy "L" n n a lda vl ldvl)
   (multiple-value-bind
```

```
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
     (dorghr n ilo ihi vl ldvl
      (f2cl-lib:array-slice work double-float (itau) ((1 *)))
      (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7))
   (setf ierr var-8))
 (setf iwrk itau)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
        var-10 var-11 var-12 var-13)
     (dhseqr "S" "V" n ilo ihi a lda wr wi vl ldvl
      (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                    var-8 var-9 var-10 var-11 var-12))
   (setf info var-13))
 (cond
   (wantvr
    (f2cl-lib:f2cl-set-string side "B" (string 1))
    (dlacpy "F" n n vl ldvl vr ldvr))))
(wantvr
 (f2cl-lib:f2cl-set-string side "R" (string 1))
 (dlacpy "L" n n a lda vr ldvr)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
     (dorghr n ilo ihi vr ldvr
      (f2cl-lib:array-slice work double-float (itau) ((1 *)))
      (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7))
   (setf ierr var-8))
 (setf iwrk itau)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
        var-10 var-11 var-12 var-13)
     (dhseqr "S" "V" n ilo ihi a lda wr wi vr ldvr
      (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                    var-8 var-9 var-10 var-11 var-12))
   (setf info var-13)))
(t
 (setf iwrk itau)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
```

```
           var-10 var-11 var-12 var-13)
         (dhseqr "E" "N" n ilo ihi a lda wr wi vr ldvr
          (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
          (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) info)
       (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                        var-8 var-9 var-10 var-11 var-12))
       (setf info var-13))))
(if (> info 0) (go label50))
(cond
  ((or wantvl wantvr)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
          var-10 var-11 var-12 var-13)
       (dtrevc side "B" select n a lda vl ldvl vr ldvr n nout
        (f2cl-lib:array-slice work double-float (iwrk) ((1 *))) ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                      var-8 var-9 var-10 var-12))
     (setf nout var-11)
     (setf ierr var-13))))
(cond
  (wantvl
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
       (dgebak "B" "L" n ilo ihi
        (f2cl-lib:array-slice work double-float (ibal) ((1 *))) n vl
        ldvl ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                      var-8))
     (setf ierr var-9))
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                 ((> i n) nil)
     (tagbody
       (cond
         ((= (f2cl-lib:fref wi (i) ((1 *))) zero)
          (setf scl
                  (/ one
                     (dnrm2 n
                      (f2cl-lib:array-slice vl
                                            double-float
                                            (1 i)
                                            ((1 ldvl) (1 *)))
                      1)))
          (dscal n scl
           (f2cl-lib:array-slice vl
                                 double-float
                                 (1 i)
```

```
                        ((1 ldvl) (1 *)))
  1))
((> (f2cl-lib:fref wi (i) ((1 *))) zero)
 (setf scl
        (/ one
           (dlapy2
            (dnrm2 n
             (f2cl-lib:array-slice vl
                                   double-float
                                   (1 i)
                                   ((1 ldvl) (1 *)))
             1)
            (dnrm2 n
             (f2cl-lib:array-slice vl
                                   double-float
                                   (1 (f2cl-lib:int-add i 1))
                                   ((1 ldvl) (1 *)))
             1))))
 (dscal n scl
  (f2cl-lib:array-slice vl
                        double-float
                        (1 i)
                        ((1 ldvl) (1 *)))
  1)
 (dscal n scl
  (f2cl-lib:array-slice vl
                        double-float
                        (1 (f2cl-lib:int-add i 1))
                        ((1 ldvl) (1 *)))
  1)
 (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
               ((> k n) nil)
   (tagbody
     (setf (f2cl-lib:fref work-%data%
                          ((f2cl-lib:int-sub
                            (f2cl-lib:int-add iwrk k)
                            1))
                          ((1 *))
                          work-%offset%)
            (+
             (expt
              (f2cl-lib:fref vl-%data%
                             (k i)
                             ((1 ldvl) (1 *))
                             vl-%offset%)
              2)
```

```
                              (expt
                               (f2cl-lib:fref vl-%data%
                                              (k (f2cl-lib:int-add i 1))
                                              ((1 ldvl) (1 *))
                                              vl-%offset%)
                               2)))))
                      (setf k
                            (idamax n
                             (f2cl-lib:array-slice work
                                                   double-float
                                                   (iwrk)
                                                   ((1 *)))
                             1))
                      (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                          (dlartg
                           (f2cl-lib:fref vl-%data%
                                          (k i)
                                          ((1 ldvl) (1 *))
                                          vl-%offset%)
                           (f2cl-lib:fref vl-%data%
                                          (k (f2cl-lib:int-add i 1))
                                          ((1 ldvl) (1 *))
                                          vl-%offset%)
                           cs sn r)
                        (declare (ignore var-0 var-1))
                        (setf cs var-2)
                        (setf sn var-3)
                        (setf r var-4))
                      (drot n
                       (f2cl-lib:array-slice vl
                                             double-float
                                             (1 i)
                                             ((1 ldvl) (1 *)))
                       1
                       (f2cl-lib:array-slice vl
                                             double-float
                                             (1 (f2cl-lib:int-add i 1))
                                             ((1 ldvl) (1 *)))
                       1 cs sn)
                      (setf (f2cl-lib:fref vl-%data%
                                           (k (f2cl-lib:int-add i 1))
                                           ((1 ldvl) (1 *))
                                           vl-%offset%)
                            zero)))))))
(cond
  (wantvr
```

```
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
    (dgebak "B" "R" n ilo ihi
     (f2cl-lib:array-slice work double-float (ibal) ((1 *))) n vr
     ldvr ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                   var-8))
  (setf ierr var-9))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i n) nil)
  (tagbody
    (cond
      ((= (f2cl-lib:fref wi (i) ((1 *))) zero)
       (setf scl
               (/ one
                  (dnrm2 n
                   (f2cl-lib:array-slice vr
                                         double-float
                                         (1 i)
                                         ((1 ldvr) (1 *)))
                   1)))
       (dscal n scl
        (f2cl-lib:array-slice vr
                              double-float
                              (1 i)
                              ((1 ldvr) (1 *)))
        1))
      ((> (f2cl-lib:fref wi (i) ((1 *))) zero)
       (setf scl
               (/ one
                  (dlapy2
                   (dnrm2 n
                    (f2cl-lib:array-slice vr
                                          double-float
                                          (1 i)
                                          ((1 ldvr) (1 *)))
                    1)
                   (dnrm2 n
                    (f2cl-lib:array-slice vr
                                          double-float
                                          (1 (f2cl-lib:int-add i 1))
                                          ((1 ldvr) (1 *)))
                    1))))
       (dscal n scl
        (f2cl-lib:array-slice vr
                              double-float
```

```
                        (1 i)
                        ((1 ldvr) (1 *)))
 1)
(dscal n scl
 (f2cl-lib:array-slice vr
                        double-float
                        (1 (f2cl-lib:int-add i 1))
                        ((1 ldvr) (1 *)))
 1)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k n) nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data%
                         ((f2cl-lib:int-sub
                           (f2cl-lib:int-add iwrk k)
                           1))
                         ((1 *))
                         work-%offset%)
            (+
             (expt
              (f2cl-lib:fref vr-%data%
                             (k i)
                             ((1 ldvr) (1 *))
                             vr-%offset%)
              2)
             (expt
              (f2cl-lib:fref vr-%data%
                             (k (f2cl-lib:int-add i 1))
                             ((1 ldvr) (1 *))
                             vr-%offset%)
              2)))))
(setf k
        (idamax n
         (f2cl-lib:array-slice work
                               double-float
                               (iwrk)
                               ((1 *)))
         1))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlartg
     (f2cl-lib:fref vr-%data%
                    (k i)
                    ((1 ldvr) (1 *))
                    vr-%offset%)
     (f2cl-lib:fref vr-%data%
                    (k (f2cl-lib:int-add i 1))
```

```
                                    ((1 ldvr) (1 *))
                                    vr-%offset%)
                        cs sn r)
                       (declare (ignore var-0 var-1))
                       (setf cs var-2)
                       (setf sn var-3)
                       (setf r var-4))
                      (drot n
                       (f2cl-lib:array-slice vr
                                             double-float
                                             (1 i)
                                             ((1 ldvr) (1 *)))
                       1
                       (f2cl-lib:array-slice vr
                                             double-float
                                             (1 (f2cl-lib:int-add i 1))
                                             ((1 ldvr) (1 *)))
                       1 cs sn)
                      (setf (f2cl-lib:fref vr-%data%
                                           (k (f2cl-lib:int-add i 1))
                                           ((1 ldvr) (1 *))
                                           vr-%offset%)
                            zero)))))))
label50
       (cond
         (scalea
          (multiple-value-bind
                (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
              (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub n info) 1
               (f2cl-lib:array-slice wr double-float ((+ info 1)) ((1 *)))
               (max (the fixnum (f2cl-lib:int-sub n info))
                    (the fixnum 1))
               ierr)
            (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                             var-8))
            (setf ierr var-9))
          (multiple-value-bind
                (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
              (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub n info) 1
               (f2cl-lib:array-slice wi double-float ((+ info 1)) ((1 *)))
               (max (the fixnum (f2cl-lib:int-sub n info))
                    (the fixnum 1))
               ierr)
            (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                             var-8))
            (setf ierr var-9))
```

```
            (cond
              ((> info 0)
               (multiple-value-bind
                     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                      var-9)
                   (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub ilo 1) 1 wr n
                    ierr)
                 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                  var-7 var-8))
                 (setf ierr var-9))
               (multiple-value-bind
                     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                      var-9)
                   (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub ilo 1) 1 wi n
                    ierr)
                 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                  var-7 var-8))
                 (setf ierr var-9))))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
              (coerce (the fixnum maxwrk) 'double-float))
  end_label
        (return
         (values nil nil nil nil nil nil nil nil nil nil nil nil nil info))))))
```

7.9 dgeevx LAPACK

⟨*dgeevx.input*⟩≡

```
)set break resume
)sys rm -f dgeevx.output
)spool dgeevx.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgeevx.help*⟩≡

```
========================================================================
dgeevx examples
========================================================================

========================================================================
Man Page Details
========================================================================

NAME
       DGEEVX - for an N-by-N real nonsymmetric matrix A, the eigenvalues and,
       optionally, the left and/or right eigenvectors

SYNOPSIS
       SUBROUTINE DGEEVX( BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, WR, WI,  VL,
                          LDVL,  VR,  LDVR,  ILO,  IHI,  SCALE, ABNRM, RCONDE,
                          RCONDV, WORK, LWORK, IWORK, INFO )

           CHARACTER      BALANC, JOBVL, JOBVR, SENSE

           INTEGER        IHI, ILO, INFO, LDA, LDVL, LDVR, LWORK, N

           DOUBLE         PRECISION ABNRM

           INTEGER        IWORK( * )

           DOUBLE         PRECISION A( LDA, * ), RCONDE( *  ),  RCONDV(  *  ),
                          SCALE(  *  ), VL( LDVL, * ), VR( LDVR, * ), WI( * ),
                          WORK( * ), WR( * )

PURPOSE
       DGEEVX computes for an N-by-N real nonsymmetric matrix A, the eigenval-
       ues and, optionally, the left and/or right eigenvectors.

       Optionally  also, it computes a balancing transformation to improve the
       conditioning of the eigenvalues and eigenvectors (ILO, IHI, SCALE,  and
       ABNRM),  reciprocal condition numbers for the eigenvalues (RCONDE), and
       reciprocal condition numbers for the right
       eigenvectors (RCONDV).

       The right eigenvector v(j) of A satisfies
                        A * v(j) = lambda(j) * v(j)
       where lambda(j) is its eigenvalue.
       The left eigenvector u(j) of A satisfies
                     u(j)**H * A = lambda(j) * u(j)**H
       where u(j)**H denotes the conjugate transpose of u(j).
```

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Balancing a matrix means permuting the rows and columns to make it more nearly upper triangular, and applying a diagonal similarity transformation D * A * D**(-1), where D is a diagonal matrix, to make its rows and columns closer in norm and the condition numbers of its eigenvalues and eigenvectors smaller. The computed reciprocal condition numbers correspond to the balanced matrix. Permuting rows and columns will not change the condition numbers (in exact arithmetic) but diagonal scaling will. For further explanation of balancing, see section 4.10.2 of the LAPACK Users' Guide.

ARGUMENTS

BALANC (input) CHARACTER*1
Indicates how the input matrix should be diagonally scaled and/or permuted to improve the conditioning of its eigenvalues. = 'N': Do not diagonally scale or permute;
= 'P': Perform permutations to make the matrix more nearly upper triangular. Do not diagonally scale; = 'S': Diagonally scale the matrix, i.e. replace A by D*A*D**(-1), where D is a diagonal matrix chosen to make the rows and columns of A more equal in norm. Do not permute; = 'B': Both diagonally scale and permute A.

Computed reciprocal condition numbers will be for the matrix after balancing and/or permuting. Permuting does not change condition numbers (in exact arithmetic), but balancing does.

JOBVL (input) CHARACTER*1
= 'N': left eigenvectors of A are not computed;
= 'V': left eigenvectors of A are computed. If SENSE = 'E' or 'B', JOBVL must = 'V'.

JOBVR (input) CHARACTER*1
= 'N': right eigenvectors of A are not computed;
= 'V': right eigenvectors of A are computed. If SENSE = 'E' or 'B', JOBVR must = 'V'.

SENSE (input) CHARACTER*1
Determines which reciprocal condition numbers are computed. = 'N': None are computed;
= 'E': Computed for eigenvalues only;
= 'V': Computed for right eigenvectors only;

```
          = 'B': Computed for eigenvalues and right eigenvectors.

          If  SENSE  =  'E' or 'B', both left and right eigenvectors must
          also be computed (JOBVL = 'V' and JOBVR = 'V').

N         (input) INTEGER
          The order of the matrix A. N >= 0.

A         (input/output) DOUBLE PRECISION array, dimension (LDA,N)
          On entry, the N-by-N matrix A.  On exit, A has  been  overwrit-
          ten.   If JOBVL = 'V' or JOBVR = 'V', A contains the real Schur
          form of the balanced version of the input matrix A.

LDA       (input) INTEGER
          The leading dimension of the array A.  LDA >= max(1,N).

WR        (output) DOUBLE PRECISION array, dimension (N)
          WI      (output) DOUBLE PRECISION array, dimension (N)  WR  and
          WI  contain  the real and imaginary parts, respectively, of the
          computed eigenvalues.  Complex conjugate pairs  of  eigenvalues
          will  appear consecutively with the eigenvalue having the posi-
          tive imaginary part first.

VL        (output) DOUBLE PRECISION array, dimension (LDVL,N)
          If JOBVL = 'V', the left eigenvectors u(j) are stored one after
          another in the columns of VL, in the same order as their eigen-
          values.  If JOBVL = 'N', VL is not  referenced.   If  the  j-th
          eigenvalue is real, then u(j) = VL(:,j), the j-th column of VL.
          If the j-th and (j+1)-st eigenvalues form a  complex  conjugate
          pair, then u(j) = VL(:,j) + i*VL(:,j+1) and
          u(j+1) = VL(:,j) - i*VL(:,j+1).

LDVL      (input) INTEGER
          The  leading  dimension of the array VL.  LDVL >= 1; if JOBVL =
          'V', LDVL >= N.

VR        (output) DOUBLE PRECISION array, dimension (LDVR,N)
          If JOBVR = 'V', the right  eigenvectors  v(j)  are  stored  one
          after  another in the columns of VR, in the same order as their
          eigenvalues.  If JOBVR = 'N', VR is not referenced.  If the  j-
          th  eigenvalue is real, then v(j) = VR(:,j), the j-th column of
          VR.  If the j-th and (j+1)-st eigenvalues form a complex conju-
          gate pair, then v(j) = VR(:,j) + i*VR(:,j+1) and
          v(j+1) = VR(:,j) - i*VR(:,j+1).

LDVR      (input) INTEGER
```

The leading dimension of the array VR. LDVR >= 1, and if JOBVR = 'V', LDVR >= N.

ILO (output) INTEGER
IHI (output) INTEGER ILO and IHI are integer values determined when A was balanced. The balanced A(i,j) = 0 if I > J and J = 1,...,ILO-1 or I = IHI+1,...,N.

SCALE (output) DOUBLE PRECISION array, dimension (N)
Details of the permutations and scaling factors applied when balancing A. If P(j) is the index of the row and column interchanged with row and column j, and D(j) is the scaling factor applied to row and column j, then SCALE(J) = P(J), for J = 1,...,ILO-1 = D(J), for J = ILO,...,IHI = P(J) for J = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

ABNRM (output) DOUBLE PRECISION
The one-norm of the balanced matrix (the maximum of the sum of absolute values of elements of any column).

RCONDE (output) DOUBLE PRECISION array, dimension (N)
RCONDE(j) is the reciprocal condition number of the j-th eigenvalue.

RCONDV (output) DOUBLE PRECISION array, dimension (N)
RCONDV(j) is the reciprocal condition number of the j-th right eigenvector.

WORK (workspace/output) DOUBLE PRECISION array, dimension (MAX(1,LWORK))
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. If SENSE = 'N' or 'E', LWORK >= max(1,2*N), and if JOBVL = 'V' or JOBVR = 'V', LWORK >= 3*N. If SENSE = 'V' or 'B', LWORK >= N*(N+6). For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK (workspace) INTEGER array, dimension (2*N-2)
If SENSE = 'N' or 'E', not referenced.

```
INFO    (output) INTEGER
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value.
        >  0:   if INFO = i, the QR algorithm failed to compute all the
        eigenvalues, and no eigenvectors or condition numbers have been
        computed;  elements  1:ILO-1  and  i+1:N  of  WR and WI contain
        eigenvalues which have converged.
```

⟨*LAPACK dgeevx*⟩≡

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dgeevx
         (balanc jobvl jobvr sense n a lda wr wi vl ldvl vr ldvr ilo ihi scale
          abnrm rconde rcondv work lwork iwork info)
    (declare (type (simple-array fixnum (*)) iwork)
             (type (double-float) abnrm)
             (type (simple-array double-float (*)) work rcondv rconde scale vr vl wi
                                                   wr a)
             (type fixnum info lwork ihi ilo ldvr ldvl lda n)
             (type character sense jobvr jobvl balanc))
    (f2cl-lib:with-multi-array-data
        ((balanc character balanc-%data% balanc-%offset%)
         (jobvl character jobvl-%data% jobvl-%offset%)
         (jobvr character jobvr-%data% jobvr-%offset%)
         (sense character sense-%data% sense-%offset%)
         (a double-float a-%data% a-%offset%)
         (wr double-float wr-%data% wr-%offset%)
         (wi double-float wi-%data% wi-%offset%)
         (vl double-float vl-%data% vl-%offset%)
         (vr double-float vr-%data% vr-%offset%)
         (scale double-float scale-%data% scale-%offset%)
         (rconde double-float rconde-%data% rconde-%offset%)
         (rcondv double-float rcondv-%data% rcondv-%offset%)
         (work double-float work-%data% work-%offset%)
         (iwork fixnum iwork-%data% iwork-%offset%))
      (prog ((dum (make-array 1 :element-type 'double-float))
             (select (make-array 1 :element-type 't)) (anrm 0.0) (bignum 0.0)
             (cs 0.0) (cscale 0.0) (eps 0.0) (r 0.0) (scl 0.0) (smlnum 0.0)
             (sn 0.0) (hswork 0) (i 0) (icond 0) (ierr 0) (itau 0) (iwrk 0)
             (k 0) (maxb 0) (maxwrk 0) (minwrk 0) (nout 0)
             (job
              (make-array '(1) :element-type 'character :initial-element #\ ))
             (side
              (make-array '(1) :element-type 'character :initial-element #\ ))
             (lquery nil) (scalea nil) (wantvl nil) (wantvr nil) (wntsnb nil)
             (wntsne nil) (wntsnn nil) (wntsnv nil))
        (declare (type (simple-array double-float (1)) dum)
                 (type (simple-array (member t nil) (1)) select)
                 (type (double-float) anrm bignum cs cscale eps r scl smlnum
                                      sn)
                 (type fixnum hswork i icond ierr itau iwrk k maxb
                                          maxwrk minwrk nout)
                 (type (simple-array character (1)) job side)
```

```
         (type (member t nil) lquery scalea wantvl wantvr wntsnb
                              wntsne wntsnn wntsnv))
(setf info 0)
(setf lquery (coerce (= lwork -1) '(member t nil)))
(setf wantvl (char-equal jobvl #\V))
(setf wantvr (char-equal jobvr #\V))
(setf wntsnn (char-equal sense #\N))
(setf wntsne (char-equal sense #\E))
(setf wntsnv (char-equal sense #\V))
(setf wntsnb (char-equal sense #\B))
(cond
  ((not
    (or (char-equal balanc #\N)
        (char-equal balanc #\S)
        (char-equal balanc #\P)
        (char-equal balanc #\B)))
   (setf info -1))
  ((and (not wantvl) (not (char-equal jobvl #\N)))
   (setf info -2))
  ((and (not wantvr) (not (char-equal jobvr #\N)))
   (setf info -3))
  ((or (not (or wntsnn wntsne wntsnb wntsnv))
       (and (or wntsne wntsnb) (not (and wantvl wantvr))))
   (setf info -4))
  ((< n 0)
   (setf info -5))
  ((< lda (max (the fixnum 1) (the fixnum n)))
   (setf info -7))
  ((or (< ldvl 1) (and wantvl (< ldvl n)))
   (setf info -11))
  ((or (< ldvr 1) (and wantvr (< ldvr n)))
   (setf info -13)))
(setf minwrk 1)
(cond
  ((and (= info 0) (or (>= lwork 1) lquery))
   (setf maxwrk
           (f2cl-lib:int-add n
                             (f2cl-lib:int-mul n
                                               (ilaenv 1 "DGEHRD" " " n
                                                1 n 0))))
   (cond
     ((and (not wantvl) (not wantvr))
      (setf minwrk
              (max (the fixnum 1)
                   (the fixnum (f2cl-lib:int-mul 2 n))))
      (if (not wntsnn)
```

```
    (setf minwrk
            (max (the fixnum minwrk)
                 (the fixnum
                      (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                                        (f2cl-lib:int-mul 6
                                                          n))))))
(setf maxb
        (max
         (the fixnum
              (ilaenv 8 "DHSEQR" "SN" n 1 n -1))
         (the fixnum 2)))
(cond
  (wntsnn
   (setf k
           (min (the fixnum maxb)
                (the fixnum n)
                (the fixnum
                     (max (the fixnum 2)
                          (the fixnum
                               (ilaenv 4 "DHSEQR" "EN" n 1 n
                                -1)))))))
  (t
   (setf k
           (min (the fixnum maxb)
                (the fixnum n)
                (the fixnum
                     (max (the fixnum 2)
                          (the fixnum
                               (ilaenv 4 "DHSEQR" "SN" n 1 n
                                -1)))))))
(setf hswork
        (max
         (the fixnum
              (f2cl-lib:int-mul k (f2cl-lib:int-add k 2)))
         (the fixnum (f2cl-lib:int-mul 2 n))))
(setf maxwrk
        (max (the fixnum maxwrk)
             (the fixnum 1)
             (the fixnum hswork)))
(if (not wntsnn)
    (setf maxwrk
            (max (the fixnum maxwrk)
                 (the fixnum
                      (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                                        (f2cl-lib:int-mul 6
                                                          n)))))))
```

```
(t
 (setf minwrk
         (max (the fixnum 1)
              (the fixnum (f2cl-lib:int-mul 3 n))))
 (if (and (not wntsnn) (not wntsne))
     (setf minwrk
             (max (the fixnum minwrk)
                  (the fixnum
                       (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                                         (f2cl-lib:int-mul 6
                                                           n))))))
 (setf maxb
         (max
          (the fixnum
               (ilaenv 8 "DHSEQR" "SN" n 1 n -1))
          (the fixnum 2)))
 (setf k
         (min (the fixnum maxb)
              (the fixnum n)
              (the fixnum
                   (max (the fixnum 2)
                        (the fixnum
                             (ilaenv 4 "DHSEQR" "EN" n 1 n -1))))))
 (setf hswork
         (max
          (the fixnum
               (f2cl-lib:int-mul k (f2cl-lib:int-add k 2)))
          (the fixnum (f2cl-lib:int-mul 2 n))))
 (setf maxwrk
         (max (the fixnum maxwrk)
              (the fixnum 1)
              (the fixnum hswork)))
 (setf maxwrk
         (max (the fixnum maxwrk)
              (the fixnum
                   (f2cl-lib:int-add n
                                     (f2cl-lib:int-mul
                                      (f2cl-lib:int-sub n 1)
                                      (ilaenv 1 "DORGHR" " " n 1 n
                                       -1))))))
 (if (and (not wntsnn) (not wntsne))
     (setf maxwrk
             (max (the fixnum maxwrk)
                  (the fixnum
                       (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                                         (f2cl-lib:int-mul 6
```

```
                                                                    n))))))
      (setf maxwrk
              (max (the fixnum maxwrk)
                   (the fixnum (f2cl-lib:int-mul 3 n))
                   (the fixnum 1)))))
   (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
           (coerce (the fixnum maxwrk) 'double-float))))
(cond
  ((and (< lwork minwrk) (not lquery))
   (setf info -21)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DGEEVX" (f2cl-lib:int-sub info))
   (go end_label))
  (lquery
   (go end_label)))
(if (= n 0) (go end_label))
(setf eps (dlamch "P"))
(setf smlnum (dlamch "S"))
(setf bignum (/ one smlnum))
(multiple-value-bind (var-0 var-1)
    (dlabad smlnum bignum)
  (declare (ignore))
  (setf smlnum var-0)
  (setf bignum var-1))
(setf smlnum (/ (f2cl-lib:fsqrt smlnum) eps))
(setf bignum (/ one smlnum))
(setf icond 0)
(setf anrm (dlange "M" n n a lda dum))
(setf scalea nil)
(cond
  ((and (> anrm zero) (< anrm smlnum))
   (setf scalea t)
   (setf cscale smlnum))
  ((> anrm bignum)
   (setf scalea t)
   (setf cscale bignum)))
(if scalea
    (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
        (dlascl "G" 0 0 anrm cscale n n a lda ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                       var-8))
      (setf ierr var-9)))
```

```
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgebal balanc n a lda ilo ihi scale ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-6))
  (setf ilo var-4)
  (setf ihi var-5)
  (setf ierr var-7))
(setf abnrm (dlange "1" n n a lda dum))
(cond
  (scalea
   (setf (f2cl-lib:fref dum (1) ((1 1))) abnrm)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
       (dlascl "G" 0 0 cscale anrm 1 1 dum 1 ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                      var-8))
     (setf ierr var-9))
   (setf abnrm (f2cl-lib:fref dum (1) ((1 1))))))
(setf itau 1)
(setf iwrk (f2cl-lib:int-add itau n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
    (dgehrd n ilo ihi a lda
     (f2cl-lib:array-slice work double-float (itau) ((1 *)))
     (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7))
  (setf ierr var-8))
(cond
  (wantvl
   (f2cl-lib:f2cl-set-string side "L" (string 1))
   (dlacpy "L" n n a lda vl ldvl)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
       (dorghr n ilo ihi vl ldvl
        (f2cl-lib:array-slice work double-float (itau) ((1 *)))
        (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7))
     (setf ierr var-8))
   (setf iwrk itau)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
          var-10 var-11 var-12 var-13)
       (dhseqr "S" "V" n ilo ihi a lda wr wi vl ldvl
        (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) info)
```

```
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                           var-8 var-9 var-10 var-11 var-12))
          (setf info var-13))
         (cond
          (wantvr
           (f2cl-lib:f2cl-set-string side "B" (string 1))
           (dlacpy "F" n n vl ldvl vr ldvr))))
        (wantvr
         (f2cl-lib:f2cl-set-string side "R" (string 1))
         (dlacpy "L" n n a lda vr ldvr)
         (multiple-value-bind
               (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
             (dorghr n ilo ihi vr ldvr
              (f2cl-lib:array-slice work double-float (itau) ((1 *)))
              (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
              (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) ierr)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7))
          (setf ierr var-8))
         (setf iwrk itau)
         (multiple-value-bind
               (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
                var-10 var-11 var-12 var-13)
             (dhseqr "S" "V" n ilo ihi a lda wr wi vr ldvr
              (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
              (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) info)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                           var-8 var-9 var-10 var-11 var-12))
          (setf info var-13)))
        (t
         (cond
          (wntsnn
           (f2cl-lib:f2cl-set-string job "E" (string 1)))
          (t
           (f2cl-lib:f2cl-set-string job "S" (string 1))))
         (setf iwrk itau)
         (multiple-value-bind
               (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
                var-10 var-11 var-12 var-13)
             (dhseqr job "N" n ilo ihi a lda wr wi vr ldvr
              (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
              (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) info)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                           var-8 var-9 var-10 var-11 var-12))
          (setf info var-13))))
    (if (> info 0) (go label50))
    (cond
```

```
  ((or wantvl wantvr)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
          var-10 var-11 var-12 var-13)
       (dtrevc side "B" select n a lda vl ldvl vr ldvr n nout
        (f2cl-lib:array-slice work double-float (iwrk) ((1 *))) ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                      var-8 var-9 var-10 var-12))
     (setf nout var-11)
     (setf ierr var-13))))
(cond
  ((not wntsnn)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
          var-10 var-11 var-12 var-13 var-14 var-15 var-16 var-17)
       (dtrsna sense "A" select n a lda vl ldvl vr ldvr rconde rcondv n
        nout (f2cl-lib:array-slice work double-float (iwrk) ((1 *))) n
        iwork icond)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                      var-8 var-9 var-10 var-11 var-12 var-14 var-15
                      var-16))
     (setf nout var-13)
     (setf icond var-17))))
(cond
  (wantvl
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
       (dgebak balanc "L" n ilo ihi scale n vl ldvl ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                      var-8))
     (setf ierr var-9))
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                 ((> i n) nil)
     (tagbody
       (cond
         ((= (f2cl-lib:fref wi (i) ((1 *))) zero)
          (setf scl
                  (/ one
                     (dnrm2 n
                      (f2cl-lib:array-slice vl
                                            double-float
                                            (1 i)
                                            ((1 ldvl) (1 *)))
                      1)))
          (dscal n scl
           (f2cl-lib:array-slice vl
```

```
                        double-float
                        (1 i)
                        ((1 ldvl) (1 *)))
  1))
 ((> (f2cl-lib:fref wi (i) ((1 *))) zero)
  (setf scl
        (/ one
           (dlapy2
            (dnrm2 n
             (f2cl-lib:array-slice vl
                                   double-float
                                   (1 i)
                                   ((1 ldvl) (1 *)))
             1)
            (dnrm2 n
             (f2cl-lib:array-slice vl
                                   double-float
                                   (1 (f2cl-lib:int-add i 1))
                                   ((1 ldvl) (1 *)))
             1))))
  (dscal n scl
   (f2cl-lib:array-slice vl
                         double-float
                         (1 i)
                         ((1 ldvl) (1 *)))
   1)
  (dscal n scl
   (f2cl-lib:array-slice vl
                         double-float
                         (1 (f2cl-lib:int-add i 1))
                         ((1 ldvl) (1 *)))
   1)
  (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                ((> k n) nil)
    (tagbody
      (setf (f2cl-lib:fref work-%data%
                           (k)
                           ((1 *))
                           work-%offset%)
            (+
             (expt
              (f2cl-lib:fref vl-%data%
                             (k i)
                             ((1 ldvl) (1 *))
                             vl-%offset%)
              2)
```

```
                                (expt
                                 (f2cl-lib:fref vl-%data%
                                                (k (f2cl-lib:int-add i 1))
                                                ((1 ldvl) (1 *))
                                                vl-%offset%)
                                 2)))))
                        (setf k (idamax n work 1))
                        (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                            (dlartg
                             (f2cl-lib:fref vl-%data%
                                            (k i)
                                            ((1 ldvl) (1 *))
                                            vl-%offset%)
                             (f2cl-lib:fref vl-%data%
                                            (k (f2cl-lib:int-add i 1))
                                            ((1 ldvl) (1 *))
                                            vl-%offset%)
                             cs sn r)
                          (declare (ignore var-0 var-1))
                          (setf cs var-2)
                          (setf sn var-3)
                          (setf r var-4))
                        (drot n
                         (f2cl-lib:array-slice vl
                                               double-float
                                               (1 i)
                                               ((1 ldvl) (1 *)))
                         1
                         (f2cl-lib:array-slice vl
                                               double-float
                                               (1 (f2cl-lib:int-add i 1))
                                               ((1 ldvl) (1 *)))
                         1 cs sn)
                        (setf (f2cl-lib:fref vl-%data%
                                             (k (f2cl-lib:int-add i 1))
                                             ((1 ldvl) (1 *))
                                             vl-%offset%)
                              zero)))))))
          (cond
            (wantvr
             (multiple-value-bind
                   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
                 (dgebak balanc "R" n ilo ihi scale n vr ldvr ierr)
               (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                                var-8))
               (setf ierr var-9))
```

```
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i n) nil)
  (tagbody
    (cond
      ((= (f2cl-lib:fref wi (i) ((1 *))) zero)
       (setf scl
              (/ one
                 (dnrm2 n
                  (f2cl-lib:array-slice vr
                                        double-float
                                        (1 i)
                                        ((1 ldvr) (1 *)))
                  1)))
       (dscal n scl
        (f2cl-lib:array-slice vr
                              double-float
                              (1 i)
                              ((1 ldvr) (1 *)))
        1))
      ((> (f2cl-lib:fref wi (i) ((1 *))) zero)
       (setf scl
              (/ one
                 (dlapy2
                  (dnrm2 n
                   (f2cl-lib:array-slice vr
                                         double-float
                                         (1 i)
                                         ((1 ldvr) (1 *)))
                   1)
                  (dnrm2 n
                   (f2cl-lib:array-slice vr
                                         double-float
                                         (1 (f2cl-lib:int-add i 1))
                                         ((1 ldvr) (1 *)))
                   1))))
       (dscal n scl
        (f2cl-lib:array-slice vr
                              double-float
                              (1 i)
                              ((1 ldvr) (1 *)))
        1)
       (dscal n scl
        (f2cl-lib:array-slice vr
                              double-float
                              (1 (f2cl-lib:int-add i 1))
                              ((1 ldvr) (1 *)))
```

```
 1)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k n) nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data%
                         (k)
                         ((1 *))
                         work-%offset%)
            (+
             (expt
              (f2cl-lib:fref vr-%data%
                             (k i)
                             ((1 ldvr) (1 *))
                             vr-%offset%)
              2)
             (expt
              (f2cl-lib:fref vr-%data%
                             (k (f2cl-lib:int-add i 1))
                             ((1 ldvr) (1 *))
                             vr-%offset%)
              2)))))
(setf k (idamax n work 1))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlartg
     (f2cl-lib:fref vr-%data%
                    (k i)
                    ((1 ldvr) (1 *))
                    vr-%offset%)
     (f2cl-lib:fref vr-%data%
                    (k (f2cl-lib:int-add i 1))
                    ((1 ldvr) (1 *))
                    vr-%offset%)
     cs sn r)
  (declare (ignore var-0 var-1))
  (setf cs var-2)
  (setf sn var-3)
  (setf r var-4))
(drot n
 (f2cl-lib:array-slice vr
                       double-float
                       (1 i)
                       ((1 ldvr) (1 *)))
 1
 (f2cl-lib:array-slice vr
                       double-float
                       (1 (f2cl-lib:int-add i 1))
```

```
                                        ((1 ldvr) (1 *)))
                    1 cs sn)
                   (setf (f2cl-lib:fref vr-%data%
                                        (k (f2cl-lib:int-add i 1))
                                        ((1 ldvr) (1 *))
                                        vr-%offset%)
                         zero)))))))
  label50
         (cond
           (scalea
            (multiple-value-bind
                  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
                (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub n info) 1
                 (f2cl-lib:array-slice wr double-float ((+ info 1)) ((1 *)))
                 (max (the fixnum (f2cl-lib:int-sub n info))
                      (the fixnum 1))
                 ierr)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                               var-8))
              (setf ierr var-9))
            (multiple-value-bind
                  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
                (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub n info) 1
                 (f2cl-lib:array-slice wi double-float ((+ info 1)) ((1 *)))
                 (max (the fixnum (f2cl-lib:int-sub n info))
                      (the fixnum 1))
                 ierr)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                               var-8))
              (setf ierr var-9))
            (cond
              ((= info 0)
               (if (and (or wntsnv wntsnb) (= icond 0))
                   (multiple-value-bind
                         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                          var-9)
                       (dlascl "G" 0 0 cscale anrm n 1 rcondv n ierr)
                     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                      var-7 var-8))
                     (setf ierr var-9))))
              (t
               (multiple-value-bind
                     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                      var-9)
                   (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub ilo 1) 1 wr n
                    ierr)
```

```
                (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                 var-7 var-8))
                (setf ierr var-9))
              (multiple-value-bind
                   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                    var-9)
                 (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub ilo 1) 1 wi n
                  ierr)
                (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                 var-7 var-8))
                (setf ierr var-9))))))
       (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                (coerce (the fixnum maxwrk) 'double-float))
end_label
       (return
        (values nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                ilo
                ihi
                nil
                abnrm
                nil
                nil
                nil
                nil
                nil
                info))))))
```

7.10 dgehd2 LAPACK

⟨*dgehd2.input*⟩≡

```
)set break resume
)sys rm -f dgehd2.output
)spool dgehd2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgehd2.help*⟩≡

```
==========================================================================
dgehd2 examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

NAME
       DGEHD2  -  a  real  general  matrix  A to upper Hessenberg form H by an
       orthogonal similarity transformation

SYNOPSIS
       SUBROUTINE DGEHD2( N, ILO, IHI, A, LDA, TAU, WORK, INFO )

           INTEGER        IHI, ILO, INFO, LDA, N

           DOUBLE         PRECISION A( LDA, * ), TAU( * ), WORK( * )

PURPOSE
       DGEHD2 reduces a real general matrix A to upper Hessenberg form H by an
       orthogonal similarity transformation:  Q' * A * Q = H .


ARGUMENTS
       N       (input) INTEGER
               The order of the matrix A.  N >= 0.

       ILO     (input) INTEGER
               IHI      (input)  INTEGER It is assumed that A is already upper
               triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI
               are  normally  set by a previous call to DGEBAL; otherwise they
               should be set to 1 and N respectively. See Further Details.

       A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
               On entry, the n by n general matrix to be  reduced.   On  exit,
               the upper triangle and the first subdiagonal of A are overwrit-
               ten with the upper Hessenberg matrix H, and the elements  below
               the  first  subdiagonal,  with  the  array  TAU,  represent the
               orthogonal matrix Q as a product of elementary reflectors.  See
               Further Details.  LDA     (input) INTEGER The leading dimension
               of the array A.  LDA >= max(1,N).

       TAU     (output) DOUBLE PRECISION array, dimension (N-1)
               The scalar factors of the elementary  reflectors  (see  Further
```

```
        Details).

WORK    (workspace) DOUBLE PRECISION array, dimension (N)

INFO    (output) INTEGER
        = 0:  successful exit.
        < 0:  if INFO = -i, the i-th argument had an illegal value.

FURTHER DETAILS
       The  matrix  Q  is  represented  as  a  product of (ihi-ilo) elementary
       reflectors

          Q = H(ilo) H(ilo+1) . . . H(ihi-1).

       Each H(i) has the form

          H(i) = I - tau * v * v'

       where tau is a real scalar, and v is a real vector with
       v(1:i) = 0, v(i+1) = 1 and v(ihi+1:n) = 0; v(i+2:ihi) is stored on exit
       in A(i+2:ihi,i), and tau in TAU(i).

       The contents of A are illustrated by the following example, with n = 7,
       ilo = 2 and ihi = 6:

       on entry,                         on exit,

       ( a   a   a   a   a   a   a )    (  a   a   h   h   h   h   a ) (     a
       a    a    a   a   a )    (      a   h   h   h   h   a ) (     a   a   a
       a   a   a )    (      h   h   h   h   h   h ) (     a   a   a    a    a
       a  )     (       v2   h   h   h   h   h ) (     a   a   a   a   a   a )
       (      v2  v3  h   h   h   h ) (     a   a    a    a    a    a  )     (
       v2    v3    v4    h    h    h  )  (                          a  )     (
       a )
```

       where a denotes an element of the original matrix A, h denotes a modified element of the upper Hessenberg matrix H, and vi denotes an element of the vector defining H(i).

⟨*LAPACK dgehd2*⟩≡

```
(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dgehd2 (n ilo ihi a lda tau work info)
    (declare (type (simple-array double-float (*)) work tau a)
             (type fixnum info lda ihi ilo n))
    (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((aii 0.0) (i 0))
        (declare (type (double-float) aii) (type fixnum i))
        (setf info 0)
        (cond
          ((< n 0)
           (setf info -1))
          ((or (< ilo 1)
               (> ilo
                  (max (the fixnum 1) (the fixnum n))))
           (setf info -2))
          ((or
            (< ihi (min (the fixnum ilo) (the fixnum n)))
            (> ihi n))
           (setf info -3))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info -5)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DGEHD2" (f2cl-lib:int-sub info))
           (go end_label)))
        (f2cl-lib:fdo (i ilo (f2cl-lib:int-add i 1))
                      ((> i (f2cl-lib:int-add ihi (f2cl-lib:int-sub 1))) nil)
          (tagbody
            (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                (dlarfg (f2cl-lib:int-sub ihi i)
                 (f2cl-lib:fref a-%data%
                                ((f2cl-lib:int-add i 1) i)
                                ((1 lda) (1 *))
                                a-%offset%)
                 (f2cl-lib:array-slice a
                                       double-float
                                       ((min (f2cl-lib:int-add i 2) n) i)
                                       ((1 lda) (1 *)))
                 1 (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
```

```
                (declare (ignore var-0 var-2 var-3))
                (setf (f2cl-lib:fref a-%data%
                                     ((f2cl-lib:int-add i 1) i)
                                     ((1 lda) (1 *))
                                     a-%offset%)
                      var-1)
                (setf (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%) var-4))
              (setf aii
                    (f2cl-lib:fref a-%data%
                                   ((f2cl-lib:int-add i 1) i)
                                   ((1 lda) (1 *))
                                   a-%offset%))
              (setf (f2cl-lib:fref a-%data%
                                   ((f2cl-lib:int-add i 1) i)
                                   ((1 lda) (1 *))
                                   a-%offset%)
                    one)
              (dlarf "Right" ihi (f2cl-lib:int-sub ihi i)
               (f2cl-lib:array-slice a double-float ((+ i 1) i) ((1 lda) (1 *)))
               1 (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
               (f2cl-lib:array-slice a
                                     double-float
                                     (1 (f2cl-lib:int-add i 1))
                                     ((1 lda) (1 *)))
               lda work)
              (dlarf "Left" (f2cl-lib:int-sub ihi i) (f2cl-lib:int-sub n i)
               (f2cl-lib:array-slice a double-float ((+ i 1) i) ((1 lda) (1 *)))
               1 (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
               (f2cl-lib:array-slice a
                                     double-float
                                     ((+ i 1) (f2cl-lib:int-add i 1))
                                     ((1 lda) (1 *)))
               lda work)
              (setf (f2cl-lib:fref a-%data%
                                   ((f2cl-lib:int-add i 1) i)
                                   ((1 lda) (1 *))
                                   a-%offset%)
                    aii)))
  end_label
         (return (values nil nil nil nil nil nil nil info))))))
```

7.11 dgehrd LAPACK

⟨*dgehrd.input*⟩≡

```
)set break resume
)sys rm -f dgehrd.output
)spool dgehrd.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgehrd.help*⟩≡

```
  ======================================================================
  dgehrd examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DGEHRD  -  Reduce a real general matrix A to upper Hessenberg form H by
         an orthogonal similarity transformation

  SYNOPSIS
         SUBROUTINE DGEHRD( N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO )

             INTEGER        IHI, ILO, INFO, LDA, LWORK, N

             DOUBLE         PRECISION A( LDA, * ), TAU( * ), WORK( * )

  PURPOSE
         DGEHRD reduces a real general matrix A to upper Hessenberg form H by an
         orthogonal similarity transformation:  Q' * A * Q = H .


  ARGUMENTS
         N       (input) INTEGER
                 The order of the matrix A.  N >= 0.

         ILO     (input) INTEGER
                 IHI      (input)  INTEGER It is assumed that A is already upper
                 triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI
                 are  normally  set by a previous call to DGEBAL; otherwise they
                 should be set to 1 and N respectively. See Further Details.

         A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
                 On entry, the N-by-N general matrix to be  reduced.   On  exit,
                 the upper triangle and the first subdiagonal of A are overwrit-
                 ten with the upper Hessenberg matrix H, and the elements  below
                 the  first  subdiagonal,  with  the  array  TAU,  represent the
                 orthogonal matrix Q as a product of elementary reflectors.  See
                 Further Details.  LDA     (input) INTEGER The leading dimension
                 of the array A.  LDA >= max(1,N).

         TAU     (output) DOUBLE PRECISION array, dimension (N-1)
                 The scalar factors of the elementary  reflectors  (see  Further
```

```
          Details).  Elements 1:ILO-1 and IHI:N-1 of TAU are set to zero.

      WORK    (workspace/output) DOUBLE PRECISION array, dimension (LWORK)
          On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

      LWORK   (input) INTEGER
          The length of the array WORK.  LWORK >= max(1,N).  For  optimum
          performance LWORK >= N*NB, where NB is the optimal blocksize.

          If  LWORK  = -1, then a workspace query is assumed; the routine
          only calculates the optimal size of  the  WORK  array,  returns
          this  value  as the first entry of the WORK array, and no error
          message related to LWORK is issued by XERBLA.

      INFO    (output) INTEGER
          = 0:  successful exit
          < 0:  if INFO = -i, the i-th argument had an illegal value.

  FURTHER DETAILS
      The matrix Q is  represented  as  a  product  of  (ihi-ilo)  elementary
      reflectors

         Q = H(ilo) H(ilo+1) . . . H(ihi-1).

      Each H(i) has the form

         H(i) = I - tau * v * v'

      where tau is a real scalar, and v is a real vector with
      v(1:i) = 0, v(i+1) = 1 and v(ihi+1:n) = 0; v(i+2:ihi) is stored on exit
      in A(i+2:ihi,i), and tau in TAU(i).

      The contents of A are illustrated by the following example, with n = 7,
      ilo = 2 and ihi = 6:

      on entry,                        on exit,

      ( a   a   a   a   a   a   a )    ( a   a   h   h   h   h   a ) (     a
      a   a   a   a   a   a )    (      a   h   h   h   h   a ) (     a    a
      a    a    a )    (      h   h   h   h   h   h ) (     a   a   a   a   a
      a )    (      v2  h   h   h   h   h ) (     a   a   a    a    a    a  )
      (       v2   v3   h    h    h    h ) (     a   a   a   a   a   a )    (
      v2   v3   v4   h    h    h  )  (                          a   )      (
      a )

      where  a denotes an element of the original matrix A, h denotes a modi-
```

```
fied element of the upper Hessenberg matrix H, and vi denotes  an  ele-
ment of the vector defining H(i).

See Quintana-Orti and Van  de  Geijn (2005).
```

⟨*LAPACK dgehrd*⟩≡

```
(let* ((nbmax 64) (ldt (+ nbmax 1)) (zero 0.0) (one 1.0))
  (declare (type (fixnum 64 64) nbmax)
           (type fixnum ldt)
           (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dgehrd (n ilo ihi a lda tau work lwork info)
    (declare (type (simple-array double-float (*)) work tau a)
             (type fixnum info lwork lda ihi ilo n))
    (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((ei 0.0) (i 0) (ib 0) (iinfo 0) (iws 0) (ldwork 0) (lwkopt 0)
             (nb 0) (nbmin 0) (nh 0) (nx 0) (lquery nil)
             (t$
              (make-array (the fixnum (reduce #'* (list ldt nbmax)))
                          :element-type 'double-float)))
        (declare (type (simple-array double-float (*)) t$)
                 (type (double-float) ei)
                 (type fixnum i ib iinfo iws ldwork lwkopt nb
                                        nbmin nh nx)
                 (type (member t nil) lquery))
        (setf info 0)
        (setf nb
                (min (the fixnum nbmax)
                     (the fixnum
                          (ilaenv 1 "DGEHRD" " " n ilo ihi -1))))
        (setf lwkopt (f2cl-lib:int-mul n nb))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                (coerce (the fixnum lwkopt) 'double-float))
        (setf lquery (coerce (= lwork -1) '(member t nil)))
        (cond
          ((< n 0)
           (setf info -1))
          ((or (< ilo 1)
               (> ilo
                  (max (the fixnum 1) (the fixnum n))))
           (setf info -2))
          ((or
            (< ihi (min (the fixnum ilo) (the fixnum n)))
            (> ihi n))
           (setf info -3))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info -5))
          ((and
```

```
     (< lwork (max (the fixnum 1) (the fixnum n)))
     (not lquery))
    (setf info -8)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DGEHRD" (f2cl-lib:int-sub info))
   (go end_label))
  (lquery
   (go end_label)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i (f2cl-lib:int-add ilo (f2cl-lib:int-sub 1))) nil)
  (tagbody
    (setf (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%) zero)))
(f2cl-lib:fdo (i
               (max (the fixnum 1)
                    (the fixnum ihi))
               (f2cl-lib:int-add i 1))
              ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
  (tagbody
    (setf (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%) zero)))
(setf nh (f2cl-lib:int-add (f2cl-lib:int-sub ihi ilo) 1))
(cond
  ((<= nh 1)
   (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
         (coerce (the fixnum 1) 'double-float))
   (go end_label)))
(setf nb
      (min (the fixnum nbmax)
           (the fixnum
                (ilaenv 1 "DGEHRD" " " n ilo ihi -1))))
(setf nbmin 2)
(setf iws 1)
(cond
  ((and (> nb 1) (< nb nh))
   (setf nx
         (max (the fixnum nb)
              (the fixnum
                   (ilaenv 3 "DGEHRD" " " n ilo ihi -1))))
   (cond
     ((< nx nh)
      (setf iws (f2cl-lib:int-mul n nb))
      (cond
        ((< lwork iws)
         (setf nbmin
```

```
                    (max (the fixnum 2)
                         (the fixnum
                              (ilaenv 2 "DGEHRD" " " n ilo ihi -1))))
              (cond
                ((>= lwork (f2cl-lib:int-mul n nbmin))
                 (setf nb (the fixnum (truncate lwork n))))
                (t
                 (setf nb 1)))))))
        (setf ldwork n)
        (cond
          ((or (< nb nbmin) (>= nb nh))
           (setf i ilo))
          (t
           (f2cl-lib:fdo (i ilo (f2cl-lib:int-add i nb))
                         ((> i
                             (f2cl-lib:int-add ihi
                                               (f2cl-lib:int-sub 1)
                                               (f2cl-lib:int-sub nx)))
                          nil)
             (tagbody
               (setf ib
                       (min (the fixnum nb)
                            (the fixnum (f2cl-lib:int-sub ihi i))))
               (dlahrd ihi i ib
                (f2cl-lib:array-slice a double-float (1 i) ((1 lda) (1 *))) lda
                (f2cl-lib:array-slice tau double-float (i) ((1 *))) t$ ldt work
                ldwork)
               (setf ei
                       (f2cl-lib:fref a-%data%
                                      ((f2cl-lib:int-add i ib)
                                       (f2cl-lib:int-sub
                                        (f2cl-lib:int-add i ib)
                                        1))
                                      ((1 lda) (1 *))
                                      a-%offset%))
               (setf (f2cl-lib:fref a-%data%
                                    ((f2cl-lib:int-add i ib)
                                     (f2cl-lib:int-sub (f2cl-lib:int-add i ib)
                                                       1))
                                    ((1 lda) (1 *))
                                    a-%offset%)
                       one)
               (dgemm "No transpose" "Transpose" ihi
                (f2cl-lib:int-add (f2cl-lib:int-sub ihi i ib) 1) ib (- one)
                work ldwork
                (f2cl-lib:array-slice a
```

```
                                    double-float
                                    ((+ i ib) i)
                                    ((1 lda) (1 *)))
                 lda one
                 (f2cl-lib:array-slice a
                                    double-float
                                    (1 (f2cl-lib:int-add i ib))
                                    ((1 lda) (1 *)))
                 lda)
                (setf (f2cl-lib:fref a-%data%
                                     ((f2cl-lib:int-add i ib)
                                      (f2cl-lib:int-sub (f2cl-lib:int-add i ib)
                                                        1))
                                     ((1 lda) (1 *))
                                     a-%offset%)
                       ei)
                (dlarfb "Left" "Transpose" "Forward" "Columnwise"
                 (f2cl-lib:int-sub ihi i)
                 (f2cl-lib:int-add (f2cl-lib:int-sub n i ib) 1) ib
                 (f2cl-lib:array-slice a
                                    double-float
                                    ((+ i 1) i)
                                    ((1 lda) (1 *)))
                 lda t$ ldt
                 (f2cl-lib:array-slice a
                                    double-float
                                    ((+ i 1) (f2cl-lib:int-add i ib))
                                    ((1 lda) (1 *)))
                 lda work ldwork)))))
        (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
            (dgehd2 n i ihi a lda tau work iinfo)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
          (setf iinfo var-7))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                (coerce (the fixnum iws) 'double-float))
  end_label
        (return (values nil nil nil nil nil nil nil nil info))))))
```

7.12 dgelq2 LAPACK

⟨*dgelq2.input*⟩≡

```
)set break resume
)sys rm -f dgelq2.output
)spool dgelq2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgelq2.help*⟩≡

```
  ==========================================================================
  dgelq2 examples
  ==========================================================================

  ==========================================================================
  Man Page Details
  ==========================================================================

  NAME
         DGELQ2 - an LQ factorization of a real m by n matrix A

  SYNOPSIS
         SUBROUTINE DGELQ2( M, N, A, LDA, TAU, WORK, INFO )

             INTEGER        INFO, LDA, M, N

             DOUBLE         PRECISION A( LDA, * ), TAU( * ), WORK( * )

  PURPOSE
         DGELQ2  computes an LQ factorization of a real m by n matrix A: A = L *
         Q.


  ARGUMENTS
         M       (input) INTEGER
                 The number of rows of the matrix A.  M >= 0.

         N       (input) INTEGER
                 The number of columns of the matrix A.  N >= 0.

         A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
                 On entry, the m by n matrix A.  On exit, the  elements  on  and
                 below the diagonal of the array contain the m by min(m,n) lower
                 trapezoidal matrix L (L is lower triangular if  m  <=  n);  the
                 elements  above the diagonal, with the array TAU, represent the
                 orthogonal matrix Q as a product of elementary reflectors  (see
                 Further  Details).   LDA     (input) INTEGER The leading dimen-
                 sion of the array A.  LDA >= max(1,M).

         TAU     (output) DOUBLE PRECISION array, dimension (min(M,N))
                 The scalar factors of the elementary  reflectors  (see  Further
                 Details).

         WORK    (workspace) DOUBLE PRECISION array, dimension (M)
```

```
        INFO    (output) INTEGER
                = 0: successful exit
                < 0: if INFO = -i, the i-th argument had an illegal value

  FURTHER DETAILS
        The matrix Q is represented as a product of elementary reflectors

           Q = H(k) . . . H(2) H(1), where k = min(m,n).

        Each H(i) has the form

           H(i) = I - tau * v * v'

        where tau is a real scalar, and v is a real vector with
        v(1:i-1)  =  0  and v(i) = 1; v(i+1:n) is stored on exit in A(i,i+1:n),
        and tau in TAU(i).
```

⟨*LAPACK dgelq2*⟩≡

```
(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dgelq2 (m n a lda tau work info)
    (declare (type (simple-array double-float (*)) work tau a)
             (type fixnum info lda n m))
    (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((aii 0.0) (i 0) (k 0))
        (declare (type (double-float) aii) (type fixnum i k))
        (setf info 0)
        (cond
          ((< m 0)
           (setf info -1))
          ((< n 0)
           (setf info -2))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info -4)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DGELQ2" (f2cl-lib:int-sub info))
           (go end_label)))
        (setf k (min (the fixnum m) (the fixnum n)))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i k) nil)
          (tagbody
            (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                (dlarfg (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
                 (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                 (f2cl-lib:array-slice a
                                       double-float
                                       (i
                                        (min
                                         (the fixnum
                                              (f2cl-lib:int-add i 1))
                                         (the fixnum n)))
                                       ((1 lda) (1 *)))
                   lda (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
              (declare (ignore var-0 var-2 var-3))
              (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                    var-1)
              (setf (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%) var-4))
```

```
            (cond
              ((< i m)
               (setf aii
                       (f2cl-lib:fref a-%data%
                                      (i i)
                                      ((1 lda) (1 *))
                                      a-%offset%))
               (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                       one)
               (dlarf "Right" (f2cl-lib:int-sub m i)
                (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
                (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
                (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
                (f2cl-lib:array-slice a
                                      double-float
                                      ((+ i 1) i)
                                      ((1 lda) (1 *)))
                lda work)
               (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                       aii)))))
  end_label
         (return (values nil nil nil nil nil nil info))))))
```

7.13 dgelqf LAPACK

⟨*dgelqf.input*⟩≡

```
)set break resume
)sys rm -f dgelqf.output
)spool dgelqf.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgelqf.help*⟩≡

```
  ==========================================================================
  dgelqf examples
  ==========================================================================

  ==========================================================================
  Man Page Details
  ==========================================================================

  NAME
         DGELQF - an LQ factorization of a real M-by-N matrix A

  SYNOPSIS
         SUBROUTINE DGELQF( M, N, A, LDA, TAU, WORK, LWORK, INFO )

             INTEGER        INFO, LDA, LWORK, M, N

             DOUBLE         PRECISION A( LDA, * ), TAU( * ), WORK( * )

  PURPOSE
         DGELQF  computes an LQ factorization of a real M-by-N matrix A: A = L *
         Q.


  ARGUMENTS
         M       (input) INTEGER
                 The number of rows of the matrix A.  M >= 0.

         N       (input) INTEGER
                 The number of columns of the matrix A.  N >= 0.

         A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
                 On entry, the M-by-N matrix A.  On exit, the  elements  on  and
                 below the diagonal of the array contain the m-by-min(m,n) lower
                 trapezoidal matrix L (L is lower triangular if  m  <=  n);  the
                 elements  above the diagonal, with the array TAU, represent the
                 orthogonal matrix Q as a product of elementary reflectors  (see
                 Further  Details).   LDA     (input) INTEGER The leading dimen-
                 sion of the array A.  LDA >= max(1,M).

         TAU     (output) DOUBLE PRECISION array, dimension (min(M,N))
                 The scalar factors of the elementary  reflectors  (see  Further
                 Details).

         WORK       (workspace/output)   DOUBLE   PRECISION   array,   dimension
         (MAX(1,LWORK))
```

```
          On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

      LWORK   (input) INTEGER
          The dimension of the array WORK.  LWORK >= max(1,M).  For opti-
          mum performance LWORK >= M*NB, where NB is the  optimal  block-
          size.

          If  LWORK  = -1, then a workspace query is assumed; the routine
          only calculates the optimal size of  the  WORK  array,  returns
          this  value  as the first entry of the WORK array, and no error
          message related to LWORK is issued by XERBLA.

      INFO    (output) INTEGER
          = 0:  successful exit
          < 0:  if INFO = -i, the i-th argument had an illegal value

FURTHER DETAILS
      The matrix Q is represented as a product of elementary reflectors

         Q = H(k) . . . H(2) H(1), where k = min(m,n).

      Each H(i) has the form

         H(i) = I - tau * v * v'

      where tau is a real scalar, and v is a real vector with
      v(1:i-1) = 0 and v(i) = 1; v(i+1:n) is stored on  exit  in  A(i,i+1:n),
      and tau in TAU(i).
```

⟨*LAPACK dgelqf*⟩≡

```
(defun dgelqf (m n a lda tau work lwork info)
  (declare (type (simple-array double-float (*)) work tau a)
           (type fixnum info lwork lda n m))
  (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (work double-float work-%data% work-%offset%))
    (prog ((i 0) (ib 0) (iinfo 0) (iws 0) (k 0) (ldwork 0) (lwkopt 0) (nb 0)
           (nbmin 0) (nx 0) (lquery nil))
      (declare (type (member t nil) lquery)
               (type fixnum nx nbmin nb lwkopt ldwork k iws iinfo
                                       ib i))
      (setf info 0)
      (setf nb (ilaenv 1 "DGELQF" " " m n -1 -1))
      (setf lwkopt (f2cl-lib:int-mul m nb))
      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
              (coerce (the fixnum lwkopt) 'double-float))
      (setf lquery (coerce (= lwork -1) '(member t nil)))
      (cond
        ((< m 0)
         (setf info -1))
        ((< n 0)
         (setf info -2))
        ((< lda (max (the fixnum 1) (the fixnum m)))
         (setf info -4))
        ((and
          (< lwork (max (the fixnum 1) (the fixnum m)))
          (not lquery))
         (setf info -7)))
      (cond
        ((/= info 0)
         (error
          " ** On entry to ~a parameter number ~a had an illegal value~%"
          "DGELQF" (f2cl-lib:int-sub info))
         (go end_label))
        (lquery
         (go end_label)))
      (setf k (min (the fixnum m) (the fixnum n)))
      (cond
        ((= k 0)
         (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                 (coerce (the fixnum 1) 'double-float))
         (go end_label)))
      (setf nbmin 2)
      (setf nx 0)
```

```
(setf iws m)
(cond
  ((and (> nb 1) (< nb k))
   (setf nx
         (max (the fixnum 0)
              (the fixnum
                   (ilaenv 3 "DGELQF" " " m n -1 -1))))
   (cond
     ((< nx k)
      (setf ldwork m)
      (setf iws (f2cl-lib:int-mul ldwork nb))
      (cond
        ((< lwork iws)
         (setf nb (the fixnum (truncate lwork ldwork)))
         (setf nbmin
                 (max (the fixnum 2)
                      (the fixnum
                           (ilaenv 2 "DGELQF" " " m n -1 -1)))))))))
(cond
  ((and (>= nb nbmin) (< nb k) (< nx k))
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i nb))
                 ((> i (f2cl-lib:int-add k (f2cl-lib:int-sub nx))) nil)
     (tagbody
       (setf ib
             (min
              (the fixnum
                   (f2cl-lib:int-add (f2cl-lib:int-sub k i) 1))
              (the fixnum nb)))
       (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
           (dgelq2 ib (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
            (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
            lda (f2cl-lib:array-slice tau double-float (i) ((1 *))) work
            iinfo)
         (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5))
         (setf iinfo var-6))
       (cond
         ((<= (f2cl-lib:int-add i ib) m)
          (dlarft "Forward" "Rowwise"
           (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) ib
           (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
           lda (f2cl-lib:array-slice tau double-float (i) ((1 *))) work
           ldwork)
          (dlarfb "Right" "No transpose" "Forward" "Rowwise"
           (f2cl-lib:int-add (f2cl-lib:int-sub m i ib) 1)
           (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) ib
           (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
```

```
                  lda work ldwork
                  (f2cl-lib:array-slice a
                                        double-float
                                        ((+ i ib) i)
                                        ((1 lda) (1 *)))
                  lda
                  (f2cl-lib:array-slice work double-float ((+ ib 1)) ((1 *)))
                  ldwork))))))
          (t
           (setf i 1)))
        (if (<= i k)
            (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
                (dgelq2 (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
                 (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
                 (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
                 (f2cl-lib:array-slice tau double-float (i) ((1 *))) work iinfo)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5))
              (setf iinfo var-6)))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
              (coerce (the fixnum iws) 'double-float))
   end_label
        (return (values nil nil nil nil nil nil nil info)))))
```

7.14 dgeqr2 LAPACK

⟨*dgeqr2.input*⟩≡

```
)set break resume
)sys rm -f dgeqr2.output
)spool dgeqr2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgeqr2.help*⟩≡

```
=====================================================================
dgeqr2 examples
=====================================================================

=====================================================================
Man Page Details
=====================================================================

NAME
       DGEQR2 - a QR factorization of a real m by n matrix A

SYNOPSIS
       SUBROUTINE DGEQR2( M, N, A, LDA, TAU, WORK, INFO )

           INTEGER        INFO, LDA, M, N

           DOUBLE         PRECISION A( LDA, * ), TAU( * ), WORK( * )

PURPOSE
       DGEQR2  computes  a QR factorization of a real m by n matrix A: A = Q *
       R.


ARGUMENTS
       M       (input) INTEGER
               The number of rows of the matrix A.  M >= 0.

       N       (input) INTEGER
               The number of columns of the matrix A.  N >= 0.

       A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
               On entry, the m by n matrix A.  On exit, the  elements  on  and
               above the diagonal of the array contain the min(m,n) by n upper
               trapezoidal matrix R (R is upper triangular if  m  >=  n);  the
               elements  below the diagonal, with the array TAU, represent the
               orthogonal matrix Q as a product of elementary reflectors  (see
               Further  Details).   LDA     (input) INTEGER The leading dimen-
               sion of the array A.  LDA >= max(1,M).

       TAU     (output) DOUBLE PRECISION array, dimension (min(M,N))
               The scalar factors of the elementary  reflectors  (see  Further
               Details).

       WORK    (workspace) DOUBLE PRECISION array, dimension (N)
```

```
        INFO    (output) INTEGER
                = 0: successful exit
                < 0: if INFO = -i, the i-th argument had an illegal value

  FURTHER DETAILS
        The matrix Q is represented as a product of elementary reflectors

           Q = H(1) H(2) . . . H(k), where k = min(m,n).

        Each H(i) has the form

           H(i) = I - tau * v * v'

        where tau is a real scalar, and v is a real vector with
        v(1:i-1)  =  0  and v(i) = 1; v(i+1:m) is stored on exit in A(i+1:m,i),
        and tau in TAU(i).
```

⟨*LAPACK dgeqr2*⟩≡

```
(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dgeqr2 (m n a lda tau work info)
    (declare (type (simple-array double-float (*)) work tau a)
             (type fixnum info lda n m))
    (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((aii 0.0) (i 0) (k 0))
        (declare (type (double-float) aii) (type fixnum i k))
        (setf info 0)
        (cond
          ((< m 0)
           (setf info -1))
          ((< n 0)
           (setf info -2))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info -4)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DGEQR2" (f2cl-lib:int-sub info))
           (go end_label)))
        (setf k (min (the fixnum m) (the fixnum n)))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i k) nil)
          (tagbody
            (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                (dlarfg (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
                 (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                 (f2cl-lib:array-slice a
                                       double-float
                                       ((min (f2cl-lib:int-add i 1) m) i)
                                       ((1 lda) (1 *)))
                 1 (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
              (declare (ignore var-0 var-2 var-3))
              (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                      var-1)
              (setf (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%) var-4))
            (cond
              ((< i n)
               (setf aii
                       (f2cl-lib:fref a-%data%
```

```
                                    (i i)
                                    ((1 lda) (1 *))
                                    a-%offset%))
                  (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                        one)
                  (dlarf "Left" (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
                   (f2cl-lib:int-sub n i)
                   (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) 1
                   (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
                   (f2cl-lib:array-slice a
                                    double-float
                                    (i (f2cl-lib:int-add i 1))
                                    ((1 lda) (1 *)))
                   lda work)
                  (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                        aii)))))
  end_label
        (return (values nil nil nil nil nil nil info))))))
```

7.15 dgeqrf LAPACK

⟨*dgeqrf.input*⟩≡

```
  )set break resume
  )sys rm -f dgeqrf.output
  )spool dgeqrf.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dgeqrf.help*⟩≡

```
=========================================================================
dgeqrf examples
=========================================================================

=========================================================================
Man Page Details
=========================================================================

NAME
       DGEQRF - a QR factorization of a real M-by-N matrix A

SYNOPSIS
       SUBROUTINE DGEQRF( M, N, A, LDA, TAU, WORK, LWORK, INFO )

           INTEGER        INFO, LDA, LWORK, M, N

           DOUBLE         PRECISION A( LDA, * ), TAU( * ), WORK( * )

PURPOSE
       DGEQRF  computes  a QR factorization of a real M-by-N matrix A: A = Q *
       R.


ARGUMENTS
       M       (input) INTEGER
               The number of rows of the matrix A.  M >= 0.

       N       (input) INTEGER
               The number of columns of the matrix A.  N >= 0.

       A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
               On entry, the M-by-N matrix A.  On exit, the  elements  on  and
               above the diagonal of the array contain the min(M,N)-by-N upper
               trapezoidal matrix R (R is upper triangular if  m  >=  n);  the
               elements  below the diagonal, with the array TAU, represent the
               orthogonal matrix Q as a product of min(m,n) elementary reflec-
               tors (see Further Details).

       LDA     (input) INTEGER
               The leading dimension of the array A.  LDA >= max(1,M).

       TAU     (output) DOUBLE PRECISION array, dimension (min(M,N))
               The  scalar  factors  of the elementary reflectors (see Further
               Details).
```

```
        WORK      (workspace/output)  DOUBLE  PRECISION  array,  dimension
        (MAX(1,LWORK))
                On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

        LWORK   (input) INTEGER
                The dimension of the array WORK.  LWORK >= max(1,N).  For opti-
                mum  performance  LWORK >= N*NB, where NB is the optimal block-
                size.

                If LWORK = -1, then a workspace query is assumed;  the  routine
                only  calculates  the  optimal  size of the WORK array, returns
                this value as the first entry of the WORK array, and  no  error
                message related to LWORK is issued by XERBLA.

        INFO    (output) INTEGER
                = 0:  successful exit
                < 0:  if INFO = -i, the i-th argument had an illegal value

 FURTHER DETAILS
        The matrix Q is represented as a product of elementary reflectors

           Q = H(1) H(2) . . . H(k), where k = min(m,n).

        Each H(i) has the form

           H(i) = I - tau * v * v'

        where tau is a real scalar, and v is a real vector with
        v(1:i-1)  =  0  and v(i) = 1; v(i+1:m) is stored on exit in A(i+1:m,i),
        and tau in TAU(i).
```

⟨*LAPACK dgeqrf*⟩≡

```
(defun dgeqrf (m n a lda tau work lwork info)
  (declare (type (simple-array double-float (*)) work tau a)
           (type fixnum info lwork lda n m))
  (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (work double-float work-%data% work-%offset%))
    (prog ((i 0) (ib 0) (iinfo 0) (iws 0) (k 0) (ldwork 0) (lwkopt 0) (nb 0)
           (nbmin 0) (nx 0) (lquery nil))
      (declare (type (member t nil) lquery)
               (type fixnum nx nbmin nb lwkopt ldwork k iws iinfo
                                       ib i))
      (setf info 0)
      (setf nb (ilaenv 1 "DGEQRF" " " m n -1 -1))
      (setf lwkopt (f2cl-lib:int-mul n nb))
      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
              (coerce (the fixnum lwkopt) 'double-float))
      (setf lquery (coerce (= lwork -1) '(member t nil)))
      (cond
        ((< m 0)
         (setf info -1))
        ((< n 0)
         (setf info -2))
        ((< lda (max (the fixnum 1) (the fixnum m)))
         (setf info -4))
        ((and
          (< lwork (max (the fixnum 1) (the fixnum n)))
          (not lquery))
         (setf info -7)))
      (cond
        ((/= info 0)
         (error
          " ** On entry to ~a parameter number ~a had an illegal value~%"
          "DGEQRF" (f2cl-lib:int-sub info))
         (go end_label))
        (lquery
         (go end_label)))
      (setf k (min (the fixnum m) (the fixnum n)))
      (cond
        ((= k 0)
         (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                 (coerce (the fixnum 1) 'double-float))
         (go end_label)))
      (setf nbmin 2)
      (setf nx 0)
```

```
(setf iws n)
(cond
  ((and (> nb 1) (< nb k))
   (setf nx
         (max (the fixnum 0)
              (the fixnum
                   (ilaenv 3 "DGEQRF" " " m n -1 -1))))
   (cond
     ((< nx k)
      (setf ldwork n)
      (setf iws (f2cl-lib:int-mul ldwork nb))
      (cond
        ((< lwork iws)
         (setf nb (the fixnum (truncate lwork ldwork)))
         (setf nbmin
                 (max (the fixnum 2)
                      (the fixnum
                           (ilaenv 2 "DGEQRF" " " m n -1 -1)))))))))))
(cond
  ((and (>= nb nbmin) (< nb k) (< nx k))
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i nb))
                 ((> i (f2cl-lib:int-add k (f2cl-lib:int-sub nx))) nil)
     (tagbody
       (setf ib
             (min
              (the fixnum
                   (f2cl-lib:int-add (f2cl-lib:int-sub k i) 1))
              (the fixnum nb)))
       (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
           (dgeqr2 (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1) ib
            (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
            lda (f2cl-lib:array-slice tau double-float (i) ((1 *))) work
            iinfo)
         (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5))
         (setf iinfo var-6))
       (cond
         ((<= (f2cl-lib:int-add i ib) n)
          (dlarft "Forward" "Columnwise"
           (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1) ib
           (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
           lda (f2cl-lib:array-slice tau double-float (i) ((1 *))) work
           ldwork)
          (dlarfb "Left" "Transpose" "Forward" "Columnwise"
           (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
           (f2cl-lib:int-add (f2cl-lib:int-sub n i ib) 1) ib
           (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
```

```
                lda work ldwork
                (f2cl-lib:array-slice a
                                      double-float
                                      (i (f2cl-lib:int-add i ib))
                                      ((1 lda) (1 *)))
                lda
                (f2cl-lib:array-slice work double-float ((+ ib 1)) ((1 *)))
                ldwork))))))
          (t
           (setf i 1)))
        (if (<= i k)
            (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
                (dgeqr2 (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
                 (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
                 (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
                 (f2cl-lib:array-slice tau double-float (i) ((1 *))) work iinfo)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5))
              (setf iinfo var-6)))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
              (coerce (the fixnum iws) 'double-float))
   end_label
        (return (values nil nil nil nil nil nil nil info)))))
```

7.16 dgesdd LAPACK

⟨*dgesdd.input*⟩≡

```
)set break resume
)sys rm -f dgesdd.output
)spool dgesdd.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgesdd.help*⟩≡

```
  ==========================================================================
  dgesdd examples
  ==========================================================================

  ==========================================================================
  Man Page Details
  ==========================================================================

  NAME
         DGESDD - the singular value decomposition (SVD) of a real M-by-N matrix
         A, optionally computing the left and right singular vectors

  SYNOPSIS
         SUBROUTINE DGESDD( JOBZ, M, N, A, LDA,  S,  U,  LDU,  VT,  LDVT,  WORK,
                            LWORK, IWORK, INFO )

             CHARACTER      JOBZ

             INTEGER        INFO, LDA, LDU, LDVT, LWORK, M, N

             INTEGER        IWORK( * )

             DOUBLE         PRECISION  A(  LDA,  *  ),  S( * ), U( LDU, * ), VT(
                            LDVT, * ), WORK( * )

  PURPOSE
         DGESDD computes the singular value decomposition (SVD) of a real M-by-N
         matrix A, optionally computing the left and right singular vectors.  If
         singular vectors are desired, it uses a divide-and-conquer algorithm.

         The SVD is written

              A = U * SIGMA * transpose(V)

         where SIGMA is an M-by-N matrix which is zero except for  its  min(m,n)
         diagonal elements, U is an M-by-M orthogonal matrix, and V is an N-by-N
         orthogonal matrix.  The diagonal elements of  SIGMA  are  the  singular
         values  of  A;  they  are  real  and  non-negative, and are returned in
         descending order.  The first min(m,n) columns of U and V are  the  left
         and right singular vectors of A.

         Note that the routine returns VT = V**T, not V.

         The  divide  and  conquer  algorithm  makes very mild assumptions about
         floating point arithmetic. It will work on machines with a guard  digit
```

in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

ARGUMENTS

JOBZ (input) CHARACTER*1
Specifies options for computing all or part of the matrix U:
= 'A': all M columns of U and all N rows of V**T are returned in the arrays U and VT; = 'S': the first min(M,N) columns of U and the first min(M,N) rows of V**T are returned in the arrays U and VT; = 'O': If M >= N, the first N columns of U are overwritten on the array A and all rows of V**T are returned in the array VT; otherwise, all columns of U are returned in the array U and the first M rows of V**T are overwritten in the array A; = 'N': no columns of U or rows of V**T are computed.

M (input) INTEGER
The number of rows of the input matrix A. M >= 0.

N (input) INTEGER
The number of columns of the input matrix A. N >= 0.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the M-by-N matrix A. On exit, if JOBZ = 'O', A is overwritten with the first N columns of U (the left singular vectors, stored columnwise) if M >= N; A is overwritten with the first M rows of V**T (the right singular vectors, stored rowwise) otherwise. if JOBZ .ne. 'O', the contents of A are destroyed.

LDA (input) INTEGER
The leading dimension of the array A. LDA >= max(1,M).

S (output) DOUBLE PRECISION array, dimension (min(M,N))
The singular values of A, sorted so that S(i) >= S(i+1).

U (output) DOUBLE PRECISION array, dimension (LDU,UCOL)
UCOL = M if JOBZ = 'A' or JOBZ = 'O' and M < N; UCOL = min(M,N) if JOBZ = 'S'. If JOBZ = 'A' or JOBZ = 'O' and M < N, U contains the M-by-M orthogonal matrix U; if JOBZ = 'S', U contains the first min(M,N) columns of U (the left singular vectors, stored columnwise); if JOBZ = 'O' and M >= N, or JOBZ = 'N', U is not referenced.

```
LDU     (input) INTEGER
        The  leading dimension of the array U.  LDU >= 1; if JOBZ = 'S'
        or 'A' or JOBZ = 'O' and M < N, LDU >= M.

VT      (output) DOUBLE PRECISION array, dimension (LDVT,N)
        If JOBZ = 'A' or JOBZ = 'O' and M >= N, VT contains the  N-by-N
        orthogonal  matrix  V**T;  if JOBZ = 'S', VT contains the first
        min(M,N) rows of V**T (the right singular vectors, stored  row-
        wise); if JOBZ = 'O' and M < N, or JOBZ = 'N', VT is not refer-
        enced.

LDVT    (input) INTEGER
        The leading dimension of the array VT.  LDVT >= 1;  if  JOBZ  =
        'A' or JOBZ = 'O' and M >= N, LDVT >= N; if JOBZ = 'S', LDVT >=
        min(M,N).

WORK      (workspace/output)   DOUBLE   PRECISION   array,    dimension
(MAX(1,LWORK))
        On exit, if INFO = 0, WORK(1) returns the optimal LWORK;

LWORK   (input) INTEGER
        The dimension of the array WORK. LWORK >= 1.  If  JOBZ  =  'N',
        LWORK >= 3*min(M,N) + max(max(M,N),7*min(M,N)).  If JOBZ = 'O',
        LWORK            >=            3*min(M,N)*min(M,N)            +
        max(max(M,N),5*min(M,N)*min(M,N)+4*min(M,N)).  If JOBZ = 'S' or
        'A'        LWORK        >=        3*min(M,N)*min(M,N)         +
        max(max(M,N),4*min(M,N)*min(M,N)+4*min(M,N)).  For good perfor-
        mance, LWORK should generally be larger.  If  LWORK  =  -1  but
        other  input  arguments  are legal, WORK(1) returns the optimal
        LWORK.

IWORK   (workspace) INTEGER array, dimension (8*min(M,N))

INFO    (output) INTEGER
        = 0:  successful exit.
        < 0:  if INFO = -i, the i-th argument had an illegal value.
        > 0:  DBDSDC did not converge, updating process failed.
```

⟨*LAPACK dgesdd*⟩≡

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dgesdd (jobz m n a lda s u ldu vt ldvt work lwork iwork info)
    (declare (type (simple-array fixnum (*)) iwork)
             (type (simple-array double-float (*)) work vt u s a)
             (type fixnum info lwork ldvt ldu lda n m)
             (type character jobz))
    (f2cl-lib:with-multi-array-data
        ((jobz character jobz-%data% jobz-%offset%)
         (a double-float a-%data% a-%offset%)
         (s double-float s-%data% s-%offset%)
         (u double-float u-%data% u-%offset%)
         (vt double-float vt-%data% vt-%offset%)
         (work double-float work-%data% work-%offset%)
         (iwork fixnum iwork-%data% iwork-%offset%))
      (prog ((dum (make-array 1 :element-type 'double-float))
             (idum (make-array 1 :element-type 'fixnum)) (anrm 0.0)
             (bignum 0.0) (eps 0.0) (smlnum 0.0) (bdspac 0) (blk 0) (chunk 0)
             (i 0) (ie 0) (ierr 0) (il 0) (ir 0) (iscl 0) (itau 0) (itaup 0)
             (itauq 0) (iu 0) (ivt 0) (ldwkvt 0) (ldwrkl 0) (ldwrkr 0)
             (ldwrku 0) (maxwrk 0) (minmn 0) (minwrk 0) (mnthr 0) (nwork 0)
             (wrkbl 0) (lquery nil) (wntqa nil) (wntqas nil) (wntqn nil)
             (wntqo nil) (wntqs nil))
        (declare (type (simple-array double-float (1)) dum)
                 (type (simple-array fixnum (1)) idum)
                 (type (double-float) anrm bignum eps smlnum)
                 (type fixnum bdspac blk chunk i ie ierr il ir
                                        iscl itau itaup itauq iu ivt ldwkvt
                                        ldwrkl ldwrkr ldwrku maxwrk minmn
                                        minwrk mnthr nwork wrkbl)
                 (type (member t nil) lquery wntqa wntqas wntqn wntqo wntqs))
        (setf info 0)
        (setf minmn (min (the fixnum m) (the fixnum n)))
        (setf mnthr (f2cl-lib:int (/ (* minmn 11.0) 6.0)))
        (setf wntqa (char-equal jobz #\A))
        (setf wntqs (char-equal jobz #\S))
        (setf wntqas (or wntqa wntqs))
        (setf wntqo (char-equal jobz #\O))
        (setf wntqn (char-equal jobz #\N))
        (setf minwrk 1)
        (setf maxwrk 1)
        (setf lquery (coerce (= lwork -1) '(member t nil)))
        (cond
          ((not (or wntqa wntqs wntqo wntqn))
```

```
         (setf info -1))
        ((< m 0)
         (setf info -2))
        ((< n 0)
         (setf info -3))
        ((< lda (max (the fixnum 1) (the fixnum m)))
         (setf info -5))
        ((or (< ldu 1) (and wntqas (< ldu m)) (and wntqo (< m n) (< ldu m)))
         (setf info -8))
        ((or (< ldvt 1)
             (and wntqa (< ldvt n))
             (and wntqs (< ldvt minmn))
             (and wntqo (>= m n) (< ldvt n)))
         (setf info -10)))
      (cond
        ((and (= info 0) (> m 0) (> n 0))
         (cond
           ((>= m n)
            (cond
              (wntqn
               (setf bdspac (f2cl-lib:int-mul 7 n)))
              (t
               (setf bdspac
                       (f2cl-lib:int-add (f2cl-lib:int-mul 3 n n)
                                         (f2cl-lib:int-mul 4 n)))))
            (cond
              ((>= m mnthr)
               (cond
                 (wntqn
                  (setf wrkbl
                          (f2cl-lib:int-add n
                                            (f2cl-lib:int-mul n
                                                              (ilaenv 1
                                                               "DGEQRF" " " m
                                                               n -1 -1))))
                  (setf wrkbl
                          (max (the fixnum wrkbl)
                               (the fixnum
                                    (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                      (f2cl-lib:int-mul 2
                                                                        n
                                                                        (ilaenv
                                                                         1
                                                                         "DGEBRD"
                                                                         " "
                                                                         n n
```

```
                                                                  -1
                                                                  -1))))))
 (setf maxwrk
         (max (the fixnum wrkbl)
              (the fixnum
                   (f2cl-lib:int-add bdspac n))))
 (setf minwrk (f2cl-lib:int-add bdspac n)))
(wntqo
 (setf wrkbl
         (f2cl-lib:int-add n
                           (f2cl-lib:int-mul n
                                             (ilaenv 1
                                              "DGEQRF" " " m
                                              n -1 -1))))
 (setf wrkbl
         (max (the fixnum wrkbl)
              (the fixnum
                   (f2cl-lib:int-add n
                                     (f2cl-lib:int-mul n
                                                       (ilaenv
                                                        1
                                                        "DORGQR"
                                                        " "
                                                        m n
                                                        n
                                                        -1))))))
 (setf wrkbl
         (max (the fixnum wrkbl)
              (the fixnum
                   (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                     (f2cl-lib:int-mul 2
                                                       n
                                                       (ilaenv
                                                        1
                                                        "DGEBRD"
                                                        " "
                                                        n n
                                                        -1
                                                        -1))))))
 (setf wrkbl
         (max (the fixnum wrkbl)
              (the fixnum
                   (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                     (f2cl-lib:int-mul n
                                                       (ilaenv
                                                        1
```

```
                                                                  "DORMBR"
                                                                  "QLN"
                                                                  n n
                                                                  n
                                                                  -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                  (f2cl-lib:int-mul n
                                                                    (ilaenv
                                                                     1
                                                                     "DORMBR"
                                                                     "PRT"
                                                                     n n
                                                                     n
                                                                     -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add bdspac
                                                  (f2cl-lib:int-mul 3
                                                                    n)))))
                (setf maxwrk
                      (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul 2 n n)))
                (setf minwrk
                      (f2cl-lib:int-add bdspac
                                        (f2cl-lib:int-mul 2 n n)
                                        (f2cl-lib:int-mul 3 n))))
               (wntqs
                (setf wrkbl
                      (f2cl-lib:int-add n
                                        (f2cl-lib:int-mul n
                                                          (ilaenv 1
                                                           "DGEQRF" " " m
                                                           n -1 -1))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add n
                                                  (f2cl-lib:int-mul n
                                                                    (ilaenv
                                                                     1
                                                                     "DORGQR"
                                                                     " "
                                                                     m n
```

```
                                                                n
                                                                -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                  (f2cl-lib:int-mul 2
                                                                    n
                                                                    (ilaenv
                                                                     1
                                                                     "DGEBRD"
                                                                     " "
                                                                     n n
                                                                     -1
                                                                     -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                  (f2cl-lib:int-mul n
                                                                    (ilaenv
                                                                     1
                                                                     "DORMBR"
                                                                     "QLN"
                                                                     n n
                                                                     n
                                                                     -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                  (f2cl-lib:int-mul n
                                                                    (ilaenv
                                                                     1
                                                                     "DORMBR"
                                                                     "PRT"
                                                                     n n
                                                                     n
                                                                     -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add bdspac
                                                  (f2cl-lib:int-mul 3
                                                                    n)))))
                (setf maxwrk
```

```
          (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul n n)))
 (setf minwrk
          (f2cl-lib:int-add bdspac
                            (f2cl-lib:int-mul n n)
                            (f2cl-lib:int-mul 3 n))))
(wntqa
 (setf wrkbl
          (f2cl-lib:int-add n
                            (f2cl-lib:int-mul n
                                              (ilaenv 1
                                               "DGEQRF" " " m
                                               n -1 -1))))
 (setf wrkbl
          (max (the fixnum wrkbl)
               (the fixnum
                    (f2cl-lib:int-add n
                                      (f2cl-lib:int-mul m
                                                        (ilaenv
                                                         1
                                                         "DORGQR"
                                                         " "
                                                         m m
                                                         n
                                                         -1))))))
 (setf wrkbl
          (max (the fixnum wrkbl)
               (the fixnum
                    (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                      (f2cl-lib:int-mul 2
                                                        n
                                                        (ilaenv
                                                         1
                                                         "DGEBRD"
                                                         " "
                                                         n n
                                                         -1
                                                         -1))))))
 (setf wrkbl
          (max (the fixnum wrkbl)
               (the fixnum
                    (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                      (f2cl-lib:int-mul n
                                                        (ilaenv
                                                         1
                                                         "DORMBR"
                                                         "QLN"
```

```
                                                                  n n
                                                                  n
                                                                  -1))))))
            (setf wrkbl
                  (max (the fixnum wrkbl)
                       (the fixnum
                            (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                              (f2cl-lib:int-mul n
                                                                (ilaenv
                                                                 1
                                                                 "DORMBR"
                                                                 "PRT"
                                                                 n n
                                                                 n
                                                                 -1))))))
            (setf wrkbl
                  (max (the fixnum wrkbl)
                       (the fixnum
                            (f2cl-lib:int-add bdspac
                                              (f2cl-lib:int-mul 3
                                                                n)))))
            (setf maxwrk
                  (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul n n)))
            (setf minwrk
                  (f2cl-lib:int-add bdspac
                                    (f2cl-lib:int-mul n n)
                                    (f2cl-lib:int-mul 3 n))))))
          (t
           (setf wrkbl
                 (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                   (f2cl-lib:int-mul
                                    (f2cl-lib:int-add m n)
                                    (ilaenv 1 "DGEBRD" " " m n -1 -1))))
           (cond
             (wntqn
              (setf maxwrk
                    (max (the fixnum wrkbl)
                         (the fixnum
                              (f2cl-lib:int-add bdspac
                                                (f2cl-lib:int-mul 3
                                                                  n)))))
              (setf minwrk
                    (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                      (max (the fixnum m)
                                           (the fixnum
                                                bdspac)))))
```

```
(wntqo
 (setf wrkbl
       (max (the fixnum wrkbl)
            (the fixnum
                 (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                   (f2cl-lib:int-mul n
                                                     (ilaenv
                                                      1
                                                      "DORMBR"
                                                      "QLN"
                                                      m n
                                                      n
                                                      -1))))))
 (setf wrkbl
       (max (the fixnum wrkbl)
            (the fixnum
                 (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                   (f2cl-lib:int-mul n
                                                     (ilaenv
                                                      1
                                                      "DORMBR"
                                                      "PRT"
                                                      n n
                                                      n
                                                      -1))))))
 (setf wrkbl
       (max (the fixnum wrkbl)
            (the fixnum
                 (f2cl-lib:int-add bdspac
                                   (f2cl-lib:int-mul 3
                                                     n)))))
 (setf maxwrk
       (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul m n)))
 (setf minwrk
       (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                         (max (the fixnum m)
                              (the fixnum
                                   (f2cl-lib:int-add
                                    (f2cl-lib:int-mul n n)
                                    bdspac))))))
(wntqs
 (setf wrkbl
       (max (the fixnum wrkbl)
            (the fixnum
                 (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                   (f2cl-lib:int-mul n
```

```
                                                                  (ilaenv
                                                                   1
                                                                   "DORMBR"
                                                                   "QLN"
                                                                   m n
                                                                   n
                                                                   -1))))))
               (setf wrkbl
                     (max (the fixnum wrkbl)
                          (the fixnum
                               (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                 (f2cl-lib:int-mul n
                                                                  (ilaenv
                                                                   1
                                                                   "DORMBR"
                                                                   "PRT"
                                                                   n n
                                                                   n
                                                                   -1))))))
               (setf maxwrk
                     (max (the fixnum wrkbl)
                          (the fixnum
                               (f2cl-lib:int-add bdspac
                                                 (f2cl-lib:int-mul 3
                                                                  n)))))
               (setf minwrk
                     (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                       (max (the fixnum m)
                                            (the fixnum
                                                 bdspac)))))
              (wntqa
               (setf wrkbl
                     (max (the fixnum wrkbl)
                          (the fixnum
                               (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                 (f2cl-lib:int-mul m
                                                                  (ilaenv
                                                                   1
                                                                   "DORMBR"
                                                                   "QLN"
                                                                   m m
                                                                   n
                                                                   -1))))))
               (setf wrkbl
                     (max (the fixnum wrkbl)
                          (the fixnum
```

```
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                  (f2cl-lib:int-mul n
                                                                    (ilaenv
                                                                     1
                                                                     "DORMBR"
                                                                     "PRT"
                                                                     n n
                                                                     n
                                                                     -1))))))
                       (setf maxwrk
                               (max (the fixnum maxwrk)
                                    (the fixnum
                                         (f2cl-lib:int-add bdspac
                                                           (f2cl-lib:int-mul 3
                                                                             n)))))
                       (setf minwrk
                               (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                 (max (the fixnum m)
                                                      (the fixnum
                                                           bdspac)))))))))
                (t
                 (cond
                   (wntqn
                    (setf bdspac (f2cl-lib:int-mul 7 m)))
                   (t
                    (setf bdspac
                            (f2cl-lib:int-add (f2cl-lib:int-mul 3 m m)
                                              (f2cl-lib:int-mul 4 m)))))
                 (cond
                   ((>= n mnthr)
                    (cond
                      (wntqn
                       (setf wrkbl
                               (f2cl-lib:int-add m
                                                 (f2cl-lib:int-mul m
                                                                   (ilaenv 1
                                                                    "DGELQF" " " m
                                                                    n -1 -1))))
                       (setf wrkbl
                               (max (the fixnum wrkbl)
                                    (the fixnum
                                         (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                           (f2cl-lib:int-mul 2
                                                                             m
                                                                             (ilaenv
                                                                              1
```

```
                                                                  "DGEBRD"
                                                                  " "
                                                                  m m
                                                                  -1
                                                                  -1))))))
          (setf maxwrk
                  (max (the fixnum wrkbl)
                       (the fixnum
                            (f2cl-lib:int-add bdspac m))))
          (setf minwrk (f2cl-lib:int-add bdspac m)))
         (wntqo
          (setf wrkbl
                  (f2cl-lib:int-add m
                                    (f2cl-lib:int-mul m
                                                      (ilaenv 1
                                                       "DGELQF" " " m
                                                       n -1 -1))))
          (setf wrkbl
                  (max (the fixnum wrkbl)
                       (the fixnum
                            (f2cl-lib:int-add m
                                              (f2cl-lib:int-mul m
                                                                (ilaenv
                                                                 1
                                                                 "DORGLQ"
                                                                 " "
                                                                 m n
                                                                 m
                                                                 -1))))))
          (setf wrkbl
                  (max (the fixnum wrkbl)
                       (the fixnum
                            (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                              (f2cl-lib:int-mul 2
                                                                m
                                                                (ilaenv
                                                                 1
                                                                 "DGEBRD"
                                                                 " "
                                                                 m m
                                                                 -1
                                                                 -1))))))
          (setf wrkbl
                  (max (the fixnum wrkbl)
                       (the fixnum
                            (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
```

```
                                        (f2cl-lib:int-mul m
                                                          (ilaenv
                                                           1
                                                           "DORMBR"
                                                           "QLN"
                                                           m m
                                                           m
                                                           -1))))))
                 (setf wrkbl
                       (max (the fixnum wrkbl)
                            (the fixnum
                                 (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                   (f2cl-lib:int-mul m
                                                                     (ilaenv
                                                                      1
                                                                      "DORMBR"
                                                                      "PRT"
                                                                      m m
                                                                      m
                                                                      -1))))))
                 (setf wrkbl
                       (max (the fixnum wrkbl)
                            (the fixnum
                                 (f2cl-lib:int-add bdspac
                                                   (f2cl-lib:int-mul 3
                                                                     m)))))
                 (setf maxwrk
                       (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul 2 m m)))
                 (setf minwrk
                       (f2cl-lib:int-add bdspac
                                         (f2cl-lib:int-mul 2 m m)
                                         (f2cl-lib:int-mul 3 m))))
                (wntqs
                 (setf wrkbl
                       (f2cl-lib:int-add m
                                         (f2cl-lib:int-mul m
                                                           (ilaenv 1
                                                            "DGELQF" " " m
                                                            n -1 -1))))
                 (setf wrkbl
                       (max (the fixnum wrkbl)
                            (the fixnum
                                 (f2cl-lib:int-add m
                                                   (f2cl-lib:int-mul m
                                                                     (ilaenv
                                                                      1
```

```
                                                                  "DORGLQ"
                                                                  " "
                                                                  m n
                                                                  m
                                                                  -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                  (f2cl-lib:int-mul 2
                                                                    m
                                                                    (ilaenv
                                                                     1
                                                                     "DGEBRD"
                                                                     " "
                                                                     m m
                                                                     -1
                                                                     -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                  (f2cl-lib:int-mul m
                                                                    (ilaenv
                                                                     1
                                                                     "DORMBR"
                                                                     "QLN"
                                                                     m m
                                                                     m
                                                                     -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                  (f2cl-lib:int-mul m
                                                                    (ilaenv
                                                                     1
                                                                     "DORMBR"
                                                                     "PRT"
                                                                     m m
                                                                     m
                                                                     -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add bdspac
```

```
                                        (f2cl-lib:int-mul 3
                                                          m)))))
          (setf maxwrk
                (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul m m)))
          (setf minwrk
                (f2cl-lib:int-add bdspac
                                  (f2cl-lib:int-mul m m)
                                  (f2cl-lib:int-mul 3 m))))
         (wntqa
          (setf wrkbl
                (f2cl-lib:int-add m
                                  (f2cl-lib:int-mul m
                                                    (ilaenv 1
                                                     "DGELQF" " " m
                                                     n -1 -1))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add m
                                            (f2cl-lib:int-mul n
                                                              (ilaenv
                                                               1
                                                               "DORGLQ"
                                                               " "
                                                               n n
                                                               m
                                                               -1))))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                            (f2cl-lib:int-mul 2
                                                              m
                                                              (ilaenv
                                                               1
                                                               "DGEBRD"
                                                               " "
                                                               m m
                                                               -1
                                                               -1))))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                            (f2cl-lib:int-mul m
                                                              (ilaenv
```

```
                                                                    1
                                                                    "DORMBR"
                                                                    "QLN"
                                                                    m m
                                                                    m
                                                                    -1))))))
                    (setf wrkbl
                            (max (the fixnum wrkbl)
                                 (the fixnum
                                      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                        (f2cl-lib:int-mul m
                                                                          (ilaenv
                                                                           1
                                                                           "DORMBR"
                                                                           "PRT"
                                                                           m m
                                                                           m
                                                                           -1))))))
                    (setf wrkbl
                            (max (the fixnum wrkbl)
                                 (the fixnum
                                      (f2cl-lib:int-add bdspac
                                                        (f2cl-lib:int-mul 3
                                                                          m)))))
                    (setf maxwrk
                            (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul m m)))
                    (setf minwrk
                            (f2cl-lib:int-add bdspac
                                              (f2cl-lib:int-mul m m)
                                              (f2cl-lib:int-mul 3 m))))))
                (t
                 (setf wrkbl
                         (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                           (f2cl-lib:int-mul
                                            (f2cl-lib:int-add m n)
                                            (ilaenv 1 "DGEBRD" " " m n -1 -1))))
                 (cond
                   (wntqn
                    (setf maxwrk
                            (max (the fixnum wrkbl)
                                 (the fixnum
                                      (f2cl-lib:int-add bdspac
                                                        (f2cl-lib:int-mul 3
                                                                          m)))))
                    (setf minwrk
                            (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
```

```
                                (max (the fixnum n)
                                     (the fixnum
                                          bdspac)))))
                        (wntqo
                         (setf wrkbl
                                (max (the fixnum wrkbl)
                                     (the fixnum
                                          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                            (f2cl-lib:int-mul m
                                                                              (ilaenv
                                                                               1
                                                                               "DORMBR"
                                                                               "QLN"
                                                                               m m
                                                                               n
                                                                               -1))))))
                         (setf wrkbl
                                (max (the fixnum wrkbl)
                                     (the fixnum
                                          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                            (f2cl-lib:int-mul m
                                                                              (ilaenv
                                                                               1
                                                                               "DORMBR"
                                                                               "PRT"
                                                                               m n
                                                                               m
                                                                               -1))))))
                         (setf wrkbl
                                (max (the fixnum wrkbl)
                                     (the fixnum
                                          (f2cl-lib:int-add bdspac
                                                            (f2cl-lib:int-mul 3
                                                                              m)))))
                         (setf maxwrk
                                (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul m n)))
                         (setf minwrk
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                  (max (the fixnum n)
                                                       (the fixnum
                                                            (f2cl-lib:int-add
                                                             (f2cl-lib:int-mul m m)
                                                             bdspac))))))
                        (wntqs
                         (setf wrkbl
                                (max (the fixnum wrkbl)
```

```
                    (the fixnum
                         (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                           (f2cl-lib:int-mul m
                                                             (ilaenv
                                                              1
                                                              "DORMBR"
                                                              "QLN"
                                                              m m
                                                              n
                                                              -1))))))
               (setf wrkbl
                     (max (the fixnum wrkbl)
                          (the fixnum
                               (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                 (f2cl-lib:int-mul m
                                                                   (ilaenv
                                                                    1
                                                                    "DORMBR"
                                                                    "PRT"
                                                                    m n
                                                                    m
                                                                    -1))))))
               (setf maxwrk
                     (max (the fixnum wrkbl)
                          (the fixnum
                               (f2cl-lib:int-add bdspac
                                                 (f2cl-lib:int-mul 3
                                                                   m)))))
               (setf minwrk
                     (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                       (max (the fixnum n)
                                            (the fixnum
                                                 bdspac)))))
              (wntqa
               (setf wrkbl
                     (max (the fixnum wrkbl)
                          (the fixnum
                               (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                 (f2cl-lib:int-mul m
                                                                   (ilaenv
                                                                    1
                                                                    "DORMBR"
                                                                    "QLN"
                                                                    m m
                                                                    n
                                                                    -1))))))
```

```
                (setf wrkbl
                        (max (the fixnum wrkbl)
                             (the fixnum
                                  (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                    (f2cl-lib:int-mul m
                                                                      (ilaenv
                                                                       1
                                                                       "DORMBR"
                                                                       "PRT"
                                                                       n n
                                                                       m
                                                                       -1))))))
                (setf maxwrk
                        (max (the fixnum wrkbl)
                             (the fixnum
                                  (f2cl-lib:int-add bdspac
                                                    (f2cl-lib:int-mul 3
                                                                      m)))))
                (setf minwrk
                        (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                          (max (the fixnum n)
                                               (the fixnum
                                                    bdspac))))))))))
           (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                   (coerce (the fixnum maxwrk) 'double-float))))
        (cond
          ((and (< lwork minwrk) (not lquery))
           (setf info -12)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DGESDD" (f2cl-lib:int-sub info))
           (go end_label))
          (lquery
           (go end_label)))
        (cond
          ((or (= m 0) (= n 0))
           (if (>= lwork 1)
               (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) one))
           (go end_label)))
        (setf eps (dlamch "P"))
        (setf smlnum (/ (f2cl-lib:fsqrt (dlamch "S")) eps))
        (setf bignum (/ one smlnum))
        (setf anrm (dlange "M" m n a lda dum))
        (setf iscl 0)
```

```
(cond
  ((and (> anrm zero) (< anrm smlnum))
   (setf iscl 1)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
       (dlascl "G" 0 0 anrm smlnum m n a lda ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                      var-8))
     (setf ierr var-9)))
  ((> anrm bignum)
   (setf iscl 1)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
       (dlascl "G" 0 0 anrm bignum m n a lda ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                      var-8))
     (setf ierr var-9))))
(cond
  ((>= m n)
   (cond
     ((>= m mnthr)
      (cond
        (wntqn
         (setf itau 1)
         (setf nwork (f2cl-lib:int-add itau n))
         (multiple-value-bind
               (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
             (dgeqrf m n a lda
              (f2cl-lib:array-slice work double-float (itau) ((1 *)))
              (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
              (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
           (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
           (setf ierr var-7))
         (dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1) zero
          zero
          (f2cl-lib:array-slice a double-float (2 1) ((1 lda) (1 *)))
          lda)
         (setf ie 1)
         (setf itauq (f2cl-lib:int-add ie n))
         (setf itaup (f2cl-lib:int-add itauq n))
         (setf nwork (f2cl-lib:int-add itaup n))
         (multiple-value-bind
               (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                var-9 var-10)
             (dgebrd n n a lda s
              (f2cl-lib:array-slice work double-float (ie) ((1 *)))
```

```
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9))
  (setf ierr var-10))
 (setf nwork (f2cl-lib:int-add ie n))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)
     (dbdsdc "U" "N" n s
      (f2cl-lib:array-slice work double-float (ie) ((1 *))) dum
      1 dum 1 dum idum
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      iwork info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf info var-13)))
(wntqo
 (setf ir 1)
 (cond
  ((>= lwork
       (f2cl-lib:int-add (f2cl-lib:int-mul lda n)
                         (f2cl-lib:int-mul n n)
                         (f2cl-lib:int-mul 3 n)
                         bdspac))
   (setf ldwrkr lda))
  (t
   (setf ldwrkr
         (the fixnum
              (truncate (- lwork (* n n) (* 3 n) bdspac)
                        n)))))
 (setf itau (f2cl-lib:int-add ir (f2cl-lib:int-mul ldwrkr n)))
 (setf nwork (f2cl-lib:int-add itau n))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
     (dgeqrf m n a lda
      (f2cl-lib:array-slice work double-float (itau) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
  (setf ierr var-7))
 (dlacpy "U" n n a lda
  (f2cl-lib:array-slice work double-float (ir) ((1 *))) ldwrkr)
 (dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1) zero
```

```
 zero
 (f2cl-lib:array-slice work double-float ((+ ir 1)) ((1 *)))
 ldwrkr)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
    (dorgqr m n n a lda
     (f2cl-lib:array-slice work double-float (itau) ((1 *)))
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf nwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10)
    (dgebrd n n
     (f2cl-lib:array-slice work double-float (ir) ((1 *)))
     ldwrkr s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
     (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9))
  (setf ierr var-10))
(setf iu nwork)
(setf nwork (f2cl-lib:int-add iu (f2cl-lib:int-mul n n)))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
    (dbdsdc "U" "I" n s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work double-float (iu) ((1 *))) n
     vt ldvt dum idum
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     iwork info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf info var-13))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
```

```
       var-9 var-10 var-11 var-12 var-13)
     (dormbr "Q" "L" "N" n n n
      (f2cl-lib:array-slice work double-float (ir) ((1 *)))
      ldwrkr
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      (f2cl-lib:array-slice work double-float (iu) ((1 *))) n
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
   (setf ierr var-13))
 (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
     (dormbr "P" "R" "T" n n n
      (f2cl-lib:array-slice work double-float (ir) ((1 *)))
      ldwrkr
      (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
      vt ldvt
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
   (setf ierr var-13))
 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i ldwrkr))
               ((> i m) nil)
   (tagbody
     (setf chunk
             (min
              (the fixnum
                   (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1))
              (the fixnum ldwrkr)))
     (dgemm "N" "N" chunk n n one
      (f2cl-lib:array-slice a
                            double-float
                            (i 1)
                            ((1 lda) (1 *)))
      lda (f2cl-lib:array-slice work double-float (iu) ((1 *)))
      n zero
      (f2cl-lib:array-slice work double-float (ir) ((1 *)))
      ldwrkr)
     (dlacpy "F" chunk n
      (f2cl-lib:array-slice work double-float (ir) ((1 *)))
      ldwrkr
      (f2cl-lib:array-slice a
                            double-float
```

```
                                (i 1)
                                ((1 lda) (1 *)))
      lda))))
(wntqs
 (setf ir 1)
 (setf ldwrkr n)
 (setf itau (f2cl-lib:int-add ir (f2cl-lib:int-mul ldwrkr n)))
 (setf nwork (f2cl-lib:int-add itau n))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
     (dgeqrf m n a lda
      (f2cl-lib:array-slice work double-float (itau) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
   (setf ierr var-7))
 (dlacpy "U" n n a lda
  (f2cl-lib:array-slice work double-float (ir) ((1 *))) ldwrkr)
 (dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1) zero
  zero
  (f2cl-lib:array-slice work double-float ((+ ir 1)) ((1 *)))
  ldwrkr)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
     (dorgqr m n n a lda
      (f2cl-lib:array-slice work double-float (itau) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7))
   (setf ierr var-8))
 (setf ie itau)
 (setf itauq (f2cl-lib:int-add ie n))
 (setf itaup (f2cl-lib:int-add itauq n))
 (setf nwork (f2cl-lib:int-add itaup n))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10)
     (dgebrd n n
      (f2cl-lib:array-slice work double-float (ir) ((1 *)))
      ldwrkr s
      (f2cl-lib:array-slice work double-float (ie) ((1 *)))
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
```

```
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9))
  (setf ierr var-10))
 (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
    (dbdsdc "U" "I" n s
     (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
     ldu vt ldvt dum idum
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     iwork info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf info var-13))
 (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "L" "N" n n n
     (f2cl-lib:array-slice work double-float (ir) ((1 *)))
     ldwrkr
     (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
     u ldu
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
 (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
    (dormbr "P" "R" "T" n n n
     (f2cl-lib:array-slice work double-float (ir) ((1 *)))
     ldwrkr
     (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
     vt ldvt
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
 (dlacpy "F" n n u ldu
  (f2cl-lib:array-slice work double-float (ir) ((1 *))) ldwrkr)
 (dgemm "N" "N" m n n one a lda
  (f2cl-lib:array-slice work double-float (ir) ((1 *))) ldwrkr
  zero u ldu))
(wntqa
```

```
(setf iu 1)
(setf ldwrku n)
(setf itau (f2cl-lib:int-add iu (f2cl-lib:int-mul ldwrku n)))
(setf nwork (f2cl-lib:int-add itau n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgeqrf m n a lda
     (f2cl-lib:array-slice work double-float (itau) ((1 *)))
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
  (setf ierr var-7))
(dlacpy "L" m n a lda u ldu)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
    (dorgqr m m n u ldu
     (f2cl-lib:array-slice work double-float (itau) ((1 *)))
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7))
  (setf ierr var-8))
(dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1) zero
 zero
 (f2cl-lib:array-slice a double-float (2 1) ((1 lda) (1 *)))
 lda)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf nwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10)
    (dgebrd n n a lda s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
     (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
    (dbdsdc "U" "I" n s
```

```
            (f2cl-lib:array-slice work double-float (ie) ((1 *)))
            (f2cl-lib:array-slice work double-float (iu) ((1 *))) n
            vt ldvt dum idum
            (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
            iwork info)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                           var-7 var-8 var-9 var-10 var-11 var-12))
          (setf info var-13))
         (multiple-value-bind
               (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                var-9 var-10 var-11 var-12 var-13)
             (dormbr "Q" "L" "N" n n n a lda
              (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
              (f2cl-lib:array-slice work double-float (iu) ((1 *)))
              ldwrku
              (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
              (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                           var-7 var-8 var-9 var-10 var-11 var-12))
          (setf ierr var-13))
         (multiple-value-bind
               (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                var-9 var-10 var-11 var-12 var-13)
             (dormbr "P" "R" "T" n n n a lda
              (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
              vt ldvt
              (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
              (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                           var-7 var-8 var-9 var-10 var-11 var-12))
          (setf ierr var-13))
         (dgemm "N" "N" m n n one u ldu
          (f2cl-lib:array-slice work double-float (iu) ((1 *))) ldwrku
          zero a lda)
         (dlacpy "F" m n a lda u ldu))))
       (t
        (setf ie 1)
        (setf itauq (f2cl-lib:int-add ie n))
        (setf itaup (f2cl-lib:int-add itauq n))
        (setf nwork (f2cl-lib:int-add itaup n))
        (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
               var-9 var-10)
            (dgebrd m n a lda s
             (f2cl-lib:array-slice work double-float (ie) ((1 *)))
             (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
```

```
      (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                     var-7 var-8 var-9))
    (setf ierr var-10))
  (cond
    (wntqn
     (multiple-value-bind
           (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
            var-9 var-10 var-11 var-12 var-13)
         (dbdsdc "U" "N" n s
          (f2cl-lib:array-slice work double-float (ie) ((1 *))) dum
          1 dum 1 dum idum
          (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
          iwork info)
       (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                        var-7 var-8 var-9 var-10 var-11 var-12))
       (setf info var-13)))
    (wntqo
     (setf iu nwork)
     (cond
       ((>= lwork
            (f2cl-lib:int-add (f2cl-lib:int-mul m n)
                              (f2cl-lib:int-mul 3 n)
                              bdspac))
        (setf ldwrku m)
        (setf nwork
              (f2cl-lib:int-add iu (f2cl-lib:int-mul ldwrku n)))
        (dlaset "F" m n zero zero
         (f2cl-lib:array-slice work double-float (iu) ((1 *)))
         ldwrku))
       (t
        (setf ldwrku n)
        (setf nwork
              (f2cl-lib:int-add iu (f2cl-lib:int-mul ldwrku n)))
        (setf ir nwork)
        (setf ldwrkr
              (the fixnum
                   (truncate (- lwork (* n n) (* 3 n)) n)))))
     (setf nwork (f2cl-lib:int-add iu (f2cl-lib:int-mul ldwrku n)))
     (multiple-value-bind
           (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
            var-9 var-10 var-11 var-12 var-13)
         (dbdsdc "U" "I" n s
          (f2cl-lib:array-slice work double-float (ie) ((1 *)))
```

```
     (f2cl-lib:array-slice work double-float (iu) ((1 *)))
     ldwrku vt ldvt dum idum
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     iwork info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf info var-13))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
    (dormbr "P" "R" "T" n n n a lda
     (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
     vt ldvt
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
(cond
  ((>= lwork
       (f2cl-lib:int-add (f2cl-lib:int-mul m n)
                         (f2cl-lib:int-mul 3 n)
                         bdspac))
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
          var-8 var-9 var-10 var-11 var-12 var-13)
       (dormbr "Q" "L" "N" m n n a lda
        (f2cl-lib:array-slice work
                              double-float
                              (itauq)
                              ((1 *)))
        (f2cl-lib:array-slice work double-float (iu) ((1 *)))
        ldwrku
        (f2cl-lib:array-slice work
                              double-float
                              (nwork)
                              ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1)
        ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                      var-6 var-7 var-8 var-9 var-10 var-11
                      var-12))
     (setf ierr var-13))
   (dlacpy "F" m n
    (f2cl-lib:array-slice work double-float (iu) ((1 *)))
    ldwrku a lda))
```

```
                    (t
                     (multiple-value-bind
                           (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                            var-8 var-9)
                         (dorgbr "Q" m n n a lda
                          (f2cl-lib:array-slice work
                                                double-float
                                                (itauq)
                                                ((1 *)))
                          (f2cl-lib:array-slice work
                                                double-float
                                                (nwork)
                                                ((1 *)))
                          (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1)
                          ierr)
                       (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                                        var-6 var-7 var-8))
                       (setf ierr var-9))
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i ldwrkr))
                                   ((> i m) nil)
                       (tagbody
                         (setf chunk
                                 (min
                                  (the fixnum
                                       (f2cl-lib:int-add (f2cl-lib:int-sub m i)
                                                         1))
                                  (the fixnum ldwrkr)))
                         (dgemm "N" "N" chunk n n one
                          (f2cl-lib:array-slice a
                                                double-float
                                                (i 1)
                                                ((1 lda) (1 *)))
                          lda
                          (f2cl-lib:array-slice work double-float (iu) ((1 *)))
                          ldwrku zero
                          (f2cl-lib:array-slice work double-float (ir) ((1 *)))
                          ldwrkr)
                         (dlacpy "F" chunk n
                          (f2cl-lib:array-slice work double-float (ir) ((1 *)))
                          ldwrkr
                          (f2cl-lib:array-slice a
                                                double-float
                                                (i 1)
                                                ((1 lda) (1 *)))
                          lda))))))
                (wntqs
```

```
 (dlaset "F" m n zero zero u ldu)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)
     (dbdsdc "U" "I" n s
      (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
      ldu vt ldvt dum idum
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      iwork info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
   (setf info var-13))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)
     (dormbr "Q" "L" "N" m n n a lda
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      u ldu
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
   (setf ierr var-13))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)
     (dormbr "P" "R" "T" n n n a lda
      (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
      vt ldvt
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
   (setf ierr var-13)))
(wntqa
 (dlaset "F" m m zero zero u ldu)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)
     (dbdsdc "U" "I" n s
      (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
      ldu vt ldvt dum idum
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      iwork info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
```

```
                (setf info var-13))
               (dlaset "F" (f2cl-lib:int-sub m n) (f2cl-lib:int-sub m n) zero
                one
                (f2cl-lib:array-slice u
                                      double-float
                                      ((+ n 1) (f2cl-lib:int-add n 1))
                                      ((1 ldu) (1 *)))
                ldu)
               (multiple-value-bind
                     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                      var-9 var-10 var-11 var-12 var-13)
                   (dormbr "Q" "L" "N" m m n a lda
                    (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
                    u ldu
                    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
                    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
                 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                  var-7 var-8 var-9 var-10 var-11 var-12))
                 (setf ierr var-13))
               (multiple-value-bind
                     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                      var-9 var-10 var-11 var-12 var-13)
                   (dormbr "P" "R" "T" n n m a lda
                    (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
                    vt ldvt
                    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
                    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
                 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                  var-7 var-8 var-9 var-10 var-11 var-12))
                 (setf ierr var-13)))))))
          (t
           (cond
             ((>= n mnthr)
              (cond
                (wntqn
                 (setf itau 1)
                 (setf nwork (f2cl-lib:int-add itau m))
                 (multiple-value-bind
                       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
                     (dgelqf m n a lda
                      (f2cl-lib:array-slice work double-float (itau) ((1 *)))
                      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
                      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
                   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
                   (setf ierr var-7))
                 (dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1) zero
```

```
 zero
 (f2cl-lib:array-slice a double-float (1 2) ((1 lda) (1 *)))
 lda)
(setf ie 1)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf nwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10)
    (dgebrd m m a lda s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
     (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9))
  (setf ierr var-10))
(setf nwork (f2cl-lib:int-add ie m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
    (dbdsdc "U" "N" m s
     (f2cl-lib:array-slice work double-float (ie) ((1 *))) dum
     1 dum 1 dum idum
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     iwork info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf info var-13)))
(wntqo
 (setf ivt 1)
 (setf il (f2cl-lib:int-add ivt (f2cl-lib:int-mul m m)))
 (cond
   ((>= lwork
        (f2cl-lib:int-add (f2cl-lib:int-mul m n)
                          (f2cl-lib:int-mul m m)
                          (f2cl-lib:int-mul 3 m)
                          bdspac))
    (setf ldwrkl m)
    (setf chunk n))
   (t
    (setf ldwrkl m)
    (setf chunk
            (the fixnum
```

```
               (truncate (- lwork (* m m)) m)))))
(setf itau (f2cl-lib:int-add il (f2cl-lib:int-mul ldwrkl m)))
(setf nwork (f2cl-lib:int-add itau m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgelqf m n a lda
     (f2cl-lib:array-slice work double-float (itau) ((1 *)))
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
  (setf ierr var-7))
(dlacpy "L" m m a lda
 (f2cl-lib:array-slice work double-float (il) ((1 *))) ldwrkl)
(dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1) zero
 zero
 (f2cl-lib:array-slice work
                       double-float
                       ((+ il ldwrkl))
                       ((1 *)))
 ldwrkl)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
    (dorglq m n m a lda
     (f2cl-lib:array-slice work double-float (itau) ((1 *)))
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf nwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10)
    (dgebrd m m
     (f2cl-lib:array-slice work double-float (il) ((1 *)))
     ldwrkl s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
     (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9))
```

```
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
    (dbdsdc "U" "I" m s
     (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
     ldu
     (f2cl-lib:array-slice work double-float (ivt) ((1 *))) m
     dum idum
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     iwork info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf info var-13))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "L" "N" m m m
     (f2cl-lib:array-slice work double-float (il) ((1 *)))
     ldwrkl
     (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
     u ldu
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
    (dormbr "P" "R" "T" m m m
     (f2cl-lib:array-slice work double-float (il) ((1 *)))
     ldwrkl
     (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
     (f2cl-lib:array-slice work double-float (ivt) ((1 *))) m
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i chunk))
              ((> i n) nil)
  (tagbody
    (setf blk
            (min
             (the fixnum
```

```
                    (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1))
                (the fixnum chunk)))
          (dgemm "N" "N" m blk m one
           (f2cl-lib:array-slice work double-float (ivt) ((1 *))) m
           (f2cl-lib:array-slice a
                                 double-float
                                 (1 i)
                                 ((1 lda) (1 *)))
           lda zero
           (f2cl-lib:array-slice work double-float (il) ((1 *)))
           ldwrkl)
          (dlacpy "F" m blk
           (f2cl-lib:array-slice work double-float (il) ((1 *)))
           ldwrkl
           (f2cl-lib:array-slice a
                                 double-float
                                 (1 i)
                                 ((1 lda) (1 *)))
           lda))))
      (wntqs
       (setf il 1)
       (setf ldwrkl m)
       (setf itau (f2cl-lib:int-add il (f2cl-lib:int-mul ldwrkl m)))
       (setf nwork (f2cl-lib:int-add itau m))
       (multiple-value-bind
             (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
          (dgelqf m n a lda
           (f2cl-lib:array-slice work double-float (itau) ((1 *)))
           (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
           (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
         (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
         (setf ierr var-7))
       (dlacpy "L" m m a lda
        (f2cl-lib:array-slice work double-float (il) ((1 *))) ldwrkl)
       (dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1) zero
        zero
        (f2cl-lib:array-slice work
                              double-float
                              ((+ il ldwrkl))
                              ((1 *)))
        ldwrkl)
       (multiple-value-bind
             (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
          (dorglq m n m a lda
           (f2cl-lib:array-slice work double-float (itau) ((1 *)))
           (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
```

```
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf nwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10)
    (dgebrd m m
     (f2cl-lib:array-slice work double-float (il) ((1 *)))
     ldwrkl s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
     (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
    (dbdsdc "U" "I" m s
     (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
     ldu vt ldvt dum idum
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     iwork info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf info var-13))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "L" "N" m m m
     (f2cl-lib:array-slice work double-float (il) ((1 *)))
     ldwrkl
     (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
     u ldu
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
```

```
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)
     (dormbr "P" "R" "T" m m m
      (f2cl-lib:array-slice work double-float (il) ((1 *)))
      ldwrkl
      (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
      vt ldvt
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
   (setf ierr var-13))
 (dlacpy "F" m m vt ldvt
  (f2cl-lib:array-slice work double-float (il) ((1 *))) ldwrkl)
 (dgemm "N" "N" m n m one
  (f2cl-lib:array-slice work double-float (il) ((1 *))) ldwrkl
  a lda zero vt ldvt))
(wntqa
 (setf ivt 1)
 (setf ldwkvt m)
 (setf itau (f2cl-lib:int-add ivt (f2cl-lib:int-mul ldwkvt m)))
 (setf nwork (f2cl-lib:int-add itau m))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
     (dgelqf m n a lda
      (f2cl-lib:array-slice work double-float (itau) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
   (setf ierr var-7))
 (dlacpy "U" m n a lda vt ldvt)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
     (dorglq n n m vt ldvt
      (f2cl-lib:array-slice work double-float (itau) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7))
   (setf ierr var-8))
 (dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1) zero
  zero
  (f2cl-lib:array-slice a double-float (1 2) ((1 lda) (1 *)))
  lda)
 (setf ie itau)
```

```
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf nwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10)
    (dgebrd m m a lda s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
     (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
    (dbdsdc "U" "I" m s
     (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
     ldu
     (f2cl-lib:array-slice work double-float (ivt) ((1 *)))
     ldwkvt dum idum
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     iwork info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf info var-13))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "L" "N" m m m a lda
     (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
     u ldu
     (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13)
    (dormbr "P" "R" "T" m m m a lda
     (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
     (f2cl-lib:array-slice work double-float (ivt) ((1 *)))
     ldwkvt
```

```
          (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
          (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                         var-7 var-8 var-9 var-10 var-11 var-12))
        (setf ierr var-13))
      (dgemm "N" "N" m n m one
       (f2cl-lib:array-slice work double-float (ivt) ((1 *))) ldwkvt
       vt ldvt zero a lda)
      (dlacpy "F" m n a lda vt ldvt))))
  (t
   (setf ie 1)
   (setf itauq (f2cl-lib:int-add ie m))
   (setf itaup (f2cl-lib:int-add itauq m))
   (setf nwork (f2cl-lib:int-add itaup m))
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
          var-9 var-10)
       (dgebrd m n a lda s
        (f2cl-lib:array-slice work double-float (ie) ((1 *)))
        (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
        (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
        (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7 var-8 var-9))
     (setf ierr var-10))
   (cond
     (wntqn
      (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
             var-9 var-10 var-11 var-12 var-13)
          (dbdsdc "L" "N" m s
           (f2cl-lib:array-slice work double-float (ie) ((1 *))) dum
           1 dum 1 dum idum
           (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
           iwork info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                         var-7 var-8 var-9 var-10 var-11 var-12))
        (setf info var-13)))
     (wntqo
      (setf ldwkvt m)
      (setf ivt nwork)
      (cond
        ((>= lwork
             (f2cl-lib:int-add (f2cl-lib:int-mul m n)
                               (f2cl-lib:int-mul 3 m)
```

```
                          bdspac))
    (dlaset "F" m n zero zero
     (f2cl-lib:array-slice work double-float (ivt) ((1 *)))
     ldwkvt)
    (setf nwork
            (f2cl-lib:int-add ivt (f2cl-lib:int-mul ldwkvt n))))
   (t
    (setf nwork
            (f2cl-lib:int-add ivt (f2cl-lib:int-mul ldwkvt m)))
    (setf il nwork)
    (setf chunk
            (the fixnum
                 (truncate (- lwork (* m m) (* 3 m)) m)))))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)
     (dbdsdc "L" "I" m s
      (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
      ldu
      (f2cl-lib:array-slice work double-float (ivt) ((1 *)))
      ldwkvt dum idum
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      iwork info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
   (setf info var-13))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)
     (dormbr "Q" "L" "N" m m n a lda
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      u ldu
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
   (setf ierr var-13))
 (cond
   ((>= lwork
        (f2cl-lib:int-add (f2cl-lib:int-mul m n)
                          (f2cl-lib:int-mul 3 m)
                          bdspac))
    (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
           var-8 var-9 var-10 var-11 var-12 var-13)
        (dormbr "P" "R" "T" m n m a lda
```

```
      (f2cl-lib:array-slice work
                            double-float
                            (itaup)
                            ((1 *)))
      (f2cl-lib:array-slice work double-float (ivt) ((1 *)))
      ldwkvt
      (f2cl-lib:array-slice work
                            double-float
                            (nwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                     var-6 var-7 var-8 var-9 var-10 var-11
                     var-12))
    (setf ierr var-13))
  (dlacpy "F" m n
   (f2cl-lib:array-slice work double-float (ivt) ((1 *)))
   ldwkvt a lda))
 (t
  (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
         var-8 var-9)
      (dorgbr "P" m n m a lda
       (f2cl-lib:array-slice work
                             double-float
                             (itaup)
                             ((1 *)))
       (f2cl-lib:array-slice work
                             double-float
                             (nwork)
                             ((1 *)))
       (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1)
       ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                     var-6 var-7 var-8))
    (setf ierr var-9))
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i chunk))
                ((> i n) nil)
    (tagbody
      (setf blk
              (min
               (the fixnum
                    (f2cl-lib:int-add (f2cl-lib:int-sub n i)
                                      1))
               (the fixnum chunk)))
```

```
          (dgemm "N" "N" m blk m one
           (f2cl-lib:array-slice work double-float (ivt) ((1 *)))
           ldwkvt
           (f2cl-lib:array-slice a
                                 double-float
                                 (1 i)
                                 ((1 lda) (1 *)))
           lda zero
           (f2cl-lib:array-slice work double-float (il) ((1 *)))
           m)
          (dlacpy "F" m blk
           (f2cl-lib:array-slice work double-float (il) ((1 *)))
           m
           (f2cl-lib:array-slice a
                                 double-float
                                 (1 i)
                                 ((1 lda) (1 *)))
           lda))))))
(wntqs
 (dlaset "F" m n zero zero vt ldvt)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)
     (dbdsdc "L" "I" m s
      (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
      ldu vt ldvt dum idum
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      iwork info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
   (setf info var-13))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)
     (dormbr "Q" "L" "N" m m n a lda
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      u ldu
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
   (setf ierr var-13))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)
     (dormbr "P" "R" "T" m n m a lda
```

```
      (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
      vt ldvt
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                     var-7 var-8 var-9 var-10 var-11 var-12))
    (setf ierr var-13)))
(wntqa
 (dlaset "F" n n zero zero vt ldvt)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)
     (dbdsdc "L" "I" m s
      (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
      ldu vt ldvt dum idum
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      iwork info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                     var-7 var-8 var-9 var-10 var-11 var-12))
    (setf info var-13))
 (dlaset "F" (f2cl-lib:int-sub n m) (f2cl-lib:int-sub n m) zero
  one
  (f2cl-lib:array-slice vt
                        double-float
                        ((+ m 1) (f2cl-lib:int-add m 1))
                        ((1 ldvt) (1 *)))
  ldvt)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)
     (dormbr "Q" "L" "N" m m n a lda
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      u ldu
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                     var-7 var-8 var-9 var-10 var-11 var-12))
    (setf ierr var-13))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)
     (dormbr "P" "R" "T" n n m a lda
      (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
      vt ldvt
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
```

```
                    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                     var-7 var-8 var-9 var-10 var-11 var-12))
                    (setf ierr var-13)))))))))
          (cond
            ((= iscl 1)
             (if (> anrm bignum)
                 (multiple-value-bind
                       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                        var-9)
                     (dlascl "G" 0 0 bignum anrm minmn 1 s minmn ierr)
                   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                    var-7 var-8))
                   (setf ierr var-9)))
             (if (< anrm smlnum)
                 (multiple-value-bind
                       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                        var-9)
                     (dlascl "G" 0 0 smlnum anrm minmn 1 s minmn ierr)
                   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                    var-7 var-8))
                   (setf ierr var-9)))))
          (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                  (coerce (realpart maxwrk) 'double-float))
   end_label
          (return
           (values nil nil nil nil nil nil nil nil nil nil nil nil nil info))))))
```

7.17 dgesvd LAPACK

⟨*dgesvd.input*⟩≡

```
  )set break resume
  )sys rm -f dgesvd.output
  )spool dgesvd.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dgesvd.help*⟩≡

```
======================================================================
dgesvd examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DGESVD - the singular value decomposition (SVD) of a real M-by-N matrix
       A, optionally computing the left and/or right singular vectors

SYNOPSIS
       SUBROUTINE DGESVD( JOBU, JOBVT, M, N, A, LDA,  S,  U,  LDU,  VT,  LDVT,
                          WORK, LWORK, INFO )

           CHARACTER      JOBU, JOBVT

           INTEGER        INFO, LDA, LDU, LDVT, LWORK, M, N

           DOUBLE         PRECISION  A(  LDA,  *  ),  S( * ), U( LDU, * ), VT(
                          LDVT, * ), WORK( * )

PURPOSE
       DGESVD computes the singular value decomposition (SVD) of a real M-by-N
       matrix  A, optionally computing the left and/or right singular vectors.
       The SVD is written

            A = U * SIGMA * transpose(V)

       where SIGMA is an M-by-N matrix which is zero except for  its  min(m,n)
       diagonal elements, U is an M-by-M orthogonal matrix, and V is an N-by-N
       orthogonal matrix.  The diagonal elements of  SIGMA  are  the  singular
       values  of  A;  they  are  real  and  non-negative, and are returned in
       descending order.  The first min(m,n) columns of U and V are  the  left
       and right singular vectors of A.

       Note that the routine returns V**T, not V.


ARGUMENTS
       JOBU    (input) CHARACTER*1
               Specifies options for computing all or part of the matrix U:
               = 'A':  all M columns of U are returned in array U:
               = 'S':  the first min(m,n) columns of U (the left singular vec-
```

```
          tors) are returned in the array U; = 'O':  the  first  min(m,n)
          columns of U (the left singular vectors) are overwritten on the
          array A; = 'N':  no columns of U (no left singular vectors) are
          computed.

JOBVT     (input) CHARACTER*1
          Specifies options for computing all or part of the matrix V**T:
          = 'A':  all N rows of V**T are returned in the array VT;
          = 'S':  the first min(m,n) rows of  V**T  (the  right  singular
          vectors)  are  returned  in  the  array  VT;  = 'O':  the first
          min(m,n) rows of V**T (the right singular  vectors)  are  over-
          written  on the array A; = 'N':  no rows of V**T (no right sin-
          gular vectors) are computed.

          JOBVT and JOBU cannot both be 'O'.

M         (input) INTEGER
          The number of rows of the input matrix A.  M >= 0.

N         (input) INTEGER
          The number of columns of the input matrix A.  N >= 0.

A         (input/output) DOUBLE PRECISION array, dimension (LDA,N)
          On entry, the M-by-N matrix A.  On exit, if JOBU = 'O',   A  is
          overwritten  with  the  first  min(m,n)  columns of U (the left
          singular vectors, stored columnwise); if  JOBVT  =  'O',  A  is
          overwritten  with  the  first  min(m,n) rows of V**T (the right
          singular vectors, stored rowwise); if JOBU .ne. 'O'  and  JOBVT
          .ne. 'O', the contents of A are destroyed.

LDA       (input) INTEGER
          The leading dimension of the array A.  LDA >= max(1,M).

S         (output) DOUBLE PRECISION array, dimension (min(M,N))
          The singular values of A, sorted so that S(i) >= S(i+1).

U         (output) DOUBLE PRECISION array, dimension (LDU,UCOL)
          (LDU,M) if JOBU = 'A' or (LDU,min(M,N)) if JOBU = 'S'.  If JOBU
          = 'A', U contains the M-by-M orthogonal matrix  U;  if  JOBU  =
          'S',  U contains the first min(m,n) columns of U (the left sin-
          gular vectors, stored columnwise); if JOBU = 'N' or 'O',  U  is
          not referenced.

LDU       (input) INTEGER
          The  leading dimension of the array U.  LDU >= 1; if JOBU = 'S'
          or 'A', LDU >= M.
```

```
VT      (output) DOUBLE PRECISION array, dimension (LDVT,N)
        If JOBVT = 'A', VT contains the N-by-N orthogonal matrix  V**T;
        if  JOBVT  =  'S',  VT contains the first min(m,n) rows of V**T
        (the right singular vectors, stored rowwise); if JOBVT = 'N' or
        'O', VT is not referenced.

LDVT    (input) INTEGER
        The  leading  dimension of the array VT.  LDVT >= 1; if JOBVT =
        'A', LDVT >= N; if JOBVT = 'S', LDVT >= min(M,N).

WORK      (workspace/output)  DOUBLE  PRECISION  array,  dimension
(MAX(1,LWORK))
        On exit, if INFO = 0, WORK(1) returns  the  optimal  LWORK;  if
        INFO > 0, WORK(2:MIN(M,N)) contains the unconverged superdiago-
        nal elements of an upper bidiagonal matrix B whose diagonal  is
        in  S  (not necessarily sorted). B satisfies A = U * B * VT, so
        it has the same singular values  as  A,  and  singular  vectors
        related by U and VT.

LWORK   (input) INTEGER
        The    dimension    of    the    array    WORK.     LWORK    >=
        MAX(1,3*MIN(M,N)+MAX(M,N),5*MIN(M,N)).  For  good  performance,
        LWORK should generally be larger.

        If  LWORK  = -1, then a workspace query is assumed; the routine
        only calculates the optimal size of  the  WORK  array,  returns
        this  value  as the first entry of the WORK array, and no error
        message related to LWORK is issued by XERBLA.

INFO    (output) INTEGER
        = 0:  successful exit.
        < 0:  if INFO = -i, the i-th argument had an illegal value.
        > 0:  if DBDSQR did  not  converge,  INFO  specifies  how  many
        superdiagonals  of  an  intermediate  bidiagonal form B did not
        converge to  zero.  See  the  description  of  WORK  above  for
        details.
```

⟨*LAPACK dgesvd*⟩≡

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dgesvd (jobu jobvt m n a lda s u ldu vt ldvt work lwork info)
    (declare (type (simple-array double-float (*)) work vt u s a)
             (type fixnum info lwork ldvt ldu lda n m)
             (type character jobvt jobu))
    (f2cl-lib:with-multi-array-data
        ((jobu character jobu-%data% jobu-%offset%)
         (jobvt character jobvt-%data% jobvt-%offset%)
         (a double-float a-%data% a-%offset%)
         (s double-float s-%data% s-%offset%)
         (u double-float u-%data% u-%offset%)
         (vt double-float vt-%data% vt-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((dum (make-array 1 :element-type 'double-float)) (anrm 0.0)
             (bignum 0.0) (eps 0.0) (smlnum 0.0) (bdspac 0) (blk 0) (chunk 0)
             (i 0) (ie 0) (ierr 0) (ir 0) (iscl 0) (itau 0) (itaup 0) (itauq 0)
             (iu 0) (iwork 0) (ldwrkr 0) (ldwrku 0) (maxwrk 0) (minmn 0)
             (minwrk 0) (mnthr 0) (ncu 0) (ncvt 0) (nru 0) (nrvt 0) (wrkbl 0)
             (lquery nil) (wntua nil) (wntuas nil) (wntun nil) (wntuo nil)
             (wntus nil) (wntva nil) (wntvas nil) (wntvn nil) (wntvo nil)
             (wntvs nil))
        (declare (type (simple-array double-float (1)) dum)
                 (type (double-float) anrm bignum eps smlnum)
                 (type fixnum bdspac blk chunk i ie ierr ir iscl
                                        itau itaup itauq iu iwork ldwrkr
                                        ldwrku maxwrk minmn minwrk mnthr ncu
                                        ncvt nru nrvt wrkbl)
                 (type (member t nil) lquery wntua wntuas wntun wntuo wntus
                                      wntva wntvas wntvn wntvo wntvs))
        (setf info 0)
        (setf minmn (min (the fixnum m) (the fixnum n)))
        (setf mnthr (ilaenv 6 "DGESVD" (f2cl-lib:f2cl-// jobu jobvt) m n 0 0))
        (setf wntua (char-equal jobu #\A))
        (setf wntus (char-equal jobu #\S))
        (setf wntuas (or wntua wntus))
        (setf wntuo (char-equal jobu #\O))
        (setf wntun (char-equal jobu #\N))
        (setf wntva (char-equal jobvt #\A))
        (setf wntvs (char-equal jobvt #\S))
        (setf wntvas (or wntva wntvs))
        (setf wntvo (char-equal jobvt #\O))
        (setf wntvn (char-equal jobvt #\N))
        (setf minwrk 1)
```

```
(setf lquery (coerce (= lwork -1) '(member t nil)))
(cond
  ((not (or wntua wntus wntuo wntun))
   (setf info -1))
  ((or (not (or wntva wntvs wntvo wntvn)) (and wntvo wntuo))
   (setf info -2))
  ((< m 0)
   (setf info -3))
  ((< n 0)
   (setf info -4))
  ((< lda (max (the fixnum 1) (the fixnum m)))
   (setf info -6))
  ((or (< ldu 1) (and wntuas (< ldu m)))
   (setf info -9))
  ((or (< ldvt 1) (and wntva (< ldvt n)) (and wntvs (< ldvt minmn)))
   (setf info -11)))
(cond
  ((and (= info 0) (or (>= lwork 1) lquery) (> m 0) (> n 0))
   (cond
     ((>= m n)
      (setf bdspac (f2cl-lib:int-mul 5 n))
      (cond
        ((>= m mnthr)
         (cond
           (wntun
            (setf maxwrk
                    (f2cl-lib:int-add n
                                      (f2cl-lib:int-mul n
                                                        (ilaenv 1
                                                         "DGEQRF" " " m
                                                         n -1 -1))))
            (setf maxwrk
                    (max (the fixnum maxwrk)
                         (the fixnum
                              (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                (f2cl-lib:int-mul 2
                                                                  n
                                                                  (ilaenv
                                                                   1
                                                                   "DGEBRD"
                                                                   " "
                                                                   n n
                                                                   -1
                                                                   -1))))))
            (if (or wntvo wntvas)
                (setf maxwrk
```

```
                    (max (the fixnum maxwrk)
                         (the fixnum
                              (f2cl-lib:int-add
                               (f2cl-lib:int-mul 3 n)
                               (f2cl-lib:int-mul
                                (f2cl-lib:int-sub n 1)
                                (ilaenv 1 "DORGBR" "P" n n n
                                 -1)))))))
                 (setf maxwrk
                         (max (the fixnum maxwrk)
                              (the fixnum bdspac)))
                 (setf minwrk
                         (max (the fixnum (f2cl-lib:int-mul 4 n))
                              (the fixnum bdspac)))
                 (setf maxwrk
                         (max (the fixnum maxwrk)
                              (the fixnum minwrk))))
                ((and wntuo wntvn)
                 (setf wrkbl
                         (f2cl-lib:int-add n
                                           (f2cl-lib:int-mul n
                                                             (ilaenv 1
                                                              "DGEQRF" " " m
                                                              n -1 -1))))
                 (setf wrkbl
                         (max (the fixnum wrkbl)
                              (the fixnum
                                   (f2cl-lib:int-add n
                                                     (f2cl-lib:int-mul n
                                                                       (ilaenv
                                                                        1
                                                                        "DORGQR"
                                                                        " "
                                                                        m n
                                                                        n
                                                                        -1))))))
                 (setf wrkbl
                         (max (the fixnum wrkbl)
                              (the fixnum
                                   (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                     (f2cl-lib:int-mul 2
                                                                       n
                                                                       (ilaenv
                                                                        1
                                                                        "DGEBRD"
                                                                        " "
```

```
                                                                n n
                                                                -1
                                                                -1))))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                            (f2cl-lib:int-mul n
                                                              (ilaenv
                                                               1
                                                               "DORGBR"
                                                               "Q"
                                                               n n
                                                               n
                                                               -1))))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum bdspac)))
          (setf maxwrk
                (max
                 (the fixnum
                      (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                                        wrkbl))
                 (the fixnum
                      (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                                        (f2cl-lib:int-mul m n)
                                        n))))
          (setf minwrk
                (max
                 (the fixnum
                      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
                 (the fixnum bdspac)))
          (setf maxwrk
                (max (the fixnum maxwrk)
                     (the fixnum minwrk))))
         ((and wntuo wntvas)
          (setf wrkbl
                (f2cl-lib:int-add n
                                  (f2cl-lib:int-mul n
                                                    (ilaenv 1
                                                     "DGEQRF" " " m
                                                     n -1 -1))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add n
```

```
                                        (f2cl-lib:int-mul n
                                                          (ilaenv
                                                           1
                                                           "DORGQR"
                                                           " "
                                                           m n
                                                           n
                                                           -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                  (f2cl-lib:int-mul 2
                                                                    n
                                                                    (ilaenv
                                                                     1
                                                                     "DGEBRD"
                                                                     " "
                                                                     n n
                                                                     -1
                                                                     -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                  (f2cl-lib:int-mul n
                                                                    (ilaenv
                                                                     1
                                                                     "DORGBR"
                                                                     "Q"
                                                                     n n
                                                                     n
                                                                     -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                  (f2cl-lib:int-mul
                                                   (f2cl-lib:int-sub n 1)
                                                   (ilaenv 1 "DORGBR" "P"
                                                    n n n -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum bdspac)))
                (setf maxwrk
                      (max
```

```
                (the fixnum
                     (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                                       wrkbl))
                (the fixnum
                     (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                                       (f2cl-lib:int-mul m n)
                                       n))))
         (setf minwrk
                 (max
                  (the fixnum
                       (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
                  (the fixnum bdspac)))
         (setf maxwrk
                 (max (the fixnum maxwrk)
                      (the fixnum minwrk))))
        ((and wntus wntvn)
         (setf wrkbl
                 (f2cl-lib:int-add n
                                   (f2cl-lib:int-mul n
                                                     (ilaenv 1
                                                      "DGEQRF" " " m
                                                      n -1 -1))))
         (setf wrkbl
                 (max (the fixnum wrkbl)
                      (the fixnum
                           (f2cl-lib:int-add n
                                             (f2cl-lib:int-mul n
                                                               (ilaenv
                                                                1
                                                                "DORGQR"
                                                                " "
                                                                m n
                                                                n
                                                                -1))))))
         (setf wrkbl
                 (max (the fixnum wrkbl)
                      (the fixnum
                           (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                             (f2cl-lib:int-mul 2
                                                               n
                                                               (ilaenv
                                                                1
                                                                "DGEBRD"
                                                                " "
                                                                n n
                                                                -1
```

```
                                                                  -1))))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                            (f2cl-lib:int-mul n
                                                              (ilaenv
                                                               1
                                                               "DORGBR"
                                                               "Q"
                                                               n n
                                                               n
                                                               -1))))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum bdspac)))
          (setf maxwrk
                (f2cl-lib:int-add (f2cl-lib:int-mul n n) wrkbl))
          (setf minwrk
                (max
                 (the fixnum
                      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
                 (the fixnum bdspac)))
          (setf maxwrk
                (max (the fixnum maxwrk)
                     (the fixnum minwrk))))
         ((and wntus wntvo)
          (setf wrkbl
                (f2cl-lib:int-add n
                                  (f2cl-lib:int-mul n
                                                    (ilaenv 1
                                                     "DGEQRF" " " m
                                                     n -1 -1))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add n
                                            (f2cl-lib:int-mul n
                                                              (ilaenv
                                                               1
                                                               "DORGQR"
                                                               " "
                                                               m n
                                                               n
                                                               -1))))))
          (setf wrkbl
```

```
        (max (the fixnum wrkbl)
             (the fixnum
                  (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                    (f2cl-lib:int-mul 2
                                                      n
                                                      (ilaenv
                                                       1
                                                       "DGEBRD"
                                                       " "
                                                       n n
                                                       -1
                                                       -1))))))
 (setf wrkbl
        (max (the fixnum wrkbl)
             (the fixnum
                  (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                    (f2cl-lib:int-mul n
                                                      (ilaenv
                                                       1
                                                       "DORGBR"
                                                       "Q"
                                                       n n
                                                       n
                                                       -1))))))
 (setf wrkbl
        (max (the fixnum wrkbl)
             (the fixnum
                  (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                    (f2cl-lib:int-mul
                                     (f2cl-lib:int-sub n 1)
                                     (ilaenv 1 "DORGBR" "P"
                                      n n n -1))))))
 (setf wrkbl
        (max (the fixnum wrkbl)
             (the fixnum bdspac)))
 (setf maxwrk
        (f2cl-lib:int-add (f2cl-lib:int-mul 2 n n) wrkbl))
 (setf minwrk
        (max
         (the fixnum
              (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
         (the fixnum bdspac)))
 (setf maxwrk
        (max (the fixnum maxwrk)
             (the fixnum minwrk))))
((and wntus wntvas)
```

```
(setf wrkbl
        (f2cl-lib:int-add n
                          (f2cl-lib:int-mul n
                                            (ilaenv 1
                                             "DGEQRF" " " m
                                             n -1 -1))))
(setf wrkbl
        (max (the fixnum wrkbl)
             (the fixnum
                  (f2cl-lib:int-add n
                                    (f2cl-lib:int-mul n
                                                      (ilaenv
                                                       1
                                                       "DORGQR"
                                                       " "
                                                       m n
                                                       n
                                                       -1))))))
(setf wrkbl
        (max (the fixnum wrkbl)
             (the fixnum
                  (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                    (f2cl-lib:int-mul 2
                                                      n
                                                      (ilaenv
                                                       1
                                                       "DGEBRD"
                                                       " "
                                                       n n
                                                       -1
                                                       -1))))))
(setf wrkbl
        (max (the fixnum wrkbl)
             (the fixnum
                  (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                    (f2cl-lib:int-mul n
                                                      (ilaenv
                                                       1
                                                       "DORGBR"
                                                       "Q"
                                                       n n
                                                       n
                                                       -1))))))
(setf wrkbl
        (max (the fixnum wrkbl)
             (the fixnum
```

```
                    (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                      (f2cl-lib:int-mul
                                       (f2cl-lib:int-sub n 1)
                                       (ilaenv 1 "DORGBR" "P"
                                        n n n -1))))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum bdspac)))
          (setf maxwrk
                (f2cl-lib:int-add (f2cl-lib:int-mul n n) wrkbl))
          (setf minwrk
                (max
                 (the fixnum
                      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
                 (the fixnum bdspac)))
          (setf maxwrk
                (max (the fixnum maxwrk)
                     (the fixnum minwrk))))
         ((and wntua wntvn)
          (setf wrkbl
                (f2cl-lib:int-add n
                                  (f2cl-lib:int-mul n
                                                    (ilaenv 1
                                                     "DGEQRF" " " m
                                                     n -1 -1))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add n
                                            (f2cl-lib:int-mul m
                                                              (ilaenv
                                                               1
                                                               "DORGQR"
                                                               " "
                                                               m m
                                                               n
                                                               -1))))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                            (f2cl-lib:int-mul 2
                                                              n
                                                              (ilaenv
                                                               1
                                                               "DGEBRD"
```

```
                                                                  " "
                                                                  n n
                                                                  -1
                                                                  -1))))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                            (f2cl-lib:int-mul n
                                                              (ilaenv
                                                               1
                                                               "DORGBR"
                                                               "Q"
                                                               n n
                                                               n
                                                               -1))))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum bdspac)))
          (setf maxwrk
                (f2cl-lib:int-add (f2cl-lib:int-mul n n) wrkbl))
          (setf minwrk
                (max
                 (the fixnum
                      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
                 (the fixnum bdspac)))
          (setf maxwrk
                (max (the fixnum maxwrk)
                     (the fixnum minwrk))))
         ((and wntua wntvo)
          (setf wrkbl
                (f2cl-lib:int-add n
                                  (f2cl-lib:int-mul n
                                                    (ilaenv 1
                                                     "DGEQRF" " " m
                                                     n -1 -1))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add n
                                            (f2cl-lib:int-mul m
                                                              (ilaenv
                                                               1
                                                               "DORGQR"
                                                               " "
                                                               m m
```

```
                                                                n
                                                                -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                  (f2cl-lib:int-mul 2
                                                                    n
                                                                    (ilaenv
                                                                     1
                                                                     "DGEBRD"
                                                                     " "
                                                                     n n
                                                                     -1
                                                                     -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                  (f2cl-lib:int-mul n
                                                                    (ilaenv
                                                                     1
                                                                     "DORGBR"
                                                                     "Q"
                                                                     n n
                                                                     n
                                                                     -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                  (f2cl-lib:int-mul
                                                   (f2cl-lib:int-sub n 1)
                                                   (ilaenv 1 "DORGBR" "P"
                                                    n n n -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum bdspac)))
                (setf maxwrk
                      (f2cl-lib:int-add (f2cl-lib:int-mul 2 n n) wrkbl))
                (setf minwrk
                      (max
                       (the fixnum
                            (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
                       (the fixnum bdspac)))
                (setf maxwrk
```

```
          (max (the fixnum maxwrk)
               (the fixnum minwrk))))
        ((and wntua wntvas)
         (setf wrkbl
          (f2cl-lib:int-add n
                            (f2cl-lib:int-mul n
                                              (ilaenv 1
                                               "DGEQRF" " " m
                                               n -1 -1))))
         (setf wrkbl
          (max (the fixnum wrkbl)
               (the fixnum
                    (f2cl-lib:int-add n
                                      (f2cl-lib:int-mul m
                                                        (ilaenv
                                                         1
                                                         "DORGQR"
                                                         " "
                                                         m m
                                                         n
                                                         -1))))))
         (setf wrkbl
          (max (the fixnum wrkbl)
               (the fixnum
                    (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                      (f2cl-lib:int-mul 2
                                                        n
                                                        (ilaenv
                                                         1
                                                         "DGEBRD"
                                                         " "
                                                         n n
                                                         -1
                                                         -1))))))
         (setf wrkbl
          (max (the fixnum wrkbl)
               (the fixnum
                    (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                      (f2cl-lib:int-mul n
                                                        (ilaenv
                                                         1
                                                         "DORGBR"
                                                         "Q"
                                                         n n
                                                         n
                                                         -1))))))
```

```
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                            (f2cl-lib:int-mul
                                             (f2cl-lib:int-sub n 1)
                                             (ilaenv 1 "DORGBR" "P"
                                              n n n -1))))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum bdspac)))
          (setf maxwrk
                (f2cl-lib:int-add (f2cl-lib:int-mul n n) wrkbl))
          (setf minwrk
                (max
                 (the fixnum
                      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
                 (the fixnum bdspac)))
          (setf maxwrk
                (max (the fixnum maxwrk)
                     (the fixnum minwrk))))))
        (t
         (setf maxwrk
               (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                 (f2cl-lib:int-mul
                                  (f2cl-lib:int-add m n)
                                  (ilaenv 1 "DGEBRD" " " m n -1 -1))))
         (if (or wntus wntuo)
             (setf maxwrk
                   (max (the fixnum maxwrk)
                        (the fixnum
                             (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                               (f2cl-lib:int-mul n
                                                                 (ilaenv
                                                                  1
                                                                  "DORGBR"
                                                                  "Q"
                                                                  m n
                                                                  n
                                                                  -1)))))))
         (if wntua
             (setf maxwrk
                   (max (the fixnum maxwrk)
                        (the fixnum
                             (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                               (f2cl-lib:int-mul m
```

```
                                                                    (ilaenv
                                                                     1
                                                                     "DORGBR"
                                                                     "Q"
                                                                     m m
                                                                     n
                                                                     -1)))))))
            (if (not wntvn)
                (setf maxwrk
                        (max (the fixnum maxwrk)
                             (the fixnum
                                  (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                                    (f2cl-lib:int-mul
                                                     (f2cl-lib:int-sub n
                                                                       1)
                                                     (ilaenv 1 "DORGBR"
                                                      "P" n n n -1)))))))
            (setf maxwrk
                    (max (the fixnum maxwrk)
                         (the fixnum bdspac)))
            (setf minwrk
                    (max
                     (the fixnum
                          (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
                     (the fixnum bdspac)))
            (setf maxwrk
                    (max (the fixnum maxwrk)
                         (the fixnum minwrk))))))
          (t
           (setf bdspac (f2cl-lib:int-mul 5 m))
           (cond
             ((>= n mnthr)
              (cond
                (wntvn
                 (setf maxwrk
                         (f2cl-lib:int-add m
                                           (f2cl-lib:int-mul m
                                                             (ilaenv 1
                                                              "DGELQF" " " m
                                                              n -1 -1))))
                 (setf maxwrk
                         (max (the fixnum maxwrk)
                              (the fixnum
                                   (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                     (f2cl-lib:int-mul 2
                                                                       m
```

```
                                                                  (ilaenv
                                                                   1
                                                                   "DGEBRD"
                                                                   " "
                                                                   m m
                                                                   -1
                                                                   -1))))))
          (if (or wntuo wntuas)
              (setf maxwrk
                      (max (the fixnum maxwrk)
                           (the fixnum
                                (f2cl-lib:int-add
                                 (f2cl-lib:int-mul 3 m)
                                 (f2cl-lib:int-mul m
                                                   (ilaenv 1 "DORGBR"
                                                    "Q" m m m
                                                    -1)))))))
          (setf maxwrk
                  (max (the fixnum maxwrk)
                       (the fixnum bdspac)))
          (setf minwrk
                  (max (the fixnum (f2cl-lib:int-mul 4 m))
                       (the fixnum bdspac)))
          (setf maxwrk
                  (max (the fixnum maxwrk)
                       (the fixnum minwrk))))
         ((and wntvo wntun)
          (setf wrkbl
                  (f2cl-lib:int-add m
                                    (f2cl-lib:int-mul m
                                                      (ilaenv 1
                                                       "DGELQF" " " m
                                                       n -1 -1))))
          (setf wrkbl
                  (max (the fixnum wrkbl)
                       (the fixnum
                            (f2cl-lib:int-add m
                                              (f2cl-lib:int-mul m
                                                                (ilaenv
                                                                 1
                                                                 "DORGLQ"
                                                                 " "
                                                                 m n
                                                                 m
                                                                 -1))))))
          (setf wrkbl
```

```
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                            (f2cl-lib:int-mul 2
                                                              m
                                                              (ilaenv
                                                               1
                                                               "DGEBRD"
                                                               " "
                                                               m m
                                                               -1
                                                               -1))))))
               (setf wrkbl
                     (max (the fixnum wrkbl)
                          (the fixnum
                               (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                 (f2cl-lib:int-mul
                                                  (f2cl-lib:int-sub m 1)
                                                  (ilaenv 1 "DORGBR" "P"
                                                   m m m -1))))))
               (setf wrkbl
                     (max (the fixnum wrkbl)
                          (the fixnum bdspac)))
               (setf maxwrk
                     (max
                      (the fixnum
                           (f2cl-lib:int-add (f2cl-lib:int-mul m m)
                                             wrkbl))
                      (the fixnum
                           (f2cl-lib:int-add (f2cl-lib:int-mul m m)
                                             (f2cl-lib:int-mul m n)
                                             m))))
               (setf minwrk
                     (max
                      (the fixnum
                           (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
                      (the fixnum bdspac)))
               (setf maxwrk
                     (max (the fixnum maxwrk)
                          (the fixnum minwrk))))
              ((and wntvo wntuas)
               (setf wrkbl
                     (f2cl-lib:int-add m
                                       (f2cl-lib:int-mul m
                                                         (ilaenv 1
                                                          "DGELQF" " " m
```

```
                                                n -1 -1))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add m
                                                  (f2cl-lib:int-mul m
                                                                    (ilaenv
                                                                     1
                                                                     "DORGLQ"
                                                                     " "
                                                                     m n
                                                                     m
                                                                     -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                  (f2cl-lib:int-mul 2
                                                                    m
                                                                    (ilaenv
                                                                     1
                                                                     "DGEBRD"
                                                                     " "
                                                                     m m
                                                                     -1
                                                                     -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                  (f2cl-lib:int-mul
                                                   (f2cl-lib:int-sub m 1)
                                                   (ilaenv 1 "DORGBR" "P"
                                                    m m m -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                  (f2cl-lib:int-mul m
                                                                    (ilaenv
                                                                     1
                                                                     "DORGBR"
                                                                     "Q"
                                                                     m m
                                                                     m
                                                                     -1))))))
```

```
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum bdspac)))
          (setf maxwrk
                (max
                 (the fixnum
                      (f2cl-lib:int-add (f2cl-lib:int-mul m m)
                                        wrkbl))
                 (the fixnum
                      (f2cl-lib:int-add (f2cl-lib:int-mul m m)
                                        (f2cl-lib:int-mul m n)
                                        m))))
          (setf minwrk
                (max
                 (the fixnum
                      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
                 (the fixnum bdspac)))
          (setf maxwrk
                (max (the fixnum maxwrk)
                     (the fixnum minwrk))))
         ((and wntvs wntun)
          (setf wrkbl
                (f2cl-lib:int-add m
                                  (f2cl-lib:int-mul m
                                                    (ilaenv 1
                                                     "DGELQF" " " m
                                                     n -1 -1))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add m
                                            (f2cl-lib:int-mul m
                                                              (ilaenv
                                                               1
                                                               "DORGLQ"
                                                               " "
                                                               m n
                                                               m
                                                               -1))))))
          (setf wrkbl
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                            (f2cl-lib:int-mul 2
                                                              m
                                                              (ilaenv
```

```
                                                                  1
                                                                  "DGEBRD"
                                                                  " "
                                                                  m m
                                                                  -1
                                                                  -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                  (f2cl-lib:int-mul
                                                   (f2cl-lib:int-sub m 1)
                                                   (ilaenv 1 "DORGBR" "P"
                                                    m m m -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum bdspac)))
                (setf maxwrk
                      (f2cl-lib:int-add (f2cl-lib:int-mul m m) wrkbl))
                (setf minwrk
                      (max
                       (the fixnum
                            (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
                       (the fixnum bdspac)))
                (setf maxwrk
                      (max (the fixnum maxwrk)
                           (the fixnum minwrk))))
               ((and wntvs wntuo)
                (setf wrkbl
                      (f2cl-lib:int-add m
                                        (f2cl-lib:int-mul m
                                                          (ilaenv 1
                                                           "DGELQF" " " m
                                                           n -1 -1))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add m
                                                  (f2cl-lib:int-mul m
                                                                    (ilaenv
                                                                     1
                                                                     "DORGLQ"
                                                                     " "
                                                                     m n
                                                                     m
                                                                     -1))))))
```

```
(setf wrkbl
        (max (the fixnum wrkbl)
             (the fixnum
                  (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                    (f2cl-lib:int-mul 2
                                                      m
                                                      (ilaenv
                                                       1
                                                       "DGEBRD"
                                                       " "
                                                       m m
                                                       -1
                                                       -1))))))
(setf wrkbl
        (max (the fixnum wrkbl)
             (the fixnum
                  (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                    (f2cl-lib:int-mul
                                     (f2cl-lib:int-sub m 1)
                                     (ilaenv 1 "DORGBR" "P"
                                      m m m -1))))))
(setf wrkbl
        (max (the fixnum wrkbl)
             (the fixnum
                  (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                    (f2cl-lib:int-mul m
                                                      (ilaenv
                                                       1
                                                       "DORGBR"
                                                       "Q"
                                                       m m
                                                       m
                                                       -1))))))
(setf wrkbl
        (max (the fixnum wrkbl)
             (the fixnum bdspac)))
(setf maxwrk
        (f2cl-lib:int-add (f2cl-lib:int-mul 2 m m) wrkbl))
(setf minwrk
        (max
         (the fixnum
              (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
         (the fixnum bdspac)))
(setf maxwrk
        (max (the fixnum maxwrk)
             (the fixnum minwrk))))
```

```
((and wntvs wntuas)
 (setf wrkbl
       (f2cl-lib:int-add m
                         (f2cl-lib:int-mul m
                                           (ilaenv 1
                                            "DGELQF" " " m
                                            n -1 -1))))
 (setf wrkbl
       (max (the fixnum wrkbl)
            (the fixnum
                 (f2cl-lib:int-add m
                                   (f2cl-lib:int-mul m
                                                     (ilaenv
                                                      1
                                                      "DORGLQ"
                                                      " "
                                                      m n
                                                      m
                                                      -1))))))
 (setf wrkbl
       (max (the fixnum wrkbl)
            (the fixnum
                 (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                   (f2cl-lib:int-mul 2
                                                     m
                                                     (ilaenv
                                                      1
                                                      "DGEBRD"
                                                      " "
                                                      m m
                                                      -1
                                                      -1))))))
 (setf wrkbl
       (max (the fixnum wrkbl)
            (the fixnum
                 (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                   (f2cl-lib:int-mul
                                    (f2cl-lib:int-sub m 1)
                                    (ilaenv 1 "DORGBR" "P"
                                     m m m -1))))))
 (setf wrkbl
       (max (the fixnum wrkbl)
            (the fixnum
                 (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                   (f2cl-lib:int-mul m
                                                     (ilaenv
```

```
                                                                  1
                                                                  "DORGBR"
                                                                  "Q"
                                                                  m m
                                                                  m
                                                                  -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum bdspac)))
                (setf maxwrk
                      (f2cl-lib:int-add (f2cl-lib:int-mul m m) wrkbl))
                (setf minwrk
                      (max
                       (the fixnum
                            (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
                       (the fixnum bdspac)))
                (setf maxwrk
                      (max (the fixnum maxwrk)
                           (the fixnum minwrk))))
               ((and wntva wntun)
                (setf wrkbl
                      (f2cl-lib:int-add m
                                        (f2cl-lib:int-mul m
                                                          (ilaenv 1
                                                           "DGELQF" " " m
                                                           n -1 -1))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add m
                                                  (f2cl-lib:int-mul n
                                                                    (ilaenv
                                                                     1
                                                                     "DORGLQ"
                                                                     " "
                                                                     n n
                                                                     m
                                                                     -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                  (f2cl-lib:int-mul 2
                                                                    m
                                                                    (ilaenv
                                                                     1
```

```
                                                                "DGEBRD"
                                                                " "
                                                                m m
                                                                -1
                                                                -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                  (f2cl-lib:int-mul
                                                   (f2cl-lib:int-sub m 1)
                                                   (ilaenv 1 "DORGBR" "P"
                                                    m m m -1))))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum bdspac)))
                (setf maxwrk
                      (f2cl-lib:int-add (f2cl-lib:int-mul m m) wrkbl))
                (setf minwrk
                      (max
                       (the fixnum
                            (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
                       (the fixnum bdspac)))
                (setf maxwrk
                      (max (the fixnum maxwrk)
                           (the fixnum minwrk))))
               ((and wntva wntuo)
                (setf wrkbl
                      (f2cl-lib:int-add m
                                        (f2cl-lib:int-mul m
                                                          (ilaenv 1
                                                           "DGELQF" " " m
                                                           n -1 -1))))
                (setf wrkbl
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add m
                                                  (f2cl-lib:int-mul n
                                                                    (ilaenv
                                                                     1
                                                                     "DORGLQ"
                                                                     " "
                                                                     n n
                                                                     m
                                                                     -1))))))
                (setf wrkbl
```

```
                (max (the fixnum wrkbl)
                     (the fixnum
                          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                            (f2cl-lib:int-mul 2
                                                              m
                                                              (ilaenv
                                                               1
                                                               "DGEBRD"
                                                               " "
                                                               m m
                                                               -1
                                                               -1))))))
               (setf wrkbl
                     (max (the fixnum wrkbl)
                          (the fixnum
                               (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                 (f2cl-lib:int-mul
                                                  (f2cl-lib:int-sub m 1)
                                                  (ilaenv 1 "DORGBR" "P"
                                                   m m m -1))))))
               (setf wrkbl
                     (max (the fixnum wrkbl)
                          (the fixnum
                               (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                 (f2cl-lib:int-mul m
                                                                   (ilaenv
                                                                    1
                                                                    "DORGBR"
                                                                    "Q"
                                                                    m m
                                                                    m
                                                                    -1))))))
               (setf wrkbl
                     (max (the fixnum wrkbl)
                          (the fixnum bdspac)))
               (setf maxwrk
                     (f2cl-lib:int-add (f2cl-lib:int-mul 2 m m) wrkbl))
               (setf minwrk
                     (max
                      (the fixnum
                           (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
                      (the fixnum bdspac)))
               (setf maxwrk
                     (max (the fixnum maxwrk)
                          (the fixnum minwrk))))
              ((and wntva wntuas)
```

```
(setf wrkbl
        (f2cl-lib:int-add m
                          (f2cl-lib:int-mul m
                                            (ilaenv 1
                                             "DGELQF" " " m
                                             n -1 -1))))
(setf wrkbl
        (max (the fixnum wrkbl)
             (the fixnum
                  (f2cl-lib:int-add m
                                    (f2cl-lib:int-mul n
                                                      (ilaenv
                                                       1
                                                       "DORGLQ"
                                                       " "
                                                       n n
                                                       m
                                                       -1))))))
(setf wrkbl
        (max (the fixnum wrkbl)
             (the fixnum
                  (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                    (f2cl-lib:int-mul 2
                                                      m
                                                      (ilaenv
                                                       1
                                                       "DGEBRD"
                                                       " "
                                                       m m
                                                       -1
                                                       -1))))))
(setf wrkbl
        (max (the fixnum wrkbl)
             (the fixnum
                  (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                    (f2cl-lib:int-mul
                                     (f2cl-lib:int-sub m 1)
                                     (ilaenv 1 "DORGBR" "P"
                                      m m m -1))))))
(setf wrkbl
        (max (the fixnum wrkbl)
             (the fixnum
                  (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                    (f2cl-lib:int-mul m
                                                      (ilaenv
                                                       1
```

```
                                                                    "DORGBR"
                                                                    "Q"
                                                                    m m
                                                                    m
                                                                    -1))))))
                    (setf wrkbl
                            (max (the fixnum wrkbl)
                                 (the fixnum bdspac)))
                    (setf maxwrk
                            (f2cl-lib:int-add (f2cl-lib:int-mul m m) wrkbl))
                    (setf minwrk
                            (max
                             (the fixnum
                                  (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
                             (the fixnum bdspac)))
                    (setf maxwrk
                            (max (the fixnum maxwrk)
                                 (the fixnum minwrk))))))
                (t
                 (setf maxwrk
                        (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                          (f2cl-lib:int-mul
                                           (f2cl-lib:int-add m n)
                                           (ilaenv 1 "DGEBRD" " " m n -1 -1))))
                 (if (or wntvs wntvo)
                     (setf maxwrk
                             (max (the fixnum maxwrk)
                                  (the fixnum
                                       (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                         (f2cl-lib:int-mul m
                                                                           (ilaenv
                                                                            1
                                                                            "DORGBR"
                                                                            "P"
                                                                            m n
                                                                            m
                                                                            -1)))))))
                 (if wntva
                     (setf maxwrk
                             (max (the fixnum maxwrk)
                                  (the fixnum
                                       (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                         (f2cl-lib:int-mul n
                                                                           (ilaenv
                                                                            1
                                                                            "DORGBR"
```

```
                                                                    "P"
                                                                    n n
                                                                    m
                                                                    -1)))))))
          (if (not wntun)
              (setf maxwrk
                      (max (the fixnum maxwrk)
                           (the fixnum
                                (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                                  (f2cl-lib:int-mul
                                                   (f2cl-lib:int-sub m
                                                                     1)
                                                   (ilaenv 1 "DORGBR"
                                                    "Q" m m m -1)))))))
          (setf maxwrk
                  (max (the fixnum maxwrk)
                       (the fixnum bdspac)))
          (setf minwrk
                  (max
                   (the fixnum
                        (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
                   (the fixnum bdspac)))
          (setf maxwrk
                  (max (the fixnum maxwrk)
                       (the fixnum minwrk)))))))
   (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
           (coerce (the fixnum maxwrk) 'double-float))))
(cond
  ((and (< lwork minwrk) (not lquery))
   (setf info -13)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DGESVD" (f2cl-lib:int-sub info))
   (go end_label))
  (lquery
   (go end_label)))
(cond
  ((or (= m 0) (= n 0))
   (if (>= lwork 1)
       (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) one))
   (go end_label)))
(setf eps (dlamch "P"))
(setf smlnum (/ (f2cl-lib:fsqrt (dlamch "S")) eps))
(setf bignum (/ one smlnum))
```

```
(setf anrm (dlange "M" m n a lda dum))
(setf iscl 0)
(cond
  ((and (> anrm zero) (< anrm smlnum))
   (setf iscl 1)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
       (dlascl "G" 0 0 anrm smlnum m n a lda ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                      var-8))
     (setf ierr var-9)))
  ((> anrm bignum)
   (setf iscl 1)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
       (dlascl "G" 0 0 anrm bignum m n a lda ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                      var-8))
     (setf ierr var-9))))
(cond
  ((>= m n)
   (cond
     ((>= m mnthr)
      (cond
        (wntun
         (setf itau 1)
         (setf iwork (f2cl-lib:int-add itau n))
         (multiple-value-bind
               (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
             (dgeqrf m n a lda
              (f2cl-lib:array-slice work double-float (itau) ((1 *)))
              (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
              (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
           (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
           (setf ierr var-7))
         (dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1) zero
          zero
          (f2cl-lib:array-slice a double-float (2 1) ((1 lda) (1 *)))
          lda)
         (setf ie 1)
         (setf itauq (f2cl-lib:int-add ie n))
         (setf itaup (f2cl-lib:int-add itauq n))
         (setf iwork (f2cl-lib:int-add itaup n))
         (multiple-value-bind
               (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                var-9 var-10)
```

```
    (dgebrd n n a lda s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
     (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
     (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9))
   (setf ierr var-10))
 (setf ncvt 0)
 (cond
   ((or wntvo wntvas)
    (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
           var-8 var-9)
        (dorgbr "P" n n n a lda
         (f2cl-lib:array-slice work
                               double-float
                               (itaup)
                               ((1 *)))
         (f2cl-lib:array-slice work
                               double-float
                               (iwork)
                               ((1 *)))
         (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
         ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                       var-6 var-7 var-8))
      (setf ierr var-9))
    (setf ncvt n)))
 (setf iwork (f2cl-lib:int-add ie n))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13 var-14)
     (dbdsqr "U" n ncvt 0 0 s
      (f2cl-lib:array-slice work double-float (ie) ((1 *))) a
      lda dum 1 dum 1
      (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
      info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12
                    var-13))
   (setf info var-14))
 (if wntvas (dlacpy "F" n n a lda vt ldvt)))
((and wntuo wntvn)
 (cond
```

```
((>= lwork
     (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                       (max
                        (the fixnum
                             (f2cl-lib:int-mul 4 n))
                        (the fixnum bdspac))))
 (setf ir 1)
 (cond
   ((>= lwork
        (f2cl-lib:int-add
         (max (the fixnum wrkbl)
              (the fixnum
                   (f2cl-lib:int-add
                    (f2cl-lib:int-mul lda n)
                    n)))
         (f2cl-lib:int-mul lda n)))
    (setf ldwrku lda)
    (setf ldwrkr lda))
   ((>= lwork
        (f2cl-lib:int-add
         (max (the fixnum wrkbl)
              (the fixnum
                   (f2cl-lib:int-add
                    (f2cl-lib:int-mul lda n)
                    n)))
         (f2cl-lib:int-mul n n)))
    (setf ldwrku lda)
    (setf ldwrkr n))
   (t
    (setf ldwrku
            (the fixnum
                 (truncate (- lwork (* n n) n) n)))
    (setf ldwrkr n)))
 (setf itau
         (f2cl-lib:int-add ir (f2cl-lib:int-mul ldwrkr n)))
 (setf iwork (f2cl-lib:int-add itau n))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
     (dgeqrf m n a lda
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
```

```
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "U" n n a lda
 (f2cl-lib:array-slice work double-float (ir) ((1 *)))
 ldwrkr)
(dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1)
 zero zero
 (f2cl-lib:array-slice work
                       double-float
                       ((+ ir 1))
                       ((1 *)))
 ldwrkr)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorgqr m n n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd n n
     (f2cl-lib:array-slice work double-float (ir) ((1 *)))
     ldwrkr s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
```

```
                                (itauq)
                                ((1 *)))
      (f2cl-lib:array-slice work
                                double-float
                                (itaup)
                                ((1 *)))
      (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9))
   (setf ierr var-10))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9)
     (dorgbr "Q" n n n
      (f2cl-lib:array-slice work double-float (ir) ((1 *)))
      ldwrkr
      (f2cl-lib:array-slice work
                                double-float
                                (itauq)
                                ((1 *)))
      (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8))
   (setf ierr var-9))
 (setf iwork (f2cl-lib:int-add ie n))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10 var-11 var-12 var-13 var-14)
     (dbdsqr "U" n 0 n 0 s
      (f2cl-lib:array-slice work double-float (ie) ((1 *)))
      dum 1
      (f2cl-lib:array-slice work double-float (ir) ((1 *)))
      ldwrkr dum 1
      (f2cl-lib:array-slice work
                                double-float
                                (iwork)
```

```
                              ((1 *)))
       info)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                      var-6 var-7 var-8 var-9 var-10 var-11
                      var-12 var-13))
     (setf info var-14))
   (setf iu (f2cl-lib:int-add ie n))
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i ldwrku))
                 ((> i m) nil)
     (tagbody
       (setf chunk
               (min
                (the fixnum
                     (f2cl-lib:int-add (f2cl-lib:int-sub m i)
                                       1))
                (the fixnum ldwrku)))
       (dgemm "N" "N" chunk n n one
        (f2cl-lib:array-slice a
                              double-float
                              (i 1)
                              ((1 lda) (1 *)))
        lda
        (f2cl-lib:array-slice work double-float (ir) ((1 *)))
        ldwrkr zero
        (f2cl-lib:array-slice work double-float (iu) ((1 *)))
        ldwrku)
       (dlacpy "F" chunk n
        (f2cl-lib:array-slice work double-float (iu) ((1 *)))
        ldwrku
        (f2cl-lib:array-slice a
                              double-float
                              (i 1)
                              ((1 lda) (1 *)))
        lda))))
  (t
   (setf ie 1)
   (setf itauq (f2cl-lib:int-add ie n))
   (setf itaup (f2cl-lib:int-add itauq n))
   (setf iwork (f2cl-lib:int-add itaup n))
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
          var-8 var-9 var-10)
       (dgebrd m n a lda s
        (f2cl-lib:array-slice work double-float (ie) ((1 *)))
        (f2cl-lib:array-slice work
                              double-float
```

```
                                (itauq)
                                ((1 *)))
      (f2cl-lib:array-slice work
                                double-float
                                (itaup)
                                ((1 *)))
      (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9))
   (setf ierr var-10))
 (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
     (dorgbr "Q" m n n a lda
      (f2cl-lib:array-slice work
                                double-float
                                (itauq)
                                ((1 *)))
      (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8))
   (setf ierr var-9))
 (setf iwork (f2cl-lib:int-add ie n))
 (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13 var-14)
     (dbdsqr "U" n 0 m 0 s
      (f2cl-lib:array-slice work double-float (ie) ((1 *)))
      dum 1 a lda dum 1
      (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
      info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
```

```
                        var-12 var-13))
      (setf info var-14)))))
((and wntuo wntvas)
 (cond
   ((>= lwork
        (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                          (max
                           (the fixnum
                                (f2cl-lib:int-mul 4 n))
                           (the fixnum bdspac))))
    (setf ir 1)
    (cond
      ((>= lwork
           (f2cl-lib:int-add
            (max (the fixnum wrkbl)
                 (the fixnum
                      (f2cl-lib:int-add
                       (f2cl-lib:int-mul lda n)
                       n)))
            (f2cl-lib:int-mul lda n)))
       (setf ldwrku lda)
       (setf ldwrkr lda))
      ((>= lwork
           (f2cl-lib:int-add
            (max (the fixnum wrkbl)
                 (the fixnum
                      (f2cl-lib:int-add
                       (f2cl-lib:int-mul lda n)
                       n)))
            (f2cl-lib:int-mul n n)))
       (setf ldwrku lda)
       (setf ldwrkr n))
      (t
       (setf ldwrku
             (the fixnum
                  (truncate (- lwork (* n n) n) n)))
       (setf ldwrkr n)))
    (setf itau
          (f2cl-lib:int-add ir (f2cl-lib:int-mul ldwrkr n)))
    (setf iwork (f2cl-lib:int-add itau n))
    (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
        (dgeqrf m n a lda
         (f2cl-lib:array-slice work
                               double-float
                               (itau)
```

```
                              ((1 *)))
      (f2cl-lib:array-slice work
                              double-float
                              (iwork)
                              ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6))
   (setf ierr var-7))
 (dlacpy "U" n n a lda vt ldvt)
 (dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1)
  zero zero
  (f2cl-lib:array-slice vt
                        double-float
                        (2 1)
                        ((1 ldvt) (1 *)))
  ldvt)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8)
     (dorgqr m n n a lda
      (f2cl-lib:array-slice work
                              double-float
                              (itau)
                              ((1 *)))
      (f2cl-lib:array-slice work
                              double-float
                              (iwork)
                              ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7))
   (setf ierr var-8))
 (setf ie itau)
 (setf itauq (f2cl-lib:int-add ie n))
 (setf itaup (f2cl-lib:int-add itauq n))
 (setf iwork (f2cl-lib:int-add itaup n))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10)
     (dgebrd n n vt ldvt s
      (f2cl-lib:array-slice work double-float (ie) ((1 *)))
      (f2cl-lib:array-slice work
                              double-float
```

```
                          (itauq)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(dlacpy "L" n n vt ldvt
 (f2cl-lib:array-slice work double-float (ir) ((1 *)))
 ldwrkr)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "Q" n n n
     (f2cl-lib:array-slice work double-float (ir) ((1 *)))
     ldwrkr
     (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "P" n n n vt ldvt
     (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))
     (f2cl-lib:array-slice work
```

```
                              double-float
                              (iwork)
                              ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                     var-6 var-7 var-8))
    (setf ierr var-9))
  (setf iwork (f2cl-lib:int-add ie n))
  (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10 var-11 var-12 var-13 var-14)
      (dbdsqr "U" n n n 0 s
       (f2cl-lib:array-slice work double-float (ie) ((1 *)))
       vt ldvt
       (f2cl-lib:array-slice work double-float (ir) ((1 *)))
       ldwrkr dum 1
       (f2cl-lib:array-slice work
                              double-float
                              (iwork)
                              ((1 *)))
       info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                     var-6 var-7 var-8 var-9 var-10 var-11
                     var-12 var-13))
    (setf info var-14))
  (setf iu (f2cl-lib:int-add ie n))
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i ldwrku))
                ((> i m) nil)
    (tagbody
      (setf chunk
              (min
               (the fixnum
                    (f2cl-lib:int-add (f2cl-lib:int-sub m i)
                                      1))
               (the fixnum ldwrku)))
      (dgemm "N" "N" chunk n n one
       (f2cl-lib:array-slice a
                              double-float
                              (i 1)
                              ((1 lda) (1 *)))
       lda
       (f2cl-lib:array-slice work double-float (ir) ((1 *)))
       ldwrkr zero
       (f2cl-lib:array-slice work double-float (iu) ((1 *)))
       ldwrku)
```

```
     (dlacpy "F" chunk n
      (f2cl-lib:array-slice work double-float (iu) ((1 *)))
      ldwrku
      (f2cl-lib:array-slice a
                            double-float
                            (i 1)
                            ((1 lda) (1 *)))
      lda))))
(t
 (setf itau 1)
 (setf iwork (f2cl-lib:int-add itau n))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
     (dgeqrf m n a lda
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6))
   (setf ierr var-7))
 (dlacpy "U" n n a lda vt ldvt)
 (dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1)
  zero zero
  (f2cl-lib:array-slice vt
                        double-float
                        (2 1)
                        ((1 ldvt) (1 *)))
  ldvt)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8)
     (dorgqr m n n a lda
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
```

```
                              ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd n n vt ldvt s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "R" "N" m n n vt ldvt
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     a lda
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
```

```
          ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8 var-9 var-10 var-11
                         var-12))
        (setf ierr var-13))
      (multiple-value-bind
           (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
            var-8 var-9)
          (dorgbr "P" n n n vt ldvt
           (f2cl-lib:array-slice work
                                 double-float
                                 (itaup)
                                 ((1 *)))
           (f2cl-lib:array-slice work
                                 double-float
                                 (iwork)
                                 ((1 *)))
           (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
           ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8))
        (setf ierr var-9))
      (setf iwork (f2cl-lib:int-add ie n))
      (multiple-value-bind
           (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
            var-8 var-9 var-10 var-11 var-12 var-13 var-14)
          (dbdsqr "U" n n m 0 s
           (f2cl-lib:array-slice work double-float (ie) ((1 *)))
           vt ldvt a lda dum 1
           (f2cl-lib:array-slice work
                                 double-float
                                 (iwork)
                                 ((1 *)))
           info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8 var-9 var-10 var-11
                         var-12 var-13))
        (setf info var-14)))))
  (wntus
   (cond
     (wntvn
      (cond
        ((>= lwork
             (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                               (max
                                (the fixnum
```

```
                                (f2cl-lib:int-mul 4 n))
                           (the fixnum bdspac))))
(setf ir 1)
(cond
  ((>= lwork
       (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda n)))
   (setf ldwrkr lda))
  (t
   (setf ldwrkr n)))
(setf itau
        (f2cl-lib:int-add ir
                          (f2cl-lib:int-mul ldwrkr n)))
(setf iwork (f2cl-lib:int-add itau n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgeqrf m n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "U" n n a lda
 (f2cl-lib:array-slice work double-float (ir) ((1 *)))
 ldwrkr)
(dlaset "L" (f2cl-lib:int-sub n 1)
 (f2cl-lib:int-sub n 1) zero zero
 (f2cl-lib:array-slice work
                       double-float
                       ((+ ir 1))
                       ((1 *)))
 ldwrkr)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorgqr m n n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
```

```
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd n n
     (f2cl-lib:array-slice work
                          double-float
                          (ir)
                          ((1 *)))
     ldwrkr s
     (f2cl-lib:array-slice work
                          double-float
                          (ie)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
```

```
      var-8 var-9)
    (dorgbr "Q" n n n
     (f2cl-lib:array-slice work
                           double-float
                           (ir)
                           ((1 *)))
     ldwrkr
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13 var-14)
    (dbdsqr "U" n 0 n 0 s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     dum 1
     (f2cl-lib:array-slice work
                           double-float
                           (ir)
                           ((1 *)))
     ldwrkr dum 1
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12 var-13))
  (setf info var-14))
(dgemm "N" "N" m n n one a lda
 (f2cl-lib:array-slice work double-float (ir) ((1 *)))
```

```
 ldwrkr zero u ldu))
(t
 (setf itau 1)
 (setf iwork (f2cl-lib:int-add itau n))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
     (dgeqrf m n a lda
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6))
   (setf ierr var-7))
 (dlacpy "L" m n a lda u ldu)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8)
     (dorgqr m n n u ldu
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7))
   (setf ierr var-8))
 (setf ie itau)
 (setf itauq (f2cl-lib:int-add ie n))
 (setf itaup (f2cl-lib:int-add itauq n))
 (setf iwork (f2cl-lib:int-add itaup n))
 (dlaset "L" (f2cl-lib:int-sub n 1)
  (f2cl-lib:int-sub n 1) zero zero
  (f2cl-lib:array-slice a
                        double-float
```

```
                        (2 1)
                        ((1 lda) (1 *)))
 lda)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd n n a lda s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "R" "N" m n n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     u ldu
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12))
```

```
      (setf ierr var-13))
    (setf iwork (f2cl-lib:int-add ie n))
    (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
           var-8 var-9 var-10 var-11 var-12 var-13 var-14)
        (dbdsqr "U" n 0 m 0 s
         (f2cl-lib:array-slice work
                               double-float
                               (ie)
                               ((1 *)))
         dum 1 u ldu dum 1
         (f2cl-lib:array-slice work
                               double-float
                               (iwork)
                               ((1 *)))
         info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                       var-6 var-7 var-8 var-9 var-10 var-11
                       var-12 var-13))
      (setf info var-14)))))
(wntvo
 (cond
   ((>= lwork
        (f2cl-lib:int-add (f2cl-lib:int-mul 2 n n)
                          (max
                           (the fixnum
                                (f2cl-lib:int-mul 4 n))
                           (the fixnum bdspac))))
    (setf iu 1)
    (cond
      ((>= lwork
           (f2cl-lib:int-add wrkbl
                             (f2cl-lib:int-mul 2 lda n)))
       (setf ldwrku lda)
       (setf ir
             (f2cl-lib:int-add iu
                               (f2cl-lib:int-mul ldwrku
                                                 n)))
       (setf ldwrkr lda))
      ((>= lwork
           (f2cl-lib:int-add wrkbl
                             (f2cl-lib:int-mul
                              (f2cl-lib:int-add lda n)
                              n)))
       (setf ldwrku lda)
       (setf ir
```

```
          (f2cl-lib:int-add iu
                            (f2cl-lib:int-mul ldwrku
                                              n)))
   (setf ldwrkr n))
  (t
   (setf ldwrku n)
   (setf ir
          (f2cl-lib:int-add iu
                            (f2cl-lib:int-mul ldwrku
                                              n)))
   (setf ldwrkr n)))
(setf itau
       (f2cl-lib:int-add ir
                         (f2cl-lib:int-mul ldwrkr n)))
(setf iwork (f2cl-lib:int-add itau n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgeqrf m n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "U" n n a lda
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku)
(dlaset "L" (f2cl-lib:int-sub n 1)
 (f2cl-lib:int-sub n 1) zero zero
 (f2cl-lib:array-slice work
                       double-float
                       ((+ iu 1))
                       ((1 *)))
 ldwrku)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorgqr m n n a lda
     (f2cl-lib:array-slice work
```

```
                              double-float
                              (itau)
                              ((1 *)))
     (f2cl-lib:array-slice work
                              double-float
                              (iwork)
                              ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd n n
     (f2cl-lib:array-slice work
                              double-float
                              (iu)
                              ((1 *)))
     ldwrku s
     (f2cl-lib:array-slice work
                              double-float
                              (ie)
                              ((1 *)))
     (f2cl-lib:array-slice work
                              double-float
                              (itauq)
                              ((1 *)))
     (f2cl-lib:array-slice work
                              double-float
                              (itaup)
                              ((1 *)))
     (f2cl-lib:array-slice work
                              double-float
                              (iwork)
                              ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
```

```
(dlacpy "U" n n
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku
 (f2cl-lib:array-slice work double-float (ir) ((1 *)))
 ldwrkr)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "Q" n n n
     (f2cl-lib:array-slice work
                           double-float
                           (iu)
                           ((1 *)))
     ldwrku
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "P" n n n
     (f2cl-lib:array-slice work
                           double-float
                           (ir)
                           ((1 *)))
     ldwrkr
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
```

```
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
 (setf iwork (f2cl-lib:int-add ie n))
 (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13 var-14)
    (dbdsqr "U" n n n 0 s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (ir)
                           ((1 *)))
     ldwrkr
     (f2cl-lib:array-slice work
                           double-float
                           (iu)
                           ((1 *)))
     ldwrku dum 1
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12 var-13))
  (setf info var-14))
 (dgemm "N" "N" m n n one a lda
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku zero u ldu)
 (dlacpy "F" n n
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr a lda))
(t
 (setf itau 1)
 (setf iwork (f2cl-lib:int-add itau n))
 (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgeqrf m n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
```

```
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "L" m n a lda u ldu)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorgqr m n n u ldu
     (f2cl-lib:array-slice work
                          double-float
                          (itau)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(dlaset "L" (f2cl-lib:int-sub n 1)
 (f2cl-lib:int-sub n 1) zero zero
 (f2cl-lib:array-slice a
                       double-float
                       (2 1)
                       ((1 lda) (1 *)))
 lda)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd n n a lda s
     (f2cl-lib:array-slice work
                          double-float
                          (ie)
```

```
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "R" "N" m n n a lda
     (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))
     u ldu
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12))
  (setf ierr var-13))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "P" n n n a lda
     (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))
     (f2cl-lib:array-slice work
```

```
                                    double-float
                                    (iwork)
                                    ((1 *)))
                (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
                ierr)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                               var-6 var-7 var-8))
              (setf ierr var-9))
            (setf iwork (f2cl-lib:int-add ie n))
            (multiple-value-bind
                 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                  var-8 var-9 var-10 var-11 var-12 var-13 var-14)
                (dbdsqr "U" n n m 0 s
                 (f2cl-lib:array-slice work
                                    double-float
                                    (ie)
                                    ((1 *)))
                 a lda u ldu dum 1
                 (f2cl-lib:array-slice work
                                    double-float
                                    (iwork)
                                    ((1 *)))
                 info)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                               var-6 var-7 var-8 var-9 var-10 var-11
                               var-12 var-13))
              (setf info var-14)))))
        (wntvas
         (cond
           ((>= lwork
                (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                                  (max
                                   (the fixnum
                                        (f2cl-lib:int-mul 4 n))
                                   (the fixnum bdspac))))
            (setf iu 1)
            (cond
              ((>= lwork
                   (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda n)))
               (setf ldwrku lda))
              (t
               (setf ldwrku n)))
            (setf itau
                  (f2cl-lib:int-add iu
                                    (f2cl-lib:int-mul ldwrku n)))
            (setf iwork (f2cl-lib:int-add itau n))
```

```
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgeqrf m n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "U" n n a lda
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku)
(dlaset "L" (f2cl-lib:int-sub n 1)
 (f2cl-lib:int-sub n 1) zero zero
 (f2cl-lib:array-slice work
                       double-float
                       ((+ iu 1))
                       ((1 *)))
 ldwrku)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorgqr m n n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
```

```
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd n n
     (f2cl-lib:array-slice work
                           double-float
                           (iu)
                           ((1 *)))
     ldwrku s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(dlacpy "U" n n
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku vt ldvt)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "Q" n n n
     (f2cl-lib:array-slice work
                           double-float
                           (iu)
                           ((1 *)))
     ldwrku
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
```

```
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "P" n n n vt ldvt
     (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13 var-14)
    (dbdsqr "U" n n n 0 s
     (f2cl-lib:array-slice work
                          double-float
                          (ie)
                          ((1 *)))
     vt ldvt
     (f2cl-lib:array-slice work
                          double-float
                          (iu)
                          ((1 *)))
     ldwrku dum 1
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
```

```
      info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12 var-13))
   (setf info var-14))
 (dgemm "N" "N" m n n one a lda
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku zero u ldu))
(t
 (setf itau 1)
 (setf iwork (f2cl-lib:int-add itau n))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
     (dgeqrf m n a lda
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6))
   (setf ierr var-7))
 (dlacpy "L" m n a lda u ldu)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8)
     (dorgqr m n n u ldu
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7))
   (setf ierr var-8))
 (dlacpy "U" n n a lda vt ldvt)
```

```
(dlaset "L" (f2cl-lib:int-sub n 1)
 (f2cl-lib:int-sub n 1) zero zero
 (f2cl-lib:array-slice vt
                       double-float
                       (2 1)
                       ((1 ldvt) (1 *)))
 ldvt)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd n n vt ldvt s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "R" "N" m n n vt ldvt
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     u ldu
     (f2cl-lib:array-slice work
```

```
                                    double-float
                                    (iwork)
                                    ((1 *)))
                (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
                ierr)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                               var-6 var-7 var-8 var-9 var-10 var-11
                               var-12))
              (setf ierr var-13))
            (multiple-value-bind
                 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                  var-8 var-9)
                (dorgbr "P" n n n vt ldvt
                 (f2cl-lib:array-slice work
                                    double-float
                                    (itaup)
                                    ((1 *)))
                 (f2cl-lib:array-slice work
                                    double-float
                                    (iwork)
                                    ((1 *)))
                 (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
                 ierr)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                               var-6 var-7 var-8))
              (setf ierr var-9))
            (setf iwork (f2cl-lib:int-add ie n))
            (multiple-value-bind
                 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                  var-8 var-9 var-10 var-11 var-12 var-13 var-14)
                (dbdsqr "U" n n m 0 s
                 (f2cl-lib:array-slice work
                                    double-float
                                    (ie)
                                    ((1 *)))
                 vt ldvt u ldu dum 1
                 (f2cl-lib:array-slice work
                                    double-float
                                    (iwork)
                                    ((1 *)))
                 info)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                               var-6 var-7 var-8 var-9 var-10 var-11
                               var-12 var-13))
              (setf info var-14)))))))
      (wntua
```

```
(cond
  (wntvn
   (cond
     ((>= lwork
          (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                            (max
                             (the fixnum
                                  (f2cl-lib:int-add n m))
                             (the fixnum
                                  (f2cl-lib:int-mul 4 n))
                             (the fixnum bdspac))))
      (setf ir 1)
      (cond
        ((>= lwork
             (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda n)))
         (setf ldwrkr lda))
        (t
         (setf ldwrkr n)))
      (setf itau
            (f2cl-lib:int-add ir
                              (f2cl-lib:int-mul ldwrkr n)))
      (setf iwork (f2cl-lib:int-add itau n))
      (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
          (dgeqrf m n a lda
           (f2cl-lib:array-slice work
                                 double-float
                                 (itau)
                                 ((1 *)))
           (f2cl-lib:array-slice work
                                 double-float
                                 (iwork)
                                 ((1 *)))
           (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
           ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6))
        (setf ierr var-7))
      (dlacpy "L" m n a lda u ldu)
      (dlacpy "U" n n a lda
       (f2cl-lib:array-slice work double-float (ir) ((1 *)))
       ldwrkr)
      (dlaset "L" (f2cl-lib:int-sub n 1)
       (f2cl-lib:int-sub n 1) zero zero
       (f2cl-lib:array-slice work
                             double-float
```

```
                        ((+ ir 1))
                        ((1 *)))
 ldwrkr)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorgqr m m n u ldu
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd n n
     (f2cl-lib:array-slice work
                           double-float
                           (ir)
                           ((1 *)))
     ldwrkr s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
```

```
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "Q" n n n
     (f2cl-lib:array-slice work
                          double-float
                          (ir)
                          ((1 *)))
     ldwrkr
     (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13 var-14)
    (dbdsqr "U" n 0 n 0 s
     (f2cl-lib:array-slice work
                          double-float
                          (ie)
                          ((1 *)))
     dum 1
     (f2cl-lib:array-slice work
                          double-float
                          (ir)
                          ((1 *)))
     ldwrkr dum 1
     (f2cl-lib:array-slice work
```

```
                              double-float
                              (iwork)
                              ((1 *)))
      info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12 var-13))
   (setf info var-14))
 (dgemm "N" "N" m n n one u ldu
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr zero a lda)
 (dlacpy "F" m n a lda u ldu))
(t
 (setf itau 1)
 (setf iwork (f2cl-lib:int-add itau n))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
     (dgeqrf m n a lda
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6))
   (setf ierr var-7))
 (dlacpy "L" m n a lda u ldu)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8)
     (dorgqr m m n u ldu
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
```

```
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(dlaset "L" (f2cl-lib:int-sub n 1)
 (f2cl-lib:int-sub n 1) zero zero
 (f2cl-lib:array-slice a
                       double-float
                       (2 1)
                       ((1 lda) (1 *)))
 lda)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd n n a lda s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "R" "N" m n n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
```

```
                                    ((1 *)))
                u ldu
                (f2cl-lib:array-slice work
                                    double-float
                                    (iwork)
                                    ((1 *)))
                (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
                ierr)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                               var-6 var-7 var-8 var-9 var-10 var-11
                               var-12))
              (setf ierr var-13))
            (setf iwork (f2cl-lib:int-add ie n))
            (multiple-value-bind
                 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                  var-8 var-9 var-10 var-11 var-12 var-13 var-14)
                (dbdsqr "U" n 0 m 0 s
                 (f2cl-lib:array-slice work
                                    double-float
                                    (ie)
                                    ((1 *)))
                 dum 1 u ldu dum 1
                 (f2cl-lib:array-slice work
                                    double-float
                                    (iwork)
                                    ((1 *)))
                 info)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                               var-6 var-7 var-8 var-9 var-10 var-11
                               var-12 var-13))
              (setf info var-14)))))
        (wntvo
         (cond
           ((>= lwork
                (f2cl-lib:int-add (f2cl-lib:int-mul 2 n n)
                                  (max
                                   (the fixnum
                                        (f2cl-lib:int-add n m))
                                   (the fixnum
                                        (f2cl-lib:int-mul 4 n))
                                   (the fixnum bdspac))))
            (setf iu 1)
            (cond
              ((>= lwork
                   (f2cl-lib:int-add wrkbl
                                     (f2cl-lib:int-mul 2 lda n)))
```

```
   (setf ldwrku lda)
   (setf ir
           (f2cl-lib:int-add iu
                             (f2cl-lib:int-mul ldwrku
                                               n)))
   (setf ldwrkr lda))
  ((>= lwork
       (f2cl-lib:int-add wrkbl
                         (f2cl-lib:int-mul
                          (f2cl-lib:int-add lda n)
                          n)))
   (setf ldwrku lda)
   (setf ir
           (f2cl-lib:int-add iu
                             (f2cl-lib:int-mul ldwrku
                                               n)))
   (setf ldwrkr n))
  (t
   (setf ldwrku n)
   (setf ir
           (f2cl-lib:int-add iu
                             (f2cl-lib:int-mul ldwrku
                                               n)))
   (setf ldwrkr n)))
(setf itau
        (f2cl-lib:int-add ir
                          (f2cl-lib:int-mul ldwrkr n)))
(setf iwork (f2cl-lib:int-add itau n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgeqrf m n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "L" m n a lda u ldu)
(multiple-value-bind
```

```
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorgqr m m n u ldu
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(dlacpy "U" n n a lda
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku)
(dlaset "L" (f2cl-lib:int-sub n 1)
 (f2cl-lib:int-sub n 1) zero zero
 (f2cl-lib:array-slice work
                       double-float
                       ((+ iu 1))
                       ((1 *)))
 ldwrku)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd n n
     (f2cl-lib:array-slice work
                           double-float
                           (iu)
                           ((1 *)))
     ldwrku s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
```

```
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(dlacpy "U" n n
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku
 (f2cl-lib:array-slice work double-float (ir) ((1 *)))
 ldwrkr)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "Q" n n n
     (f2cl-lib:array-slice work
                          double-float
                          (iu)
                          ((1 *)))
     ldwrku
     (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "P" n n n
     (f2cl-lib:array-slice work
```

```
                                double-float
                                (ir)
                                ((1 *)))
          ldwrkr
          (f2cl-lib:array-slice work
                                double-float
                                (itaup)
                                ((1 *)))
          (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
          (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
          ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8))
        (setf ierr var-9))
      (setf iwork (f2cl-lib:int-add ie n))
      (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
             var-8 var-9 var-10 var-11 var-12 var-13 var-14)
          (dbdsqr "U" n n n 0 s
          (f2cl-lib:array-slice work
                                double-float
                                (ie)
                                ((1 *)))
          (f2cl-lib:array-slice work
                                double-float
                                (ir)
                                ((1 *)))
          ldwrkr
          (f2cl-lib:array-slice work
                                double-float
                                (iu)
                                ((1 *)))
          ldwrku dum 1
          (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
          info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8 var-9 var-10 var-11
                         var-12 var-13))
        (setf info var-14))
      (dgemm "N" "N" m n n one u ldu
```

```
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku zero a lda)
 (dlacpy "F" m n a lda u ldu)
 (dlacpy "F" n n
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr a lda))
(t
 (setf itau 1)
 (setf iwork (f2cl-lib:int-add itau n))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
     (dgeqrf m n a lda
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6))
   (setf ierr var-7))
 (dlacpy "L" m n a lda u ldu)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8)
     (dorgqr m m n u ldu
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7))
   (setf ierr var-8))
 (setf ie itau)
 (setf itauq (f2cl-lib:int-add ie n))
 (setf itaup (f2cl-lib:int-add itauq n))
```

```
(setf iwork (f2cl-lib:int-add itaup n))
(dlaset "L" (f2cl-lib:int-sub n 1)
 (f2cl-lib:int-sub n 1) zero zero
 (f2cl-lib:array-slice a
                       double-float
                       (2 1)
                       ((1 lda) (1 *)))
 lda)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd n n a lda s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "R" "N" m n n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     u ldu
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
```

```
          (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
          ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8 var-9 var-10 var-11
                         var-12))
        (setf ierr var-13))
      (multiple-value-bind
           (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
            var-8 var-9)
          (dorgbr "P" n n n a lda
           (f2cl-lib:array-slice work
                                 double-float
                                 (itaup)
                                 ((1 *)))
           (f2cl-lib:array-slice work
                                 double-float
                                 (iwork)
                                 ((1 *)))
           (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
           ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8))
        (setf ierr var-9))
      (setf iwork (f2cl-lib:int-add ie n))
      (multiple-value-bind
           (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
            var-8 var-9 var-10 var-11 var-12 var-13 var-14)
          (dbdsqr "U" n n m 0 s
           (f2cl-lib:array-slice work
                                 double-float
                                 (ie)
                                 ((1 *)))
           a lda u ldu dum 1
           (f2cl-lib:array-slice work
                                 double-float
                                 (iwork)
                                 ((1 *)))
           info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8 var-9 var-10 var-11
                         var-12 var-13))
        (setf info var-14)))))
  (wntvas
   (cond
     ((>= lwork
          (f2cl-lib:int-add (f2cl-lib:int-mul n n)
```

```
                        (max
                         (the fixnum
                              (f2cl-lib:int-add n m))
                         (the fixnum
                              (f2cl-lib:int-mul 4 n))
                         (the fixnum bdspac))))
(setf iu 1)
(cond
  ((>= lwork
       (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda n)))
   (setf ldwrku lda))
  (t
   (setf ldwrku n)))
(setf itau
        (f2cl-lib:int-add iu
                          (f2cl-lib:int-mul ldwrku n)))
(setf iwork (f2cl-lib:int-add itau n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgeqrf m n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "L" m n a lda u ldu)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorgqr m m n u ldu
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
```

```
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(dlacpy "U" n n a lda
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku)
(dlaset "L" (f2cl-lib:int-sub n 1)
 (f2cl-lib:int-sub n 1) zero zero
 (f2cl-lib:array-slice work
                       double-float
                       ((+ iu 1))
                       ((1 *)))
 ldwrku)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd n n
     (f2cl-lib:array-slice work
                           double-float
                           (iu)
                           ((1 *)))
     ldwrku s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
```

```
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(dlacpy "U" n n
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku vt ldvt)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "Q" n n n
     (f2cl-lib:array-slice work
                           double-float
                           (iu)
                           ((1 *)))
     ldwrku
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "P" n n n vt ldvt
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
```

```
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10 var-11 var-12 var-13 var-14)
     (dbdsqr "U" n n n 0 s
      (f2cl-lib:array-slice work
                            double-float
                            (ie)
                            ((1 *)))
      vt ldvt
      (f2cl-lib:array-slice work
                            double-float
                            (iu)
                            ((1 *)))
      ldwrku dum 1
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12 var-13))
   (setf info var-14))
 (dgemm "N" "N" m n n one u ldu
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku zero a lda)
 (dlacpy "F" m n a lda u ldu))
(t
 (setf itau 1)
 (setf iwork (f2cl-lib:int-add itau n))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
     (dgeqrf m n a lda
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6))
   (setf ierr var-7))
```

```
(dlacpy "L" m n a lda u ldu)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorgqr m m n u ldu
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(dlacpy "U" n n a lda vt ldvt)
(dlaset "L" (f2cl-lib:int-sub n 1)
 (f2cl-lib:int-sub n 1) zero zero
 (f2cl-lib:array-slice vt
                       double-float
                       (2 1)
                       ((1 ldvt) (1 *)))
 ldvt)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd n n vt ldvt s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
```

```
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "R" "N" m n n vt ldvt
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     u ldu
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12))
  (setf ierr var-13))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "P" n n n vt ldvt
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
```

```
                (multiple-value-bind
                      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                       var-8 var-9 var-10 var-11 var-12 var-13 var-14)
                    (dbdsqr "U" n n m 0 s
                     (f2cl-lib:array-slice work
                                           double-float
                                           (ie)
                                           ((1 *)))
                     vt ldvt u ldu dum 1
                     (f2cl-lib:array-slice work
                                           double-float
                                           (iwork)
                                           ((1 *)))
                     info)
                  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                                   var-6 var-7 var-8 var-9 var-10 var-11
                                   var-12 var-13))
                  (setf info var-14)))))))))
          (t
           (setf ie 1)
           (setf itauq (f2cl-lib:int-add ie n))
           (setf itaup (f2cl-lib:int-add itauq n))
           (setf iwork (f2cl-lib:int-add itaup n))
           (multiple-value-bind
                 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                  var-9 var-10)
               (dgebrd m n a lda s
                (f2cl-lib:array-slice work double-float (ie) ((1 *)))
                (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
                (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
                (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
                (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
             (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                              var-7 var-8 var-9))
             (setf ierr var-10))
           (cond
             (wntuas
              (dlacpy "L" m n a lda u ldu)
              (if wntus (setf ncu n))
              (if wntua (setf ncu m))
              (multiple-value-bind
                    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                     var-9)
                  (dorgbr "Q" m ncu n u ldu
                   (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
                   (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
```

```
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7 var-8))
     (setf ierr var-9))))
(cond
  (wntvas
   (dlacpy "U" n n a lda vt ldvt)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
          var-9)
       (dorgbr "P" n n n vt ldvt
        (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
        (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7 var-8))
     (setf ierr var-9))))
(cond
  (wntuo
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
          var-9)
       (dorgbr "Q" m n n a lda
        (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
        (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7 var-8))
     (setf ierr var-9))))
(cond
  (wntvo
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
          var-9)
       (dorgbr "P" n n n a lda
        (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
        (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7 var-8))
     (setf ierr var-9))))
(setf iwork (f2cl-lib:int-add ie n))
(if (or wntuas wntuo) (setf nru m))
(if wntun (setf nru 0))
(if (or wntvas wntvo) (setf ncvt n))
(if wntvn (setf ncvt 0))
```

```
          (cond
            ((and (not wntuo) (not wntvo))
             (multiple-value-bind
                   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                    var-9 var-10 var-11 var-12 var-13 var-14)
                 (dbdsqr "U" n ncvt nru 0 s
                  (f2cl-lib:array-slice work double-float (ie) ((1 *))) vt
                  ldvt u ldu dum 1
                  (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
                  info)
               (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                var-7 var-8 var-9 var-10 var-11 var-12
                                var-13))
               (setf info var-14)))
            ((and (not wntuo) wntvo)
             (multiple-value-bind
                   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                    var-9 var-10 var-11 var-12 var-13 var-14)
                 (dbdsqr "U" n ncvt nru 0 s
                  (f2cl-lib:array-slice work double-float (ie) ((1 *))) a
                  lda u ldu dum 1
                  (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
                  info)
               (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                var-7 var-8 var-9 var-10 var-11 var-12
                                var-13))
               (setf info var-14)))
            (t
             (multiple-value-bind
                   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                    var-9 var-10 var-11 var-12 var-13 var-14)
                 (dbdsqr "U" n ncvt nru 0 s
                  (f2cl-lib:array-slice work double-float (ie) ((1 *))) vt
                  ldvt a lda dum 1
                  (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
                  info)
               (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                var-7 var-8 var-9 var-10 var-11 var-12
                                var-13))
               (setf info var-14)))))))
      (t
       (cond
         ((>= n mnthr)
          (cond
            (wntvn
             (setf itau 1)
```

```
(setf iwork (f2cl-lib:int-add itau m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgelqf m n a lda
     (f2cl-lib:array-slice work double-float (itau) ((1 *)))
     (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
  (setf ierr var-7))
(dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1) zero
 zero
 (f2cl-lib:array-slice a double-float (1 2) ((1 lda) (1 *)))
 lda)
(setf ie 1)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10)
    (dgebrd m m a lda s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
     (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
     (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9))
  (setf ierr var-10))
(cond
  ((or wntuo wntuas)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
          var-8 var-9)
       (dorgbr "Q" m m m a lda
        (f2cl-lib:array-slice work
                              double-float
                              (itauq)
                              ((1 *)))
        (f2cl-lib:array-slice work
                              double-float
                              (iwork)
                              ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
        ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
```

```
                        var-6 var-7 var-8))
      (setf ierr var-9))))
 (setf iwork (f2cl-lib:int-add ie m))
 (setf nru 0)
 (if (or wntuo wntuas) (setf nru m))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13 var-14)
     (dbdsqr "U" m 0 nru 0 s
      (f2cl-lib:array-slice work double-float (ie) ((1 *))) dum
      1 a lda dum 1
      (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
      info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12
                    var-13))
   (setf info var-14))
 (if wntuas (dlacpy "F" m m a lda u ldu)))
((and wntvo wntun)
 (cond
   ((>= lwork
        (f2cl-lib:int-add (f2cl-lib:int-mul m m)
                          (max
                           (the fixnum
                                (f2cl-lib:int-mul 4 m))
                           (the fixnum bdspac))))
    (setf ir 1)
    (cond
      ((>= lwork
           (f2cl-lib:int-add
            (max (the fixnum wrkbl)
                 (the fixnum
                      (f2cl-lib:int-add
                       (f2cl-lib:int-mul lda n)
                       m)))
            (f2cl-lib:int-mul lda m)))
       (setf ldwrku lda)
       (setf chunk n)
       (setf ldwrkr lda))
      ((>= lwork
           (f2cl-lib:int-add
            (max (the fixnum wrkbl)
                 (the fixnum
                      (f2cl-lib:int-add
                       (f2cl-lib:int-mul lda n)
                       m)))
```

```
        (f2cl-lib:int-mul m m)))
   (setf ldwrku lda)
   (setf chunk n)
   (setf ldwrkr m))
  (t
   (setf ldwrku m)
   (setf chunk
           (the fixnum
                (truncate (- lwork (* m m) m) m)))
   (setf ldwrkr m)))
(setf itau
        (f2cl-lib:int-add ir (f2cl-lib:int-mul ldwrkr m)))
(setf iwork (f2cl-lib:int-add itau m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgelqf m n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "L" m m a lda
 (f2cl-lib:array-slice work double-float (ir) ((1 *)))
 ldwrkr)
(dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1)
 zero zero
 (f2cl-lib:array-slice work
                       double-float
                       ((+ ir ldwrkr))
                       ((1 *)))
 ldwrkr)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorglq m n m a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
```

```
                              ((1 *)))
      (f2cl-lib:array-slice work
                              double-float
                              (iwork)
                              ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7))
   (setf ierr var-8))
 (setf ie itau)
 (setf itauq (f2cl-lib:int-add ie m))
 (setf itaup (f2cl-lib:int-add itauq m))
 (setf iwork (f2cl-lib:int-add itaup m))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10)
     (dgebrd m m
      (f2cl-lib:array-slice work double-float (ir) ((1 *)))
      ldwrkr s
      (f2cl-lib:array-slice work double-float (ie) ((1 *)))
      (f2cl-lib:array-slice work
                              double-float
                              (itauq)
                              ((1 *)))
      (f2cl-lib:array-slice work
                              double-float
                              (itaup)
                              ((1 *)))
      (f2cl-lib:array-slice work
                              double-float
                              (iwork)
                              ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9))
   (setf ierr var-10))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9)
     (dorgbr "P" m m m
      (f2cl-lib:array-slice work double-float (ir) ((1 *)))
      ldwrkr
      (f2cl-lib:array-slice work
                              double-float
```

```
                              (itaup)
                              ((1 *)))
     (f2cl-lib:array-slice work
                              double-float
                              (iwork)
                              ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13 var-14)
    (dbdsqr "U" m m 0 0 s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work double-float (ir) ((1 *)))
     ldwrkr dum 1 dum 1
     (f2cl-lib:array-slice work
                              double-float
                              (iwork)
                              ((1 *)))
     info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12 var-13))
  (setf info var-14))
(setf iu (f2cl-lib:int-add ie m))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i chunk))
              ((> i n) nil)
  (tagbody
    (setf blk
            (min
             (the fixnum
                  (f2cl-lib:int-add (f2cl-lib:int-sub n i)
                                    1))
             (the fixnum chunk)))
    (dgemm "N" "N" m blk m one
     (f2cl-lib:array-slice work double-float (ir) ((1 *)))
     ldwrkr
     (f2cl-lib:array-slice a
                              double-float
                              (1 i)
                              ((1 lda) (1 *)))
     lda zero
```

```
      (f2cl-lib:array-slice work double-float (iu) ((1 *)))
      ldwrku)
     (dlacpy "F" m blk
      (f2cl-lib:array-slice work double-float (iu) ((1 *)))
      ldwrku
      (f2cl-lib:array-slice a
                            double-float
                            (1 i)
                            ((1 lda) (1 *)))
      lda))))
(t
 (setf ie 1)
 (setf itauq (f2cl-lib:int-add ie m))
 (setf itaup (f2cl-lib:int-add itauq m))
 (setf iwork (f2cl-lib:int-add itaup m))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10)
     (dgebrd m n a lda s
      (f2cl-lib:array-slice work double-float (ie) ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (itauq)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (itaup)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9))
   (setf ierr var-10))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9)
     (dorgbr "P" m n m a lda
      (f2cl-lib:array-slice work
                            double-float
                            (itaup)
                            ((1 *)))
      (f2cl-lib:array-slice work
```

```
                                double-float
                                (iwork)
                                ((1 *)))
                   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
                   ierr)
                  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                                   var-6 var-7 var-8))
                  (setf ierr var-9))
                 (setf iwork (f2cl-lib:int-add ie m))
                 (multiple-value-bind
                       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                        var-8 var-9 var-10 var-11 var-12 var-13 var-14)
                     (dbdsqr "L" m n 0 0 s
                      (f2cl-lib:array-slice work double-float (ie) ((1 *)))
                      a lda dum 1 dum 1
                      (f2cl-lib:array-slice work
                                            double-float
                                            (iwork)
                                            ((1 *)))
                      info)
                   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                                    var-6 var-7 var-8 var-9 var-10 var-11
                                    var-12 var-13))
                   (setf info var-14)))))
          ((and wntvo wntuas)
           (cond
             ((>= lwork
                  (f2cl-lib:int-add (f2cl-lib:int-mul m m)
                                    (max
                                     (the fixnum
                                          (f2cl-lib:int-mul 4 m))
                                     (the fixnum bdspac))))
              (setf ir 1)
              (cond
                ((>= lwork
                     (f2cl-lib:int-add
                      (max (the fixnum wrkbl)
                           (the fixnum
                                (f2cl-lib:int-add
                                 (f2cl-lib:int-mul lda n)
                                 m)))
                      (f2cl-lib:int-mul lda m)))
                 (setf ldwrku lda)
                 (setf chunk n)
                 (setf ldwrkr lda))
                ((>= lwork
```

```
        (f2cl-lib:int-add
         (max (the fixnum wrkbl)
              (the fixnum
                   (f2cl-lib:int-add
                    (f2cl-lib:int-mul lda n)
                    m)))
         (f2cl-lib:int-mul m m)))
   (setf ldwrku lda)
   (setf chunk n)
   (setf ldwrkr m))
  (t
   (setf ldwrku m)
   (setf chunk
           (the fixnum
                (truncate (- lwork (* m m) m) m)))
   (setf ldwrkr m)))
(setf itau
        (f2cl-lib:int-add ir (f2cl-lib:int-mul ldwrkr m)))
(setf iwork (f2cl-lib:int-add itau m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgelqf m n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "L" m m a lda u ldu)
(dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1)
 zero zero
 (f2cl-lib:array-slice u
                       double-float
                       (1 2)
                       ((1 ldu) (1 *)))
 ldu)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
```

```
    (dorglq m n m a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd m m u ldu s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(dlacpy "U" m m u ldu
 (f2cl-lib:array-slice work double-float (ir) ((1 *)))
 ldwrkr)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
```

```
    (dorgbr "P" m m m
     (f2cl-lib:array-slice work double-float (ir) ((1 *)))
     ldwrkr
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "Q" m m m u ldu
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13 var-14)
    (dbdsqr "U" m m m 0 s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work double-float (ir) ((1 *)))
     ldwrkr u ldu dum 1
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     info)
```

```
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12 var-13))
  (setf info var-14))
 (setf iu (f2cl-lib:int-add ie m))
 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i chunk))
               ((> i n) nil)
  (tagbody
    (setf blk
            (min
             (the fixnum
                  (f2cl-lib:int-add (f2cl-lib:int-sub n i)
                                    1))
             (the fixnum chunk)))
    (dgemm "N" "N" m blk m one
     (f2cl-lib:array-slice work double-float (ir) ((1 *)))
     ldwrkr
     (f2cl-lib:array-slice a
                           double-float
                           (1 i)
                           ((1 lda) (1 *)))
     lda zero
     (f2cl-lib:array-slice work double-float (iu) ((1 *)))
     ldwrku)
    (dlacpy "F" m blk
     (f2cl-lib:array-slice work double-float (iu) ((1 *)))
     ldwrku
     (f2cl-lib:array-slice a
                           double-float
                           (1 i)
                           ((1 lda) (1 *)))
     lda))))
(t
 (setf itau 1)
 (setf iwork (f2cl-lib:int-add itau m))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
     (dgelqf m n a lda
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
```

```
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "L" m m a lda u ldu)
(dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1)
 zero zero
 (f2cl-lib:array-slice u
                       double-float
                       (1 2)
                       ((1 ldu) (1 *)))
 ldu)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorglq m n m a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd m m u ldu s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
```

```
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13)
    (dormbr "P" "L" "T" m n m u ldu
     (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))
     a lda
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12))
  (setf ierr var-13))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "Q" m m m u ldu
     (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
```

```
      (setf iwork (f2cl-lib:int-add ie m))
      (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
             var-8 var-9 var-10 var-11 var-12 var-13 var-14)
          (dbdsqr "U" m n m 0 s
           (f2cl-lib:array-slice work double-float (ie) ((1 *)))
           a lda u ldu dum 1
           (f2cl-lib:array-slice work
                                 double-float
                                 (iwork)
                                 ((1 *)))
           info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8 var-9 var-10 var-11
                         var-12 var-13))
        (setf info var-14)))))
  (wntvs
   (cond
     (wntun
      (cond
        ((>= lwork
             (f2cl-lib:int-add (f2cl-lib:int-mul m m)
                               (max
                                (the fixnum
                                     (f2cl-lib:int-mul 4 m))
                                (the fixnum bdspac))))
         (setf ir 1)
         (cond
           ((>= lwork
                (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda m)))
            (setf ldwrkr lda))
           (t
            (setf ldwrkr m)))
         (setf itau
                (f2cl-lib:int-add ir
                                  (f2cl-lib:int-mul ldwrkr m)))
         (setf iwork (f2cl-lib:int-add itau m))
         (multiple-value-bind
               (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
             (dgelqf m n a lda
              (f2cl-lib:array-slice work
                                    double-float
                                    (itau)
                                    ((1 *)))
              (f2cl-lib:array-slice work
                                    double-float
```

```
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "L" m m a lda
 (f2cl-lib:array-slice work double-float (ir) ((1 *)))
 ldwrkr)
(dlaset "U" (f2cl-lib:int-sub m 1)
 (f2cl-lib:int-sub m 1) zero zero
 (f2cl-lib:array-slice work
                       double-float
                       ((+ ir ldwrkr))
                       ((1 *)))
 ldwrkr)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorglq m n m a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd m m
     (f2cl-lib:array-slice work
                           double-float
                           (ir)
                           ((1 *)))
```

```
      ldwrkr s
      (f2cl-lib:array-slice work
                            double-float
                            (ie)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (itauq)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (itaup)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                     var-6 var-7 var-8 var-9))
    (setf ierr var-10))
  (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9)
      (dorgbr "P" m m m
      (f2cl-lib:array-slice work
                            double-float
                            (ir)
                            ((1 *)))
      ldwrkr
      (f2cl-lib:array-slice work
                            double-float
                            (itaup)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                     var-6 var-7 var-8))
    (setf ierr var-9))
  (setf iwork (f2cl-lib:int-add ie m))
  (multiple-value-bind
```

```
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13 var-14)
     (dbdsqr "U" m m 0 0 s
      (f2cl-lib:array-slice work
                            double-float
                            (ie)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (ir)
                            ((1 *)))
      ldwrkr dum 1 dum 1
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12 var-13))
   (setf info var-14))
 (dgemm "N" "N" m n m one
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr a lda zero vt ldvt))
(t
 (setf itau 1)
 (setf iwork (f2cl-lib:int-add itau m))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
     (dgelqf m n a lda
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6))
   (setf ierr var-7))
 (dlacpy "U" m n a lda vt ldvt)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
```

```
      var-8)
    (dorglq m n m vt ldvt
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(dlaset "U" (f2cl-lib:int-sub m 1)
 (f2cl-lib:int-sub m 1) zero zero
 (f2cl-lib:array-slice a
                       double-float
                       (1 2)
                       ((1 lda) (1 *)))
 lda)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd m m a lda s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
```

```
          (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
          ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8 var-9))
        (setf ierr var-10))
       (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
             var-8 var-9 var-10 var-11 var-12 var-13)
          (dormbr "P" "L" "T" m n m a lda
           (f2cl-lib:array-slice work
                                 double-float
                                 (itaup)
                                 ((1 *)))
           vt ldvt
           (f2cl-lib:array-slice work
                                 double-float
                                 (iwork)
                                 ((1 *)))
           (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
           ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8 var-9 var-10 var-11
                         var-12))
        (setf ierr var-13))
       (setf iwork (f2cl-lib:int-add ie m))
       (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
             var-8 var-9 var-10 var-11 var-12 var-13 var-14)
          (dbdsqr "U" m n 0 0 s
           (f2cl-lib:array-slice work
                                 double-float
                                 (ie)
                                 ((1 *)))
           vt ldvt dum 1 dum 1
           (f2cl-lib:array-slice work
                                 double-float
                                 (iwork)
                                 ((1 *)))
           info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8 var-9 var-10 var-11
                         var-12 var-13))
        (setf info var-14)))))
  (wntuo
   (cond
     ((>= lwork
```

```
    (f2cl-lib:int-add (f2cl-lib:int-mul 2 m m)
                      (max
                       (the fixnum
                            (f2cl-lib:int-mul 4 m))
                       (the fixnum bdspac))))
(setf iu 1)
(cond
  ((>= lwork
       (f2cl-lib:int-add wrkbl
                         (f2cl-lib:int-mul 2 lda m)))
   (setf ldwrku lda)
   (setf ir
           (f2cl-lib:int-add iu
                             (f2cl-lib:int-mul ldwrku
                                               m)))
   (setf ldwrkr lda))
  ((>= lwork
       (f2cl-lib:int-add wrkbl
                         (f2cl-lib:int-mul
                          (f2cl-lib:int-add lda m)
                          m)))
   (setf ldwrku lda)
   (setf ir
           (f2cl-lib:int-add iu
                             (f2cl-lib:int-mul ldwrku
                                               m)))
   (setf ldwrkr m))
  (t
   (setf ldwrku m)
   (setf ir
           (f2cl-lib:int-add iu
                             (f2cl-lib:int-mul ldwrku
                                               m)))
   (setf ldwrkr m)))
(setf itau
        (f2cl-lib:int-add ir
                          (f2cl-lib:int-mul ldwrkr m)))
(setf iwork (f2cl-lib:int-add itau m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgelqf m n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
```

```
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "L" m m a lda
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku)
(dlaset "U" (f2cl-lib:int-sub m 1)
 (f2cl-lib:int-sub m 1) zero zero
 (f2cl-lib:array-slice work
                       double-float
                       ((+ iu ldwrku))
                       ((1 *)))
 ldwrku)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorglq m n m a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd m m
     (f2cl-lib:array-slice work
                           double-float
                           (iu)
```

```
                                ((1 *)))
          ldwrku s
          (f2cl-lib:array-slice work
                                double-float
                                (ie)
                                ((1 *)))
          (f2cl-lib:array-slice work
                                double-float
                                (itauq)
                                ((1 *)))
          (f2cl-lib:array-slice work
                                double-float
                                (itaup)
                                ((1 *)))
          (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
          (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
          ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8 var-9))
        (setf ierr var-10))
      (dlacpy "L" m m
       (f2cl-lib:array-slice work double-float (iu) ((1 *)))
       ldwrku
       (f2cl-lib:array-slice work double-float (ir) ((1 *)))
       ldwrkr)
      (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
             var-8 var-9)
          (dorgbr "P" m m m
           (f2cl-lib:array-slice work
                                 double-float
                                 (iu)
                                 ((1 *)))
           ldwrku
           (f2cl-lib:array-slice work
                                 double-float
                                 (itaup)
                                 ((1 *)))
           (f2cl-lib:array-slice work
                                 double-float
                                 (iwork)
                                 ((1 *)))
           (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
```

```
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
      var-8 var-9)
   (dorgbr "Q" m m m
    (f2cl-lib:array-slice work
                          double-float
                          (ir)
                          ((1 *)))
    ldwrkr
    (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))
    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
      var-8 var-9 var-10 var-11 var-12 var-13 var-14)
   (dbdsqr "U" m m m 0 s
    (f2cl-lib:array-slice work
                          double-float
                          (ie)
                          ((1 *)))
    (f2cl-lib:array-slice work
                          double-float
                          (iu)
                          ((1 *)))
    ldwrku
    (f2cl-lib:array-slice work
                          double-float
                          (ir)
                          ((1 *)))
    ldwrkr dum 1
    (f2cl-lib:array-slice work
```

```
                                double-float
                                (iwork)
                                ((1 *)))
        info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                     var-6 var-7 var-8 var-9 var-10 var-11
                     var-12 var-13))
    (setf info var-14))
  (dgemm "N" "N" m n m one
   (f2cl-lib:array-slice work double-float (iu) ((1 *)))
   ldwrku a lda zero vt ldvt)
  (dlacpy "F" m m
   (f2cl-lib:array-slice work double-float (ir) ((1 *)))
   ldwrkr a lda))
 (t
  (setf itau 1)
  (setf iwork (f2cl-lib:int-add itau m))
  (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
      (dgelqf m n a lda
       (f2cl-lib:array-slice work
                             double-float
                             (itau)
                             ((1 *)))
       (f2cl-lib:array-slice work
                             double-float
                             (iwork)
                             ((1 *)))
       (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
       ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                     var-6))
    (setf ierr var-7))
  (dlacpy "U" m n a lda vt ldvt)
  (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
         var-8)
      (dorglq m n m vt ldvt
       (f2cl-lib:array-slice work
                             double-float
                             (itau)
                             ((1 *)))
       (f2cl-lib:array-slice work
                             double-float
                             (iwork)
                             ((1 *)))
```

```
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(dlaset "U" (f2cl-lib:int-sub m 1)
 (f2cl-lib:int-sub m 1) zero zero
 (f2cl-lib:array-slice a
                       double-float
                       (1 2)
                       ((1 lda) (1 *)))
 lda)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd m m a lda s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13)
    (dormbr "P" "L" "T" m n m a lda
     (f2cl-lib:array-slice work
```

```
                              double-float
                              (itaup)
                              ((1 *)))
      vt ldvt
      (f2cl-lib:array-slice work
                              double-float
                              (iwork)
                              ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12))
  (setf ierr var-13))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "Q" m m m a lda
     (f2cl-lib:array-slice work
                              double-float
                              (itauq)
                              ((1 *)))
     (f2cl-lib:array-slice work
                              double-float
                              (iwork)
                              ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13 var-14)
    (dbdsqr "U" m n m 0 s
     (f2cl-lib:array-slice work
                              double-float
                              (ie)
                              ((1 *)))
     vt ldvt a lda dum 1
     (f2cl-lib:array-slice work
                              double-float
                              (iwork)
                              ((1 *)))
     info)
```

```
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                       var-6 var-7 var-8 var-9 var-10 var-11
                       var-12 var-13))
      (setf info var-14)))))
(wntuas
 (cond
   ((>= lwork
        (f2cl-lib:int-add (f2cl-lib:int-mul m m)
                          (max
                           (the fixnum
                                (f2cl-lib:int-mul 4 m))
                           (the fixnum bdspac))))
    (setf iu 1)
    (cond
      ((>= lwork
           (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda m)))
       (setf ldwrku lda))
      (t
       (setf ldwrku m)))
    (setf itau
            (f2cl-lib:int-add iu
                              (f2cl-lib:int-mul ldwrku m)))
    (setf iwork (f2cl-lib:int-add itau m))
    (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
        (dgelqf m n a lda
         (f2cl-lib:array-slice work
                               double-float
                               (itau)
                               ((1 *)))
         (f2cl-lib:array-slice work
                               double-float
                               (iwork)
                               ((1 *)))
         (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
         ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                       var-6))
      (setf ierr var-7))
    (dlacpy "L" m m a lda
     (f2cl-lib:array-slice work double-float (iu) ((1 *)))
     ldwrku)
    (dlaset "U" (f2cl-lib:int-sub m 1)
     (f2cl-lib:int-sub m 1) zero zero
     (f2cl-lib:array-slice work
                           double-float
```

```
                        ((+ iu ldwrku))
                        ((1 *)))
 ldwrku)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorglq m n m a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd m m
     (f2cl-lib:array-slice work
                           double-float
                           (iu)
                           ((1 *)))
     ldwrku s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
```

```
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(dlacpy "L" m m
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku u ldu)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "P" m m m
     (f2cl-lib:array-slice work
                          double-float
                          (iu)
                          ((1 *)))
     ldwrku
     (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "Q" m m m u ldu
     (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
```

```
     ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8))
   (setf ierr var-9))
 (setf iwork (f2cl-lib:int-add ie m))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10 var-11 var-12 var-13 var-14)
     (dbdsqr "U" m m m 0 s
      (f2cl-lib:array-slice work
                            double-float
                            (ie)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iu)
                            ((1 *)))
      ldwrku u ldu dum 1
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12 var-13))
   (setf info var-14))
 (dgemm "N" "N" m n m one
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku a lda zero vt ldvt))
(t
 (setf itau 1)
 (setf iwork (f2cl-lib:int-add itau m))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
     (dgelqf m n a lda
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
```

```
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "U" m n a lda vt ldvt)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorglq m n m vt ldvt
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(dlacpy "L" m m a lda u ldu)
(dlaset "U" (f2cl-lib:int-sub m 1)
 (f2cl-lib:int-sub m 1) zero zero
 (f2cl-lib:array-slice u
                       double-float
                       (1 2)
                       ((1 ldu) (1 *)))
 ldu)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd m m u ldu s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
```

```
                                double-float
                                (itaup)
                                ((1 *)))
        (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
        ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                     var-6 var-7 var-8 var-9))
    (setf ierr var-10))
  (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10 var-11 var-12 var-13)
      (dormbr "P" "L" "T" m n m u ldu
       (f2cl-lib:array-slice work
                                double-float
                                (itaup)
                                ((1 *)))
       vt ldvt
       (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
       (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
       ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                     var-6 var-7 var-8 var-9 var-10 var-11
                     var-12))
    (setf ierr var-13))
  (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9)
      (dorgbr "Q" m m m u ldu
       (f2cl-lib:array-slice work
                                double-float
                                (itauq)
                                ((1 *)))
       (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
       (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
       ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
```

```
                                var-6 var-7 var-8))
                      (setf ierr var-9))
                    (setf iwork (f2cl-lib:int-add ie m))
                    (multiple-value-bind
                          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                           var-8 var-9 var-10 var-11 var-12 var-13 var-14)
                        (dbdsqr "U" m n m 0 s
                         (f2cl-lib:array-slice work
                                               double-float
                                               (ie)
                                               ((1 *)))
                         vt ldvt u ldu dum 1
                         (f2cl-lib:array-slice work
                                               double-float
                                               (iwork)
                                               ((1 *)))
                         info)
                      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                                var-6 var-7 var-8 var-9 var-10 var-11
                                var-12 var-13))
                      (setf info var-14)))))))
(wntva
 (cond
   (wntun
    (cond
      ((>= lwork
           (f2cl-lib:int-add (f2cl-lib:int-mul m m)
                             (max
                              (the fixnum
                                   (f2cl-lib:int-add n m))
                              (the fixnum
                                   (f2cl-lib:int-mul 4 m))
                              (the fixnum bdspac))))
       (setf ir 1)
       (cond
         ((>= lwork
              (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda m)))
          (setf ldwrkr lda))
         (t
          (setf ldwrkr m)))
       (setf itau
             (f2cl-lib:int-add ir
                               (f2cl-lib:int-mul ldwrkr m)))
       (setf iwork (f2cl-lib:int-add itau m))
       (multiple-value-bind
             (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
```

```
    (dgelqf m n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "U" m n a lda vt ldvt)
(dlacpy "L" m m a lda
 (f2cl-lib:array-slice work double-float (ir) ((1 *)))
 ldwrkr)
(dlaset "U" (f2cl-lib:int-sub m 1)
 (f2cl-lib:int-sub m 1) zero zero
 (f2cl-lib:array-slice work
                       double-float
                       ((+ ir ldwrkr))
                       ((1 *)))
 ldwrkr)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorglq n n m vt ldvt
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
```

```
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd m m
     (f2cl-lib:array-slice work
                           double-float
                           (ir)
                           ((1 *)))
     ldwrkr s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "P" m m m
     (f2cl-lib:array-slice work
                           double-float
                           (ir)
                           ((1 *)))
     ldwrkr
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
```

```
                              ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8))
   (setf ierr var-9))
 (setf iwork (f2cl-lib:int-add ie m))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10 var-11 var-12 var-13 var-14)
     (dbdsqr "U" m m 0 0 s
      (f2cl-lib:array-slice work
                            double-float
                            (ie)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (ir)
                            ((1 *)))
      ldwrkr dum 1 dum 1
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12 var-13))
   (setf info var-14))
 (dgemm "N" "N" m n m one
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr vt ldvt zero a lda)
 (dlacpy "F" m n a lda vt ldvt))
(t
 (setf itau 1)
 (setf iwork (f2cl-lib:int-add itau m))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
     (dgelqf m n a lda
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
```

```
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "U" m n a lda vt ldvt)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorglq n n m vt ldvt
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(dlaset "U" (f2cl-lib:int-sub m 1)
 (f2cl-lib:int-sub m 1) zero zero
 (f2cl-lib:array-slice a
                       double-float
                       (1 2)
                       ((1 lda) (1 *)))
 lda)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd m m a lda s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
```

```
                              ((1 *)))
      (f2cl-lib:array-slice work
                              double-float
                              (itaup)
                              ((1 *)))
      (f2cl-lib:array-slice work
                              double-float
                              (iwork)
                              ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9))
   (setf ierr var-10))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10 var-11 var-12 var-13)
     (dormbr "P" "L" "T" m n m a lda
      (f2cl-lib:array-slice work
                              double-float
                              (itaup)
                              ((1 *)))
      vt ldvt
      (f2cl-lib:array-slice work
                              double-float
                              (iwork)
                              ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12))
   (setf ierr var-13))
 (setf iwork (f2cl-lib:int-add ie m))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10 var-11 var-12 var-13 var-14)
     (dbdsqr "U" m n 0 0 s
      (f2cl-lib:array-slice work
                              double-float
                              (ie)
                              ((1 *)))
      vt ldvt dum 1 dum 1
      (f2cl-lib:array-slice work
                              double-float
                              (iwork)
```

```
                                ((1 *)))
          info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8 var-9 var-10 var-11
                         var-12 var-13))
        (setf info var-14)))))
(wntuo
 (cond
   ((>= lwork
        (f2cl-lib:int-add (f2cl-lib:int-mul 2 m m)
                          (max
                           (the fixnum
                                (f2cl-lib:int-add n m))
                           (the fixnum
                                (f2cl-lib:int-mul 4 m))
                           (the fixnum bdspac))))
    (setf iu 1)
    (cond
      ((>= lwork
           (f2cl-lib:int-add wrkbl
                             (f2cl-lib:int-mul 2 lda m)))
       (setf ldwrku lda)
       (setf ir
               (f2cl-lib:int-add iu
                                 (f2cl-lib:int-mul ldwrku
                                                   m)))
       (setf ldwrkr lda))
      ((>= lwork
           (f2cl-lib:int-add wrkbl
                             (f2cl-lib:int-mul
                              (f2cl-lib:int-add lda m)
                              m)))
       (setf ldwrku lda)
       (setf ir
               (f2cl-lib:int-add iu
                                 (f2cl-lib:int-mul ldwrku
                                                   m)))
       (setf ldwrkr m))
      (t
       (setf ldwrku m)
       (setf ir
               (f2cl-lib:int-add iu
                                 (f2cl-lib:int-mul ldwrku
                                                   m)))
       (setf ldwrkr m)))
    (setf itau
```

```
        (f2cl-lib:int-add ir
                          (f2cl-lib:int-mul ldwrkr m)))
(setf iwork (f2cl-lib:int-add itau m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgelqf m n a lda
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "U" m n a lda vt ldvt)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorglq n n m vt ldvt
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(dlacpy "L" m m a lda
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku)
(dlaset "U" (f2cl-lib:int-sub m 1)
 (f2cl-lib:int-sub m 1) zero zero
 (f2cl-lib:array-slice work
                       double-float
                       ((+ iu ldwrku))
                       ((1 *)))
```

```
 ldwrku)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd m m
     (f2cl-lib:array-slice work
                           double-float
                           (iu)
                           ((1 *)))
     ldwrku s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(dlacpy "L" m m
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku
 (f2cl-lib:array-slice work double-float (ir) ((1 *)))
 ldwrkr)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "P" m m m
     (f2cl-lib:array-slice work
                           double-float
```

```
                                (iu)
                                ((1 *)))
          ldwrku
          (f2cl-lib:array-slice work
                                double-float
                                (itaup)
                                ((1 *)))
          (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
          (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
          ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8))
        (setf ierr var-9))
      (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
             var-8 var-9)
          (dorgbr "Q" m m m
          (f2cl-lib:array-slice work
                                double-float
                                (ir)
                                ((1 *)))
          ldwrkr
          (f2cl-lib:array-slice work
                                double-float
                                (itauq)
                                ((1 *)))
          (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
          (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
          ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8))
        (setf ierr var-9))
      (setf iwork (f2cl-lib:int-add ie m))
      (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
             var-8 var-9 var-10 var-11 var-12 var-13 var-14)
          (dbdsqr "U" m m m 0 s
          (f2cl-lib:array-slice work
                                double-float
                                (ie)
```

```
                          ((1 *)))
      (f2cl-lib:array-slice work
                          double-float
                          (iu)
                          ((1 *)))
      ldwrku
      (f2cl-lib:array-slice work
                          double-float
                          (ir)
                          ((1 *)))
      ldwrkr dum 1
      (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
      info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                     var-6 var-7 var-8 var-9 var-10 var-11
                     var-12 var-13))
    (setf info var-14))
   (dgemm "N" "N" m n m one
    (f2cl-lib:array-slice work double-float (iu) ((1 *)))
    ldwrku vt ldvt zero a lda)
   (dlacpy "F" m n a lda vt ldvt)
   (dlacpy "F" m m
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr a lda))
  (t
   (setf itau 1)
   (setf iwork (f2cl-lib:int-add itau m))
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
       (dgelqf m n a lda
        (f2cl-lib:array-slice work
                          double-float
                          (itau)
                          ((1 *)))
        (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
        ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                      var-6))
     (setf ierr var-7))
```

```
(dlacpy "U" m n a lda vt ldvt)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorglq n n m vt ldvt
     (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(dlaset "U" (f2cl-lib:int-sub m 1)
 (f2cl-lib:int-sub m 1) zero zero
 (f2cl-lib:array-slice a
                       double-float
                       (1 2)
                       ((1 lda) (1 *)))
 lda)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd m m a lda s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
```

```
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13)
    (dormbr "P" "L" "T" m n m a lda
     (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))
     vt ldvt
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12))
  (setf ierr var-13))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
    (dorgbr "Q" m m m a lda
     (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
```

```
                (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                 var-8 var-9 var-10 var-11 var-12 var-13 var-14)
               (dbdsqr "U" m n m 0 s
                (f2cl-lib:array-slice work
                                      double-float
                                      (ie)
                                      ((1 *)))
                vt ldvt a lda dum 1
                (f2cl-lib:array-slice work
                                      double-float
                                      (iwork)
                                      ((1 *)))
                info)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                               var-6 var-7 var-8 var-9 var-10 var-11
                               var-12 var-13))
              (setf info var-14)))))
        (wntuas
         (cond
           ((>= lwork
                (f2cl-lib:int-add (f2cl-lib:int-mul m m)
                                  (max
                                   (the fixnum
                                        (f2cl-lib:int-add n m))
                                   (the fixnum
                                        (f2cl-lib:int-mul 4 m))
                                   (the fixnum bdspac))))
            (setf iu 1)
            (cond
              ((>= lwork
                   (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda m)))
               (setf ldwrku lda))
              (t
               (setf ldwrku m)))
            (setf itau
                  (f2cl-lib:int-add iu
                                    (f2cl-lib:int-mul ldwrku m)))
            (setf iwork (f2cl-lib:int-add itau m))
            (multiple-value-bind
                  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
                (dgelqf m n a lda
                 (f2cl-lib:array-slice work
                                       double-float
                                       (itau)
                                       ((1 *)))
                 (f2cl-lib:array-slice work
```

```
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6))
  (setf ierr var-7))
(dlacpy "U" m n a lda vt ldvt)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8)
    (dorglq n n m vt ldvt
     (f2cl-lib:array-slice work
                          double-float
                          (itau)
                          ((1 *)))
     (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(dlacpy "L" m m a lda
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku)
(dlaset "U" (f2cl-lib:int-sub m 1)
 (f2cl-lib:int-sub m 1) zero zero
 (f2cl-lib:array-slice work
                       double-float
                       ((+ iu ldwrku))
                       ((1 *)))
 ldwrku)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd m m
     (f2cl-lib:array-slice work
                          double-float
```

```
                                (iu)
                                ((1 *)))
          ldwrku s
          (f2cl-lib:array-slice work
                                double-float
                                (ie)
                                ((1 *)))
          (f2cl-lib:array-slice work
                                double-float
                                (itauq)
                                ((1 *)))
          (f2cl-lib:array-slice work
                                double-float
                                (itaup)
                                ((1 *)))
          (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
          (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
          ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                     var-6 var-7 var-8 var-9))
    (setf ierr var-10))
  (dlacpy "L" m m
   (f2cl-lib:array-slice work double-float (iu) ((1 *)))
   ldwrku u ldu)
  (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
         var-8 var-9)
      (dorgbr "P" m m m
          (f2cl-lib:array-slice work
                                double-float
                                (iu)
                                ((1 *)))
          ldwrku
          (f2cl-lib:array-slice work
                                double-float
                                (itaup)
                                ((1 *)))
          (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
          (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
          ierr)
```

```
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
      var-8 var-9)
   (dorgbr "Q" m m m u ldu
    (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))
    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
      var-8 var-9 var-10 var-11 var-12 var-13 var-14)
   (dbdsqr "U" m m m 0 s
    (f2cl-lib:array-slice work
                          double-float
                          (ie)
                          ((1 *)))
    (f2cl-lib:array-slice work
                          double-float
                          (iu)
                          ((1 *)))
    ldwrku u ldu dum 1
    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12 var-13))
  (setf info var-14))
(dgemm "N" "N" m n m one
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku vt ldvt zero a lda)
```

```
 (dlacpy "F" m n a lda vt ldvt))
(t
 (setf itau 1)
 (setf iwork (f2cl-lib:int-add itau m))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
     (dgelqf m n a lda
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6))
   (setf ierr var-7))
 (dlacpy "U" m n a lda vt ldvt)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8)
     (dorglq n n m vt ldvt
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7))
   (setf ierr var-8))
 (dlacpy "L" m m a lda u ldu)
 (dlaset "U" (f2cl-lib:int-sub m 1)
  (f2cl-lib:int-sub m 1) zero zero
  (f2cl-lib:array-slice u
                        double-float
                        (1 2)
                        ((1 ldu) (1 *)))
  ldu)
```

```
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
    (dgebrd m m u ldu s
     (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13)
    (dormbr "P" "L" "T" m n m u ldu
     (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
     vt ldvt
     (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
     ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
```

```
                                var-12))
                  (setf ierr var-13))
                 (multiple-value-bind
                       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                        var-8 var-9)
                     (dorgbr "Q" m m m u ldu
                      (f2cl-lib:array-slice work
                                            double-float
                                            (itauq)
                                            ((1 *)))
                      (f2cl-lib:array-slice work
                                            double-float
                                            (iwork)
                                            ((1 *)))
                      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
                      ierr)
                   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                                    var-6 var-7 var-8))
                   (setf ierr var-9))
                 (setf iwork (f2cl-lib:int-add ie m))
                 (multiple-value-bind
                       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                        var-8 var-9 var-10 var-11 var-12 var-13 var-14)
                     (dbdsqr "U" m n m 0 s
                      (f2cl-lib:array-slice work
                                            double-float
                                            (ie)
                                            ((1 *)))
                      vt ldvt u ldu dum 1
                      (f2cl-lib:array-slice work
                                            double-float
                                            (iwork)
                                            ((1 *)))
                      info)
                   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                                    var-6 var-7 var-8 var-9 var-10 var-11
                                    var-12 var-13))
                   (setf info var-14)))))))))
           (t
            (setf ie 1)
            (setf itauq (f2cl-lib:int-add ie m))
            (setf itaup (f2cl-lib:int-add itauq m))
            (setf iwork (f2cl-lib:int-add itaup m))
            (multiple-value-bind
                  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                   var-9 var-10)
```

```
    (dgebrd m n a lda s
     (f2cl-lib:array-slice work double-float (ie) ((1 *)))
     (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
     (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
     (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9))
  (setf ierr var-10))
(cond
  (wntuas
   (dlacpy "L" m m a lda u ldu)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
          var-9)
       (dorgbr "Q" m m n u ldu
        (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
        (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7 var-8))
     (setf ierr var-9))))
(cond
  (wntvas
   (dlacpy "U" m n a lda vt ldvt)
   (if wntva (setf nrvt n))
   (if wntvs (setf nrvt m))
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
          var-9)
       (dorgbr "P" nrvt n m vt ldvt
        (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
        (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7 var-8))
     (setf ierr var-9))))
(cond
  (wntuo
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
          var-9)
       (dorgbr "Q" m m n a lda
        (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
        (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
```

```
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                       var-7 var-8))
      (setf ierr var-9))))
(cond
  (wntvo
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
          var-9)
       (dorgbr "P" m n m a lda
        (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
        (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                       var-7 var-8))
      (setf ierr var-9))))
(setf iwork (f2cl-lib:int-add ie m))
(if (or wntuas wntuo) (setf nru m))
(if wntun (setf nru 0))
(if (or wntvas wntvo) (setf ncvt n))
(if wntvn (setf ncvt 0))
(cond
  ((and (not wntuo) (not wntvo))
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
          var-9 var-10 var-11 var-12 var-13 var-14)
       (dbdsqr "L" m ncvt nru 0 s
        (f2cl-lib:array-slice work double-float (ie) ((1 *))) vt
        ldvt u ldu dum 1
        (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
        info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                       var-7 var-8 var-9 var-10 var-11 var-12
                       var-13))
      (setf info var-14)))
  ((and (not wntuo) wntvo)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
          var-9 var-10 var-11 var-12 var-13 var-14)
       (dbdsqr "L" m ncvt nru 0 s
        (f2cl-lib:array-slice work double-float (ie) ((1 *))) a
        lda u ldu dum 1
        (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
        info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                       var-7 var-8 var-9 var-10 var-11 var-12
                       var-13))
```

```
                (setf info var-14)))
               (t
                (multiple-value-bind
                      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                       var-9 var-10 var-11 var-12 var-13 var-14)
                    (dbdsqr "L" m ncvt nru 0 s
                     (f2cl-lib:array-slice work double-float (ie) ((1 *))) vt
                     ldvt a lda dum 1
                     (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
                     info)
                  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                   var-7 var-8 var-9 var-10 var-11 var-12
                                   var-13))
                  (setf info var-14))))))))
          (cond
            ((/= info 0)
             (cond
               ((> ie 2)
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                              ((> i
                                  (f2cl-lib:int-add minmn (f2cl-lib:int-sub 1)))
                               nil)
                  (tagbody
                    (setf (f2cl-lib:fref work-%data%
                                         ((f2cl-lib:int-add i 1))
                                         ((1 *))
                                         work-%offset%)
                          (f2cl-lib:fref work-%data%
                                         ((f2cl-lib:int-sub
                                           (f2cl-lib:int-add i ie)
                                           1))
                                         ((1 *))
                                         work-%offset%))))))
               (cond
                 ((< ie 2)
                  (f2cl-lib:fdo (i (f2cl-lib:int-add minmn (f2cl-lib:int-sub 1))
                                   (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                                ((> i 1) nil)
                    (tagbody
                      (setf (f2cl-lib:fref work-%data%
                                           ((f2cl-lib:int-add i 1))
                                           ((1 *))
                                           work-%offset%)
                            (f2cl-lib:fref work-%data%
                                           ((f2cl-lib:int-sub
                                             (f2cl-lib:int-add i ie)
```

```
                                          1))
                                         ((1 *))
                                         work-%offset%))))))))
        (cond
          ((= iscl 1)
           (if (> anrm bignum)
               (multiple-value-bind
                     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                      var-9)
                   (dlascl "G" 0 0 bignum anrm minmn 1 s minmn ierr)
                 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                  var-7 var-8))
                 (setf ierr var-9)))
           (if (and (/= info 0) (> anrm bignum))
               (multiple-value-bind
                     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                      var-9)
                   (dlascl "G" 0 0 bignum anrm (f2cl-lib:int-sub minmn 1) 1
                    (f2cl-lib:array-slice work double-float (2) ((1 *))) minmn
                    ierr)
                 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                  var-7 var-8))
                 (setf ierr var-9)))
           (if (< anrm smlnum)
               (multiple-value-bind
                     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                      var-9)
                   (dlascl "G" 0 0 smlnum anrm minmn 1 s minmn ierr)
                 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                  var-7 var-8))
                 (setf ierr var-9)))
           (if (and (/= info 0) (< anrm smlnum))
               (multiple-value-bind
                     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                      var-9)
                   (dlascl "G" 0 0 smlnum anrm (f2cl-lib:int-sub minmn 1) 1
                    (f2cl-lib:array-slice work double-float (2) ((1 *))) minmn
                    ierr)
                 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                  var-7 var-8))
                 (setf ierr var-9)))))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                (coerce (the fixnum maxwrk) 'double-float))
  end_label
        (return
         (values nil nil nil nil nil nil nil nil nil nil nil nil nil info))))))
```

7.18 dgesv LAPACK

⟨*dgesv.input*⟩≡

```
)set break resume
)sys rm -f dgesv.output
)spool dgesv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgesv.help*⟩≡

```
  ========================================================================
  dgesv examples
  ========================================================================

  ========================================================================
  Man Page Details
  ========================================================================

  NAME
         DGESV - the solution to a real system of linear equations  A * X = B,

  SYNOPSIS
         SUBROUTINE DGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )

             INTEGER       INFO, LDA, LDB, N, NRHS

             INTEGER       IPIV( * )

             DOUBLE        PRECISION A( LDA, * ), B( LDB, * )

  PURPOSE
         DGESV computes the solution to a real system of linear equations
            A  *  X  =  B, where A is an N-by-N matrix and X and B are N-by-NRHS
         matrices.

         The LU decomposition with partial pivoting and row interchanges is used
         to factor A as
            A = P * L * U,
         where  P  is a permutation matrix, L is unit lower triangular, and U is
         upper triangular.  The factored form of A is then  used  to  solve  the
         system of equations A * X = B.


  ARGUMENTS
         N       (input) INTEGER
                 The  number  of linear equations, i.e., the order of the matrix
                 A.  N >= 0.

         NRHS    (input) INTEGER
                 The number of right hand sides, i.e., the number of columns  of
                 the matrix B.  NRHS >= 0.

         A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
                 On  entry,  the N-by-N coefficient matrix A.  On exit, the fac-
                 tors L and U from the factorization A = P*L*U; the unit  diago-
```

```
          nal elements of L are not stored.

LDA       (input) INTEGER
          The leading dimension of the array A.  LDA >= max(1,N).

IPIV      (output) INTEGER array, dimension (N)
          The  pivot  indices that define the permutation matrix P; row i
          of the matrix was interchanged with row IPIV(i).

B         (input/output) DOUBLE PRECISION array, dimension (LDB,NRHS)
          On entry, the N-by-NRHS matrix of right hand side matrix B.  On
          exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB       (input) INTEGER
          The leading dimension of the array B.  LDB >= max(1,N).

INFO      (output) INTEGER
          = 0:  successful exit
          < 0:  if INFO = -i, the i-th argument had an illegal value
          >  0:   if INFO = i, U(i,i) is exactly zero.  The factorization
          has been completed, but the factor U is  exactly  singular,  so
          the solution could not be computed.
```

⟨*LAPACK dgesv*⟩≡

```
(defun dgesv (n nrhs a lda ipiv b ldb$ info)
  (declare (type (simple-array fixnum (*)) ipiv)
           (type (simple-array double-float (*)) b a)
           (type fixnum info ldb$ lda nrhs n))
  (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (b double-float b-%data% b-%offset%)
       (ipiv fixnum ipiv-%data% ipiv-%offset%))
    (prog ()
      (declare)
      (setf info 0)
      (cond
        ((< n 0)
         (setf info -1))
        ((< nrhs 0)
         (setf info -2))
        ((< lda (max (the fixnum 1) (the fixnum n)))
         (setf info -4))
        ((< ldb$ (max (the fixnum 1) (the fixnum n)))
         (setf info -7)))
      (cond
        ((/= info 0)
         (error
          " ** On entry to ~a parameter number ~a had an illegal value~%"
          "DGESV " (f2cl-lib:int-sub info))
         (go end_label)))
      (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
          (dgetrf n n a lda ipiv info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4))
        (setf info var-5))
      (cond
        ((= info 0)
         (multiple-value-bind
               (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
             (dgetrs "No transpose" n nrhs a lda ipiv b ldb$ info)
           (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7))
           (setf info var-8))))
      (go end_label)
 end_label
      (return (values nil nil nil nil nil nil nil info)))))
```

7.19 dgetf2 LAPACK

⟨*dgetf2.input*⟩≡

```
)set break resume
)sys rm -f dgetf2.output
)spool dgetf2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgetf2.help*⟩≡

```
  ======================================================================
  dgetf2 examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DGETF2 - an LU factorization of a general m-by-n matrix A using partial
         pivoting with row interchanges

  SYNOPSIS
         SUBROUTINE DGETF2( M, N, A, LDA, IPIV, INFO )

             INTEGER        INFO, LDA, M, N

             INTEGER        IPIV( * )

             DOUBLE         PRECISION A( LDA, * )

  PURPOSE
         DGETF2 computes an LU factorization of a general m-by-n matrix A  using
         partial pivoting with row interchanges.

         The factorization has the form
            A = P * L * U
         where P is a permutation matrix, L is lower triangular with unit diago-
         nal elements (lower trapezoidal if m > n), and U  is  upper  triangular
         (upper trapezoidal if m < n).

         This is the right-looking Level 2 BLAS version of the algorithm.


  ARGUMENTS
         M       (input) INTEGER
                 The number of rows of the matrix A.  M >= 0.

         N       (input) INTEGER
                 The number of columns of the matrix A.  N >= 0.

         A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
                 On  entry, the m by n matrix to be factored.  On exit, the fac-
                 tors L and U from the factorization A = P*L*U; the unit  diago-
                 nal elements of L are not stored.
```

```
LDA     (input) INTEGER
        The leading dimension of the array A.  LDA >= max(1,M).

IPIV    (output) INTEGER array, dimension (min(M,N))
        The  pivot indices; for 1 <= i <= min(M,N), row i of the matrix
        was interchanged with row IPIV(i).

INFO    (output) INTEGER
        = 0: successful exit
        < 0: if INFO = -k, the k-th argument had an illegal value
        > 0: if INFO = k, U(k,k) is exactly zero. The factorization has
        been completed, but the factor U is exactly singular, and divi-
        sion by zero will occur if it is used  to  solve  a  system  of
        equations.
```

⟨*LAPACK dgetf2*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dgetf2 (m n a lda ipiv info)
    (declare (type (simple-array fixnum (*)) ipiv)
             (type (simple-array double-float (*)) a)
             (type fixnum info lda n m))
    (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (ipiv fixnum ipiv-%data% ipiv-%offset%))
      (prog ((j 0) (jp 0))
        (declare (type fixnum j jp))
        (setf info 0)
        (cond
          ((< m 0)
           (setf info -1))
          ((< n 0)
           (setf info -2))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info -4)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DGETF2" (f2cl-lib:int-sub info))
           (go end_label)))
        (if (or (= m 0) (= n 0)) (go end_label))
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                      ((> j
                          (min (the fixnum m)
                               (the fixnum n)))
                       nil)
          (tagbody
            (setf jp
                    (f2cl-lib:int-add (f2cl-lib:int-sub j 1)
                                      (idamax
                                       (f2cl-lib:int-add (f2cl-lib:int-sub m j)
                                                         1)
                                       (f2cl-lib:array-slice a
                                                             double-float
                                                             (j j)
                                                             ((1 lda) (1 *)))
                                       1)))
            (setf (f2cl-lib:fref ipiv-%data% (j) ((1 *)) ipiv-%offset%) jp)
            (cond
```

```
              ((/= (f2cl-lib:fref a (jp j) ((1 lda) (1 *))) zero)
               (if (/= jp j)
                   (dswap n
                    (f2cl-lib:array-slice a double-float (j 1) ((1 lda) (1 *)))
                    lda
                    (f2cl-lib:array-slice a
                                          double-float
                                          (jp 1)
                                          ((1 lda) (1 *)))
                    lda))
               (if (< j m)
                   (dscal (f2cl-lib:int-sub m j)
                    (/ one
                       (f2cl-lib:fref a-%data%
                                      (j j)
                                      ((1 lda) (1 *))
                                      a-%offset%))
                    (f2cl-lib:array-slice a
                                          double-float
                                          ((+ j 1) j)
                                          ((1 lda) (1 *)))
                    1)))
              ((= info 0)
               (setf info j)))
             (cond
              ((< j (min (the fixnum m) (the fixnum n)))
               (dger (f2cl-lib:int-sub m j) (f2cl-lib:int-sub n j) (- one)
                (f2cl-lib:array-slice a
                                      double-float
                                      ((+ j 1) j)
                                      ((1 lda) (1 *)))
                1
                (f2cl-lib:array-slice a
                                      double-float
                                      (j (f2cl-lib:int-add j 1))
                                      ((1 lda) (1 *)))
                lda
                (f2cl-lib:array-slice a
                                      double-float
                                      ((+ j 1) (f2cl-lib:int-add j 1))
                                      ((1 lda) (1 *)))
                lda)))))
       (go end_label)
end_label
       (return (values nil nil nil nil nil info))))))
```

7.20 dgetrf LAPACK

⟨*dgetrf.input*⟩≡

```
)set break resume
)sys rm -f dgetrf.output
)spool dgetrf.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgetrf.help*⟩≡

```
=====================================================================
dgetrf examples
=====================================================================

=====================================================================
Man Page Details
=====================================================================

NAME
       DGETRF - an LU factorization of a general M-by-N matrix A using partial
       pivoting with row interchanges

SYNOPSIS
       SUBROUTINE DGETRF( M, N, A, LDA, IPIV, INFO )

           INTEGER        INFO, LDA, M, N

           INTEGER        IPIV( * )

           DOUBLE         PRECISION A( LDA, * )

PURPOSE
       DGETRF computes an LU factorization of a general M-by-N matrix A  using
       partial pivoting with row interchanges.

       The factorization has the form
          A = P * L * U
       where P is a permutation matrix, L is lower triangular with unit diago-
       nal elements (lower trapezoidal if m > n), and U  is  upper  triangular
       (upper trapezoidal if m < n).

       This is the right-looking Level 3 BLAS version of the algorithm.


ARGUMENTS
       M       (input) INTEGER
               The number of rows of the matrix A.  M >= 0.

       N       (input) INTEGER
               The number of columns of the matrix A.  N >= 0.

       A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
               On  entry, the M-by-N matrix to be factored.  On exit, the fac-
               tors L and U from the factorization A = P*L*U; the unit  diago-
               nal elements of L are not stored.
```

```
LDA     (input) INTEGER
        The leading dimension of the array A.  LDA >= max(1,M).

IPIV    (output) INTEGER array, dimension (min(M,N))
        The  pivot indices; for 1 <= i <= min(M,N), row i of the matrix
        was interchanged with row IPIV(i).

INFO    (output) INTEGER
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value
        > 0:  if INFO = i, U(i,i) is exactly  zero.  The  factorization
        has  been  completed, but the factor U is exactly singular, and
        division by zero will occur if it is used to solve a system  of
        equations.
```

⟨*LAPACK dgetrf*⟩≡

```
(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dgetrf (m n a lda ipiv info)
    (declare (type (simple-array fixnum (*)) ipiv)
             (type (simple-array double-float (*)) a)
             (type fixnum info lda n m))
    (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (ipiv fixnum ipiv-%data% ipiv-%offset%))
      (prog ((i 0) (iinfo 0) (j 0) (jb 0) (nb 0))
        (declare (type fixnum i iinfo j jb nb))
        (setf info 0)
        (cond
          ((< m 0)
           (setf info -1))
          ((< n 0)
           (setf info -2))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info -4)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DGETRF" (f2cl-lib:int-sub info))
           (go end_label)))
        (if (or (= m 0) (= n 0)) (go end_label))
        (setf nb (ilaenv 1 "DGETRF" " " m n -1 -1))
        (cond
          ((or (<= nb 1)
               (>= nb
                   (min (the fixnum m) (the fixnum n))))
           (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
               (dgetf2 m n a lda ipiv info)
             (declare (ignore var-0 var-1 var-2 var-3 var-4))
             (setf info var-5)))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j nb))
                         ((> j
                             (min (the fixnum m)
                                  (the fixnum n)))
                          nil)
             (tagbody
               (setf jb
                       (min
                        (the fixnum
```

```
                (f2cl-lib:int-add
                 (f2cl-lib:int-sub
                  (min (the fixnum m)
                       (the fixnum n))
                  j)
                 1))
          (the fixnum nb)))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
    (dgetf2 (f2cl-lib:int-add (f2cl-lib:int-sub m j) 1) jb
     (f2cl-lib:array-slice a double-float (j j) ((1 lda) (1 *)))
     lda
     (f2cl-lib:array-slice ipiv fixnum (j) ((1 *)))
     iinfo)
  (declare (ignore var-0 var-1 var-2 var-3 var-4))
  (setf iinfo var-5))
(if (and (= info 0) (> iinfo 0))
    (setf info (f2cl-lib:int-sub (f2cl-lib:int-add iinfo j) 1)))
(f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
              ((> i
                  (min (the fixnum m)
                       (the fixnum
                            (f2cl-lib:int-add j
                                              jb
                                              (f2cl-lib:int-sub
                                               1)))))
               nil)
  (tagbody
    (setf (f2cl-lib:fref ipiv-%data% (i) ((1 *)) ipiv-%offset%)
            (f2cl-lib:int-add (f2cl-lib:int-sub j 1)
                              (f2cl-lib:fref ipiv-%data%
                                             (i)
                                             ((1 *))
                                             ipiv-%offset%)))))
(dlaswp (f2cl-lib:int-sub j 1) a lda j
 (f2cl-lib:int-sub (f2cl-lib:int-add j jb) 1) ipiv 1)
(cond
  ((<= (f2cl-lib:int-add j jb) n)
   (dlaswp (f2cl-lib:int-add (f2cl-lib:int-sub n j jb) 1)
    (f2cl-lib:array-slice a
                          double-float
                          (1 (f2cl-lib:int-add j jb))
                          ((1 lda) (1 *)))
    lda j (f2cl-lib:int-sub (f2cl-lib:int-add j jb) 1) ipiv 1)
   (dtrsm "Left" "Lower" "No transpose" "Unit" jb
    (f2cl-lib:int-add (f2cl-lib:int-sub n j jb) 1) one
    (f2cl-lib:array-slice a double-float (j j) ((1 lda) (1 *)))
```

```
                    lda
                    (f2cl-lib:array-slice a
                                          double-float
                                          (j (f2cl-lib:int-add j jb))
                                          ((1 lda) (1 *)))
                    lda)
                   (cond
                     ((<= (f2cl-lib:int-add j jb) m)
                      (dgemm "No transpose" "No transpose"
                       (f2cl-lib:int-add (f2cl-lib:int-sub m j jb) 1)
                       (f2cl-lib:int-add (f2cl-lib:int-sub n j jb) 1) jb (- one)
                       (f2cl-lib:array-slice a
                                             double-float
                                             ((+ j jb) j)
                                             ((1 lda) (1 *)))
                       lda
                       (f2cl-lib:array-slice a
                                             double-float
                                             (j (f2cl-lib:int-add j jb))
                                             ((1 lda) (1 *)))
                       lda one
                       (f2cl-lib:array-slice a
                                             double-float
                                             ((+ j jb) (f2cl-lib:int-add j jb))
                                             ((1 lda) (1 *)))
                       lda)))))))))
  end_label
         (return (values nil nil nil nil nil info))))))
```

7.21 dgetrs LAPACK

⟨*dgetrs.input*⟩≡

```
)set break resume
)sys rm -f dgetrs.output
)spool dgetrs.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dgetrs.help*⟩≡

```
  ======================================================================
  dgetrs examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DGETRS  -  a system of linear equations  A * X = B or A' * X = B with a
         general N-by-N matrix A using the LU factorization computed by DGETRF

  SYNOPSIS
         SUBROUTINE DGETRS( TRANS, N, NRHS, A, LDA, IPIV, B, LDB, INFO )

             CHARACTER      TRANS

             INTEGER        INFO, LDA, LDB, N, NRHS

             INTEGER        IPIV( * )

             DOUBLE         PRECISION A( LDA, * ), B( LDB, * )

  PURPOSE
         DGETRS solves a system of linear equations
            A * X = B  or  A' * X = B with a general N-by-N matrix A  using  the
         LU factorization computed by DGETRF.


  ARGUMENTS
         TRANS   (input) CHARACTER*1
                 Specifies the form of the system of equations:
                 = 'N':  A * X = B  (No transpose)
                 = 'T':  A'* X = B  (Transpose)
                 = 'C':  A'* X = B  (Conjugate transpose = Transpose)

         N       (input) INTEGER
                 The order of the matrix A.  N >= 0.

         NRHS    (input) INTEGER
                 The  number of right hand sides, i.e., the number of columns of
                 the matrix B.  NRHS >= 0.

         A       (input) DOUBLE PRECISION array, dimension (LDA,N)
                 The factors L and U from the factorization A =  P*L*U  as  com-
```

```
          puted by DGETRF.

LDA       (input) INTEGER
          The leading dimension of the array A.  LDA >= max(1,N).

IPIV      (input) INTEGER array, dimension (N)
          The pivot indices from DGETRF; for 1<=i<=N, row i of the matrix
          was interchanged with row IPIV(i).

B         (input/output) DOUBLE PRECISION array, dimension (LDB,NRHS)
          On entry, the right hand side matrix B.  On exit, the  solution
          matrix X.

LDB       (input) INTEGER
          The leading dimension of the array B.  LDB >= max(1,N).

INFO      (output) INTEGER
          = 0:  successful exit
          < 0:  if INFO = -i, the i-th argument had an illegal value
```

⟨*LAPACK dgetrs*⟩≡

```
(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dgetrs (trans n nrhs a lda ipiv b ldb$ info)
    (declare (type (simple-array fixnum (*)) ipiv)
             (type (simple-array double-float (*)) b a)
             (type fixnum info ldb$ lda nrhs n)
             (type character trans))
    (f2cl-lib:with-multi-array-data
        ((trans character trans-%data% trans-%offset%)
         (a double-float a-%data% a-%offset%)
         (b double-float b-%data% b-%offset%)
         (ipiv fixnum ipiv-%data% ipiv-%offset%))
      (prog ((notran nil))
        (declare (type (member t nil) notran))
        (setf info 0)
        (setf notran (char-equal trans #\N))
        (cond
          ((and (not notran) (not (char-equal trans #\T)) (not (char-equal trans #\C)))
           (setf info -1))
          ((< n 0)
           (setf info -2))
          ((< nrhs 0)
           (setf info -3))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info -5))
          ((< ldb$ (max (the fixnum 1) (the fixnum n)))
           (setf info -8)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DGETRS" (f2cl-lib:int-sub info))
           (go end_label)))
        (if (or (= n 0) (= nrhs 0)) (go end_label))
        (cond
          (notran
           (dlaswp nrhs b ldb$ 1 n ipiv 1)
           (dtrsm "Left" "Lower" "No transpose" "Unit" n nrhs one a lda b ldb$)
           (dtrsm "Left" "Upper" "No transpose" "Non-unit" n nrhs one a lda b
            ldb$))
          (t
           (dtrsm "Left" "Upper" "Transpose" "Non-unit" n nrhs one a lda b ldb$)
           (dtrsm "Left" "Lower" "Transpose" "Unit" n nrhs one a lda b ldb$)
           (dlaswp nrhs b ldb$ 1 n ipiv -1)))
        (go end_label)
```

```
   end_label
          (return (values nil nil nil nil nil nil nil nil info))))))
```

7.22 dhseqr LAPACK

⟨*dhseqr.input*⟩≡

```
  )set break resume
  )sys rm -f dhseqr.output
  )spool dhseqr.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dhseqr.help*⟩≡

```
  ======================================================================
  dhseqr examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DHSEQR - compute the eigenvalues of a Hessenberg matrix H  and, option-
         ally, the matrices T and Z from the Schur decomposition  H = Z T  Z**T,
         where T is an upper quasi-triangular matrix (the  Schur form), and Z is
         the orthogonal matrix of Schur vectors

  SYNOPSIS
         SUBROUTINE DHSEQR( JOB, COMPZ, N, ILO, IHI, H, LDH,  WR,  WI,  Z,  LDZ,
                            WORK, LWORK, INFO )

             INTEGER        IHI, ILO, INFO, LDH, LDZ, LWORK, N

             CHARACTER      COMPZ, JOB

             DOUBLE         PRECISION  H( LDH, * ), WI( * ), WORK( * ), WR( * ),
                            Z( LDZ, * )

  PURPOSE
            DHSEQR computes the eigenvalues of a Hessenberg matrix H
            and, optionally, the matrices T and Z from the Schur decomposition
            H = Z T Z**T, where T is an upper quasi-triangular matrix (the
            Schur form), and Z is the orthogonal matrix of Schur vectors.

            Optionally Z may be postmultiplied into an input orthogonal
            matrix Q so that this routine can give the Schur factorization
            of a matrix A which has been reduced to the Hessenberg form H
            by the orthogonal matrix Q:  A = Q*H*Q**T = (QZ)*T*(QZ)**T.


  ARGUMENTS
         JOB   (input) CHARACTER*1
               = 'E':  compute eigenvalues only;
               = 'S':  compute eigenvalues and the Schur form T.

               COMPZ (input) CHARACTER*1
               = 'N':  no Schur vectors are computed;
               = 'I':  Z is initialized to the unit matrix and the matrix  Z  of
```

```
          Schur vectors of H is returned; = 'V':  Z must contain an orthog-
          onal matrix Q on entry, and the product Q*Z is returned.

    N     (input) INTEGER
          The order of the matrix H.  N .GE. 0.

    ILO   (input) INTEGER
          IHI   (input) INTEGER It is assumed that H is already upper  tri-
          angular  in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are
          normally set by a previous call to DGEBAL,  and  then  passed  to
          DGEHRD  when the matrix output by DGEBAL is reduced to Hessenberg
          form. Otherwise ILO and IHI should be set  to  1  and  N  respec-
          tively.   If  N.GT.0,  then 1.LE.ILO.LE.IHI.LE.N.  If N = 0, then
          ILO = 1 and IHI = 0.

    H     (input/output) DOUBLE PRECISION array, dimension (LDH,N)
          On entry, the upper Hessenberg matrix H.  On exit, if  INFO  =  0
          and  JOB = 'S', then H contains the upper quasi-triangular matrix
          T from the Schur decomposition (the Schur form); 2-by-2  diagonal
          blocks  (corresponding to complex conjugate pairs of eigenvalues)
          are returned in standard  form,  with  H(i,i)  =  H(i+1,i+1)  and
          H(i+1,i)*H(i,i+1).LT.0.  If  INFO = 0 and JOB = 'E', the contents
          of H are unspecified on  exit.   (The  output  value  of  H  when
          INFO.GT.0 is given under the description of INFO below.)

          Unlike earlier versions of DHSEQR, this subroutine may explicitly
          H(i,j) = 0 for i.GT.j and j = 1, 2,  ...  ILO-1  or  j  =  IHI+1,
          IHI+2, ... N.

    LDH   (input) INTEGER
          The leading dimension of the array H. LDH .GE. max(1,N).

    WR    (output) DOUBLE PRECISION array, dimension (N)
          WI    (output) DOUBLE PRECISION array, dimension (N) The real and
          imaginary parts, respectively, of the  computed  eigenvalues.  If
          two  eigenvalues  are  computed as a complex conjugate pair, they
          are stored in consecutive elements of WR and WI, say the i-th and
          (i+1)th,  with WI(i) .GT. 0 and WI(i+1) .LT. 0. If JOB = 'S', the
          eigenvalues are stored in the same order as on  the  diagonal  of
          the  Schur  form  returned  in  H,  with  WR(i)  = H(i,i) and, if
          H(i:i+1,i:i+1)  is  a  2-by-2  diagonal  block,  WI(i)  =
          sqrt(-H(i+1,i)*H(i,i+1)) and WI(i+1) = -WI(i).

    Z     (input/output) DOUBLE PRECISION array, dimension (LDZ,N)
          If  COMPZ = 'N', Z is not referenced.  If COMPZ = 'I', on entry Z
          need not be set and on exit, if INFO = 0, Z contains the orthogo-
```

nal matrix Z of the Schur vectors of H. If COMPZ = 'V', on entry Z must contain an N-by-N matrix Q, which is assumed to be equal to the unit matrix except for the submatrix Z(ILO:IHI,ILO:IHI). On exit, if INFO = 0, Z contains Q*Z. Normally Q is the orthogonal matrix generated by DORGHR after the call to DGEHRD which formed the Hessenberg matrix H. (The output value of Z when INFO.GT.0 is given under the description of INFO below.)

LDZ (input) INTEGER
The leading dimension of the array Z. if COMPZ = 'I' or COMPZ = 'V', then LDZ.GE.MAX(1,N). Otherwize, LDZ.GE.1.

WORK (workspace/output) DOUBLE PRECISION array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns an estimate of the optimal value for LWORK.

LWORK (input) INTEGER The dimension of the array WORK. LWORK .GE. max(1,N) is sufficient, but LWORK typically as large as 6*N may be required for optimal performance. A workspace query to determine the optimal workspace size is recommended.

If LWORK = -1, then DHSEQR does a workspace query. In this case, DHSEQR checks the input parameters and estimates the optimal workspace size for the given values of N, ILO and IHI. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

INFO (output) INTEGER
= 0: successful exit
value
the eigenvalues. Elements 1:ilo-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and JOB = 'E', then on exit, the remaining unconverged eigenvalues are the eigen- values of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and JOB = 'S', then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is an orthogonal matrix. The final value of H is upper Hessenberg and quasi-triangular in rows and columns INFO+1 through IHI.

```
If INFO .GT. 0 and COMPZ = 'V', then on exit

(final value of Z)  =  (initial value of Z)*U

where U is the orthogonal matrix in (*) (regard- less of the value
of JOB.)

If INFO .GT. 0 and COMPZ = 'I', then on exit (final value of Z)  =
U  where  U  is  the orthogonal matrix in (*) (regard- less of the
value of JOB.)

If INFO .GT. 0 and COMPZ = 'N', then Z is not accessed.
```

⟨*LAPACK dhseqr*⟩≡

```
(let* ((zero 0.0) (one 1.0) (two 2.0) (const 1.5) (nsmax 15) (lds nsmax))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two)
           (type (double-float 1.5 1.5) const)
           (type (fixnum 15 15) nsmax)
           (type fixnum lds))
  (defun dhseqr (job compz n ilo ihi h ldh wr wi z ldz work lwork info)
    (declare (type (simple-array double-float (*)) work z wi wr h)
             (type fixnum info lwork ldz ldh ihi ilo n)
             (type character compz job))
    (f2cl-lib:with-multi-array-data
        ((job character job-%data% job-%offset%)
         (compz character compz-%data% compz-%offset%)
         (h double-float h-%data% h-%offset%)
         (wr double-float wr-%data% wr-%offset%)
         (wi double-float wi-%data% wi-%offset%)
         (z double-float z-%data% z-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((s
              (make-array (the fixnum (reduce #'* (list lds nsmax)))
                          :element-type 'double-float))
             (v
              (make-array (f2cl-lib:int-add nsmax 1)
                          :element-type 'double-float))
             (vv
              (make-array (f2cl-lib:int-add nsmax 1)
                          :element-type 'double-float))
             (absw 0.0) (ovfl 0.0) (smlnum 0.0) (tau 0.0) (temp 0.0) (tst1 0.0)
             (ulp 0.0) (unfl 0.0) (i 0) (i1 0) (i2 0) (ierr 0) (ii 0) (itemp 0)
             (itn 0) (its 0) (j 0) (k 0) (l 0) (maxb 0) (nh 0) (nr 0) (ns 0)
             (nv 0) (initz nil) (lquery nil) (wantt nil) (wantz nil))
        (declare (type (simple-array double-float (*)) s v vv)
                 (type (double-float) absw ovfl smlnum tau temp tst1 ulp unfl)
                 (type fixnum i i1 i2 ierr ii itemp itn its j k l
                                          maxb nh nr ns nv)
                 (type (member t nil) initz lquery wantt wantz))
        (setf wantt (char-equal job #\S))
        (setf initz (char-equal compz #\I))
        (setf wantz (or initz (char-equal compz #\V)))
        (setf info 0)
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                (coerce
                 (the fixnum
                      (max (the fixnum 1)
```

```
                    (the fixnum n)))
          'double-float))
(setf lquery (coerce (= lwork -1) '(member t nil)))
(cond
  ((and (not (char-equal job #\E)) (not wantt))
   (setf info -1))
  ((and (not (char-equal compz #\N)) (not wantz))
   (setf info -2))
  ((< n 0)
   (setf info -3))
  ((or (< ilo 1)
       (> ilo
          (max (the fixnum 1) (the fixnum n))))
   (setf info -4))
  ((or
    (< ihi (min (the fixnum ilo) (the fixnum n)))
    (> ihi n))
   (setf info -5))
  ((< ldh (max (the fixnum 1) (the fixnum n)))
   (setf info -7))
  ((or (< ldz 1)
       (and wantz
            (< ldz
               (max (the fixnum 1)
                    (the fixnum n)))))
   (setf info -11))
  ((and
    (< lwork (max (the fixnum 1) (the fixnum n)))
    (not lquery))
   (setf info -13)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DHSEQR" (f2cl-lib:int-sub info))
   (go end_label))
  (lquery
   (go end_label)))
(if initz (dlaset "Full" n n zero one z ldz))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i (f2cl-lib:int-add ilo (f2cl-lib:int-sub 1))) nil)
  (tagbody
    (setf (f2cl-lib:fref wr-%data% (i) ((1 *)) wr-%offset%)
            (f2cl-lib:fref h-%data% (i i) ((1 ldh) (1 *)) h-%offset%))
    (setf (f2cl-lib:fref wi-%data% (i) ((1 *)) wi-%offset%) zero)))
(f2cl-lib:fdo (i (f2cl-lib:int-add ihi 1) (f2cl-lib:int-add i 1))
```

```
              ((> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref wr-%data% (i) ((1 *)) wr-%offset%)
            (f2cl-lib:fref h-%data% (i i) ((1 ldh) (1 *)) h-%offset%))
    (setf (f2cl-lib:fref wi-%data% (i) ((1 *)) wi-%offset%) zero)))
(if (= n 0) (go end_label))
(cond
  ((= ilo ihi)
   (setf (f2cl-lib:fref wr-%data% (ilo) ((1 *)) wr-%offset%)
           (f2cl-lib:fref h-%data%
                          (ilo ilo)
                          ((1 ldh) (1 *))
                          h-%offset%))
   (setf (f2cl-lib:fref wi-%data% (ilo) ((1 *)) wi-%offset%) zero)
   (go end_label)))
(f2cl-lib:fdo (j ilo (f2cl-lib:int-add j 1))
              ((> j (f2cl-lib:int-add ihi (f2cl-lib:int-sub 2))) nil)
  (tagbody
    (f2cl-lib:fdo (i (f2cl-lib:int-add j 2) (f2cl-lib:int-add i 1))
                  ((> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref h-%data% (i j) ((1 ldh) (1 *)) h-%offset%)
                zero)))))
(setf nh (f2cl-lib:int-add (f2cl-lib:int-sub ihi ilo) 1))
(setf ns (ilaenv 4 "DHSEQR" (f2cl-lib:f2cl-// job compz) n ilo ihi -1))
(setf maxb
        (ilaenv 8 "DHSEQR" (f2cl-lib:f2cl-// job compz) n ilo ihi -1))
(cond
  ((or (<= ns 2) (> ns nh) (>= maxb nh))
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
          var-10 var-11 var-12 var-13)
       (dlahqr wantt wantz n ilo ihi h ldh wr wi ilo ihi z ldz info)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                      var-8 var-9 var-10 var-11 var-12))
     (setf info var-13))
   (go end_label)))
(setf maxb
        (max (the fixnum 3) (the fixnum maxb)))
(setf ns
        (min (the fixnum ns)
             (the fixnum maxb)
             (the fixnum nsmax)))
(setf unfl (dlamch "Safe minimum"))
(setf ovfl (/ one unfl))
(multiple-value-bind (var-0 var-1)
```

```
            (dlabad unfl ovfl)
          (declare (ignore))
          (setf unfl var-0)
          (setf ovfl var-1))
        (setf ulp (dlamch "Precision"))
        (setf smlnum (* unfl (/ nh ulp)))
        (cond
          (wantt
           (setf i1 1)
           (setf i2 n)))
        (setf itn (f2cl-lib:int-mul 30 nh))
        (setf i ihi)
label50
        (setf l ilo)
        (if (< i ilo) (go label170))
        (f2cl-lib:fdo (its 0 (f2cl-lib:int-add its 1))
                      ((> its itn) nil)
          (tagbody
            (f2cl-lib:fdo (k i (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
                          ((> k (f2cl-lib:int-add l 1)) nil)
              (tagbody
                (setf tst1
                        (+
                         (abs
                          (f2cl-lib:fref h-%data%
                                         ((f2cl-lib:int-sub k 1)
                                          (f2cl-lib:int-sub k 1))
                                         ((1 ldh) (1 *))
                                         h-%offset%))
                         (abs
                          (f2cl-lib:fref h-%data%
                                         (k k)
                                         ((1 ldh) (1 *))
                                         h-%offset%))))
                (if (= tst1 zero)
                    (setf tst1
                            (dlanhs "1"
                             (f2cl-lib:int-add (f2cl-lib:int-sub i l) 1)
                             (f2cl-lib:array-slice h
                                                   double-float
                                                   (l l)
                                                   ((1 ldh) (1 *)))
                             ldh work)))
                (if
                 (<=
                  (abs
```

```
                    (f2cl-lib:fref h-%data%
                                   (k (f2cl-lib:int-sub k 1))
                                   ((1 ldh) (1 *))
                                   h-%offset%))
                   (max (* ulp tst1) smlnum))
                  (go label70))))
  label70
              (setf l k)
              (cond
                ((> l ilo)
                 (setf (f2cl-lib:fref h-%data%
                                      (l (f2cl-lib:int-sub l 1))
                                      ((1 ldh) (1 *))
                                      h-%offset%)
                       zero)))
              (if (>= l (f2cl-lib:int-add (f2cl-lib:int-sub i maxb) 1))
                  (go label160))
              (cond
                ((not wantt)
                 (setf i1 l)
                 (setf i2 i)))
              (cond
                ((or (= its 20) (= its 30))
                 (f2cl-lib:fdo (ii (f2cl-lib:int-add i (f2cl-lib:int-sub ns) 1)
                                (f2cl-lib:int-add ii 1))
                               ((> ii i) nil)
                   (tagbody
                     (setf (f2cl-lib:fref wr-%data% (ii) ((1 *)) wr-%offset%)
                             (* const
                                (+
                                 (abs
                                  (f2cl-lib:fref h-%data%
                                                 (ii (f2cl-lib:int-sub ii 1))
                                                 ((1 ldh) (1 *))
                                                 h-%offset%))
                                 (abs
                                  (f2cl-lib:fref h-%data%
                                                 (ii ii)
                                                 ((1 ldh) (1 *))
                                                 h-%offset%)))))
                     (setf (f2cl-lib:fref wi-%data% (ii) ((1 *)) wi-%offset%)
                             zero))))
                (t
                 (dlacpy "Full" ns ns
                  (f2cl-lib:array-slice h
                                        double-float
```

```
                                ((+ i (f2cl-lib:int-sub ns) 1)
                                 (f2cl-lib:int-add
                                  (f2cl-lib:int-sub i ns)
                                  1))
                                ((1 ldh) (1 *)))
                         ldh s lds)
                        (multiple-value-bind
                              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                               var-9 var-10 var-11 var-12 var-13)
                            (dlahqr nil nil ns 1 ns s lds
                             (f2cl-lib:array-slice wr
                                                   double-float
                                                   ((+ i (f2cl-lib:int-sub ns) 1))
                                                   ((1 *)))
                             (f2cl-lib:array-slice wi
                                                   double-float
                                                   ((+ i (f2cl-lib:int-sub ns) 1))
                                                   ((1 *)))
                             1 ns z ldz ierr)
                          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                           var-7 var-8 var-9 var-10 var-11 var-12))
                          (setf ierr var-13))
                        (cond
                          ((> ierr 0)
                           (f2cl-lib:fdo (ii 1 (f2cl-lib:int-add ii 1))
                                         ((> ii ierr) nil)
                             (tagbody
                               (setf (f2cl-lib:fref wr-%data%
                                                    ((f2cl-lib:int-add
                                                      (f2cl-lib:int-sub i ns)
                                                      ii))
                                                    ((1 *))
                                                    wr-%offset%)
                                     (f2cl-lib:fref s (ii ii) ((1 lds) (1 nsmax))))
                               (setf (f2cl-lib:fref wi-%data%
                                                    ((f2cl-lib:int-add
                                                      (f2cl-lib:int-sub i ns)
                                                      ii))
                                                    ((1 *))
                                                    wi-%offset%)
                                     zero)))))))
                    (setf (f2cl-lib:fref v (1) ((1 (f2cl-lib:int-add nsmax 1)))) one)
                    (f2cl-lib:fdo (ii 2 (f2cl-lib:int-add ii 1))
                                  ((> ii (f2cl-lib:int-add ns 1)) nil)
                      (tagbody
                        (setf (f2cl-lib:fref v (ii) ((1 (f2cl-lib:int-add nsmax 1))))
```

```
            zero)))
(setf nv 1)
(f2cl-lib:fdo (j (f2cl-lib:int-add i (f2cl-lib:int-sub ns) 1)
               (f2cl-lib:int-add j 1))
              ((> j i) nil)
  (tagbody
    (cond
      ((>= (f2cl-lib:fref wi (j) ((1 *))) zero)
       (cond
         ((= (f2cl-lib:fref wi (j) ((1 *))) zero)
          (dcopy (f2cl-lib:int-add nv 1) v 1 vv 1)
          (dgemv "No transpose" (f2cl-lib:int-add nv 1) nv one
           (f2cl-lib:array-slice h
                                 double-float
                                 (1 1)
                                 ((1 ldh) (1 *)))
           ldh vv 1
           (- (f2cl-lib:fref wr-%data% (j) ((1 *)) wr-%offset%)) v
           1)
          (setf nv (f2cl-lib:int-add nv 1)))
         ((> (f2cl-lib:fref wi (j) ((1 *))) zero)
          (dcopy (f2cl-lib:int-add nv 1) v 1 vv 1)
          (dgemv "No transpose" (f2cl-lib:int-add nv 1) nv one
           (f2cl-lib:array-slice h
                                 double-float
                                 (1 1)
                                 ((1 ldh) (1 *)))
           ldh v 1
           (* (- two)
              (f2cl-lib:fref wr-%data% (j) ((1 *)) wr-%offset%))
           vv 1)
          (setf itemp (idamax (f2cl-lib:int-add nv 1) vv 1))
          (setf temp
                (/ one
                   (max
                    (abs
                     (f2cl-lib:fref vv
                                    (itemp)
                                    ((1
                                      (f2cl-lib:int-add nsmax
                                                        1)))))
                    smlnum)))
          (dscal (f2cl-lib:int-add nv 1) temp vv 1)
          (setf absw
                (dlapy2
                 (f2cl-lib:fref wr-%data%
```

```
                                        (j)
                                        ((1 *))
                                        wr-%offset%)
                                (f2cl-lib:fref wi-%data%
                                        (j)
                                        ((1 *))
                                        wi-%offset%)))
                        (setf temp (* temp absw absw))
                        (dgemv "No transpose" (f2cl-lib:int-add nv 2)
                         (f2cl-lib:int-add nv 1) one
                         (f2cl-lib:array-slice h
                                               double-float
                                               (l l)
                                               ((1 ldh) (1 *)))
                         ldh vv 1 temp v 1)
                        (setf nv (f2cl-lib:int-add nv 2))))
                    (setf itemp (idamax nv v 1))
                    (setf temp
                            (abs
                             (f2cl-lib:fref v
                                            (itemp)
                                            ((1 (f2cl-lib:int-add nsmax 1))))))
                    (cond
                      ((= temp zero)
                       (setf (f2cl-lib:fref v
                                            (1)
                                            ((1 (f2cl-lib:int-add nsmax 1))))
                             one)
                       (f2cl-lib:fdo (ii 2 (f2cl-lib:int-add ii 1))
                                     ((> ii nv) nil)
                         (tagbody
                           (setf (f2cl-lib:fref v
                                                (ii)
                                                ((1
                                                  (f2cl-lib:int-add nsmax 1))))
                                 zero))))
                      (t
                       (setf temp (max temp smlnum))
                       (dscal nv (/ one temp) v 1)))))))
            (f2cl-lib:fdo (k l (f2cl-lib:int-add k 1))
                          ((> k (f2cl-lib:int-add i (f2cl-lib:int-sub 1))) nil)
              (tagbody
                (setf nr
                        (min (the fixnum (f2cl-lib:int-add ns 1))
                             (the fixnum
                                  (f2cl-lib:int-add (f2cl-lib:int-sub i k)
```

```
                                        1))))
(if (> k l)
    (dcopy nr
     (f2cl-lib:array-slice h
                           double-float
                           (k (f2cl-lib:int-sub k 1))
                           ((1 ldh) (1 *)))
     1 v 1))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlarfg nr
     (f2cl-lib:fref v (1) ((1 (f2cl-lib:int-add nsmax 1))))
     (f2cl-lib:array-slice v
                           double-float
                           (2)
                           ((1 (f2cl-lib:int-add nsmax 1))))
     1 tau)
  (declare (ignore var-0 var-2 var-3))
  (setf (f2cl-lib:fref v (1) ((1 (f2cl-lib:int-add nsmax 1))))
        var-1)
  (setf tau var-4))
(cond
  ((> k l)
   (setf (f2cl-lib:fref h-%data%
                        (k (f2cl-lib:int-sub k 1))
                        ((1 ldh) (1 *))
                        h-%offset%)
         (f2cl-lib:fref v
                        (1)
                        ((1 (f2cl-lib:int-add nsmax 1)))))
   (f2cl-lib:fdo (ii (f2cl-lib:int-add k 1)
                  (f2cl-lib:int-add ii 1))
                 ((> ii i) nil)
     (tagbody
       (setf (f2cl-lib:fref h-%data%
                            (ii (f2cl-lib:int-sub k 1))
                            ((1 ldh) (1 *))
                            h-%offset%)
             zero)))))
(setf (f2cl-lib:fref v (1) ((1 (f2cl-lib:int-add nsmax 1))))
       one)
(dlarfx "Left" nr (f2cl-lib:int-add (f2cl-lib:int-sub i2 k) 1)
 v tau
 (f2cl-lib:array-slice h double-float (k k) ((1 ldh) (1 *)))
 ldh work)
(dlarfx "Right"
 (f2cl-lib:int-add
```

```
                 (f2cl-lib:int-sub
                  (min (the fixnum (f2cl-lib:int-add k nr))
                       (the fixnum i))
                  i1)
                 1)
                nr v tau
                (f2cl-lib:array-slice h double-float (i1 k) ((1 ldh) (1 *)))
                ldh work)
               (cond
                 (wantz
                  (dlarfx "Right" nh nr v tau
                   (f2cl-lib:array-slice z
                                         double-float
                                         (ilo k)
                                         ((1 ldz) (1 *)))
                   ldz work)))))))
       (setf info i)
       (go end_label)
label160
       (multiple-value-bind
             (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
              var-10 var-11 var-12 var-13)
           (dlahqr wantt wantz n l i h ldh wr wi ilo ihi z ldz info)
         (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                          var-8 var-9 var-10 var-11 var-12))
         (setf info var-13))
       (if (> info 0) (go end_label))
       (setf itn (f2cl-lib:int-sub itn its))
       (setf i (f2cl-lib:int-sub l 1))
       (go label50)
label170
       (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
               (coerce
                (the fixnum
                     (max (the fixnum 1)
                          (the fixnum n)))
                'double-float))
end_label
       (return
        (values nil nil nil nil nil nil nil nil nil nil nil nil nil info))))))
```

7.23 dlabad LAPACK

⟨*dlabad.input*⟩≡

```
)set break resume
)sys rm -f dlabad.output
)spool dlabad.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlabad.help*⟩≡

```
======================================================================
dlabad examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DLABAD  - a input the values computed by DLAMCH for underflow and over-
       flow, and returns the square root of each of these values if the log of
       LARGE is sufficiently large

SYNOPSIS
       SUBROUTINE DLABAD( SMALL, LARGE )

           DOUBLE         PRECISION LARGE, SMALL

PURPOSE
       DLABAD  takes  as input the values computed by DLAMCH for underflow and
       overflow, and returns the square root of each of these  values  if  the
       log  of  LARGE  is  sufficiently large.  This subroutine is intended to
       identify machines with a large exponent range, such as the  Crays,  and
       redefine  the  underflow  and overflow limits to be the square roots of
       the values computed by  DLAMCH.   This  subroutine  is  needed  because
       DLAMCH does not compensate for poor arithmetic in the upper half of the
       exponent range, as is found on a Cray.


ARGUMENTS
       SMALL   (input/output) DOUBLE PRECISION
               On entry, the underflow threshold as computed  by  DLAMCH.   On
               exit, if LOG10(LARGE) is sufficiently large, the square root of
               SMALL, otherwise unchanged.

       LARGE   (input/output) DOUBLE PRECISION
               On entry, the overflow threshold as  computed  by  DLAMCH.   On
               exit, if LOG10(LARGE) is sufficiently large, the square root of
               LARGE, otherwise unchanged.
```

⟨*LAPACK dlabad*⟩≡
```
(defun dlabad (small large)
  (declare (type (double-float) large small))
  (prog ()
    (declare)
    (cond
      ((> (f2cl-lib:log10 large) 2000.0)
       (setf small (f2cl-lib:fsqrt small))
       (setf large (f2cl-lib:fsqrt large))))
    (return (values small large))))
```

7.24 dlabrd LAPACK

⟨*dlabrd.input*⟩≡
```
)set break resume
)sys rm -f dlabrd.output
)spool dlabrd.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlabrd.help*⟩≡

```
======================================================================
dlabrd examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DLABRD  - the first NB rows and columns of a real general m by n matrix
       A to upper or lower bidiagonal form by an orthogonal transformation  Q'
       * A * P, and returns the matrices X and Y which are needed to apply the
       transformation to the unreduced part of A

SYNOPSIS
       SUBROUTINE DLABRD( M, N, NB, A, LDA, D, E, TAUQ, TAUP, X, LDX, Y, LDY )

           INTEGER        LDA, LDX, LDY, M, N, NB

           DOUBLE         PRECISION  A(  LDA,  * ), D( * ), E( * ), TAUP( * ),
                          TAUQ( * ), X( LDX, * ), Y( LDY, * )

PURPOSE
       DLABRD reduces the first NB rows and columns of a real general m  by  n
       matrix A to upper or lower bidiagonal form by an orthogonal transforma-
       tion Q' * A * P, and returns the matrices X and Y which are  needed  to
       apply the transformation to the unreduced part of A.

       If  m  >=  n, A is reduced to upper bidiagonal form; if m < n, to lower
       bidiagonal form.

       This is an auxiliary routine called by DGEBRD


ARGUMENTS
       M       (input) INTEGER
               The number of rows in the matrix A.

       N       (input) INTEGER
               The number of columns in the matrix A.

       NB      (input) INTEGER
               The number of leading rows and columns of A to be reduced.

       A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
```

On entry, the m by n general matrix to be reduced. On exit, the first NB rows and columns of the matrix are overwritten; the rest of the array is unchanged. If m >= n, elements on and below the diagonal in the first NB columns, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors; and elements above the diagonal in the first NB rows, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors. If m < n, elements below the diagonal in the first NB columns, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and elements on and above the diagonal in the first NB rows, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors. See Further Details. LDA (input) INTEGER The leading dimension of the array A. LDA >= max(1,M).

D (output) DOUBLE PRECISION array, dimension (NB)
The diagonal elements of the first NB rows and columns of the reduced matrix. D(i) = A(i,i).

E (output) DOUBLE PRECISION array, dimension (NB)
The off-diagonal elements of the first NB rows and columns of the reduced matrix.

TAUQ (output) DOUBLE PRECISION array dimension (NB)
The scalar factors of the elementary reflectors which represent the orthogonal matrix Q. See Further Details. TAUP (output) DOUBLE PRECISION array, dimension (NB) The scalar factors of the elementary reflectors which represent the orthogonal matrix P. See Further Details. X (output) DOUBLE PRECISION array, dimension (LDX,NB) The m-by-nb matrix X required to update the unreduced part of A.

LDX (input) INTEGER
The leading dimension of the array X. LDX >= M.

Y (output) DOUBLE PRECISION array, dimension (LDY,NB)
The n-by-nb matrix Y required to update the unreduced part of A.

LDY (input) INTEGER
The leading dimension of the array Y. LDY >= N.

FURTHER DETAILS
The matrices Q and P are represented as products of elementary reflectors:

```
   Q = H(1) H(2) . . . H(nb)  and  P = G(1) G(2) . . . G(nb)

Each H(i) and G(i) has the form:

   H(i) = I - tauq * v * v'  and G(i) = I - taup * u * u'

where tauq and taup are real scalars, and v and u are real vectors.

If m >= n, v(1:i-1) = 0, v(i) = 1, and v(i:m)  is  stored  on  exit  in
A(i:m,i);  u(1:i)  =  0,  u(i+1) = 1, and u(i+1:n) is stored on exit in
A(i,i+1:n); tauq is stored in TAUQ(i) and taup in TAUP(i).

If m < n, v(1:i) = 0, v(i+1) = 1, and v(i+1:m) is  stored  on  exit  in
A(i+2:m,i);  u(1:i-1)  =  0,  u(i) = 1, and u(i:n) is stored on exit in
A(i,i+1:n); tauq is stored in TAUQ(i) and taup in TAUP(i).

The elements of the vectors v and u together form the m-by-nb matrix  V
and  the nb-by-n matrix U' which are needed, with X and Y, to apply the
transformation to the unreduced part  of  the  matrix,  using  a  block
update of the form:  A := A - V*Y' - X*U'.

The  contents  of  A  on exit are illustrated by the following examples
with nb = 2:

m = 6 and n = 5 (m > n):          m = 5 and n = 6 (m < n):

  (  1   1   u1  u1  u1 )           (  1   u1  u1  u1  u1  u1 )
  (  v1  1   1   u2  u2 )           (  1   1   u2  u2  u2  u2 )
  (  v1  v2  a   a   a  )           (  v1  1   a   a   a   a  )
  (  v1  v2  a   a   a  )           (  v1  v2  a   a   a   a  )
  (  v1  v2  a   a   a  )           (  v1  v2  a   a   a   a  )
  (  v1  v2  a   a   a  )
```

where a denotes an element of the original matrix which is unchanged, vi denotes an element of the vector defining H(i), and ui an element of the vector defining G(i).

⟨*LAPACK dlabrd*⟩≡

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dlabrd (m n nb a lda d e tauq taup x ldx y ldy)
    (declare (type (simple-array double-float (*)) y x taup tauq e d a)
             (type fixnum ldy ldx lda nb n m))
    (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (d double-float d-%data% d-%offset%)
         (e double-float e-%data% e-%offset%)
         (tauq double-float tauq-%data% tauq-%offset%)
         (taup double-float taup-%data% taup-%offset%)
         (x double-float x-%data% x-%offset%)
         (y double-float y-%data% y-%offset%))
      (prog ((i 0))
        (declare (type fixnum i))
        (if (or (<= m 0) (<= n 0)) (go end_label))
        (cond
          ((>= m n)
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                         ((> i nb) nil)
             (tagbody
               (dgemv "No transpose"
                (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
                (f2cl-lib:int-sub i 1) (- one)
                (f2cl-lib:array-slice a double-float (i 1) ((1 lda) (1 *))) lda
                (f2cl-lib:array-slice y double-float (i 1) ((1 ldy) (1 *))) ldy
                one (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
                1)
               (dgemv "No transpose"
                (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
                (f2cl-lib:int-sub i 1) (- one)
                (f2cl-lib:array-slice x double-float (i 1) ((1 ldx) (1 *))) ldx
                (f2cl-lib:array-slice a double-float (1 i) ((1 lda) (1 *))) 1
                one (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
                1)
               (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                   (dlarfg (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
                    (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                    (f2cl-lib:array-slice a
                                          double-float
                                          ((min (f2cl-lib:int-add i 1) m) i)
                                          ((1 lda) (1 *)))
                    1 (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%))
                 (declare (ignore var-0 var-2 var-3))
```

```
  (setf (f2cl-lib:fref a-%data%
                       (i i)
                       ((1 lda) (1 *))
                       a-%offset%)
        var-1)
  (setf (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
        var-4))
(setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
      (f2cl-lib:fref a-%data%
                     (i i)
                     ((1 lda) (1 *))
                     a-%offset%))
(cond
  ((< i n)
   (setf (f2cl-lib:fref a-%data%
                        (i i)
                        ((1 lda) (1 *))
                        a-%offset%)
         one)
   (dgemv "Transpose"
    (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
    (f2cl-lib:int-sub n i) one
    (f2cl-lib:array-slice a
                          double-float
                          (i (f2cl-lib:int-add i 1))
                          ((1 lda) (1 *)))
    lda
    (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
    1 zero
    (f2cl-lib:array-slice y
                          double-float
                          ((+ i 1) i)
                          ((1 ldy) (1 *)))
    1)
   (dgemv "Transpose"
    (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
    (f2cl-lib:int-sub i 1) one
    (f2cl-lib:array-slice a double-float (i 1) ((1 lda) (1 *)))
    lda
    (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
    1 zero
    (f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
    1)
   (dgemv "No transpose" (f2cl-lib:int-sub n i)
    (f2cl-lib:int-sub i 1) (- one)
    (f2cl-lib:array-slice y
```

```
                        double-float
                        ((+ i 1) 1)
                        ((1 ldy) (1 *)))
 ldy
 (f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
 1 one
 (f2cl-lib:array-slice y
                        double-float
                        ((+ i 1) i)
                        ((1 ldy) (1 *)))
 1)
(dgemv "Transpose"
 (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
 (f2cl-lib:int-sub i 1) one
 (f2cl-lib:array-slice x double-float (i 1) ((1 ldx) (1 *)))
 ldx
 (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
 1 zero
 (f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
 1)
(dgemv "Transpose" (f2cl-lib:int-sub i 1)
 (f2cl-lib:int-sub n i) (- one)
 (f2cl-lib:array-slice a
                        double-float
                        (1 (f2cl-lib:int-add i 1))
                        ((1 lda) (1 *)))
 lda
 (f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
 1 one
 (f2cl-lib:array-slice y
                        double-float
                        ((+ i 1) i)
                        ((1 ldy) (1 *)))
 1)
(dscal (f2cl-lib:int-sub n i)
 (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
 (f2cl-lib:array-slice y
                        double-float
                        ((+ i 1) i)
                        ((1 ldy) (1 *)))
 1)
(dgemv "No transpose" (f2cl-lib:int-sub n i) i (- one)
 (f2cl-lib:array-slice y
                        double-float
                        ((+ i 1) 1)
                        ((1 ldy) (1 *)))
```

```
 ldy
 (f2cl-lib:array-slice a double-float (i 1) ((1 lda) (1 *)))
 lda one
 (f2cl-lib:array-slice a
                       double-float
                       (i (f2cl-lib:int-add i 1))
                       ((1 lda) (1 *)))
 lda)
(dgemv "Transpose" (f2cl-lib:int-sub i 1)
 (f2cl-lib:int-sub n i) (- one)
 (f2cl-lib:array-slice a
                       double-float
                       (1 (f2cl-lib:int-add i 1))
                       ((1 lda) (1 *)))
 lda
 (f2cl-lib:array-slice x double-float (i 1) ((1 ldx) (1 *)))
 ldx one
 (f2cl-lib:array-slice a
                       double-float
                       (i (f2cl-lib:int-add i 1))
                       ((1 lda) (1 *)))
 lda)
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlarfg (f2cl-lib:int-sub n i)
     (f2cl-lib:fref a-%data%
                    (i (f2cl-lib:int-add i 1))
                    ((1 lda) (1 *))
                    a-%offset%)
     (f2cl-lib:array-slice a
                           double-float
                           (i
                            (min
                             (the fixnum
                                  (f2cl-lib:int-add i 2))
                             (the fixnum n)))
                           ((1 lda) (1 *)))
     lda
     (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%))
  (declare (ignore var-0 var-2 var-3))
  (setf (f2cl-lib:fref a-%data%
                       (i (f2cl-lib:int-add i 1))
                       ((1 lda) (1 *))
                       a-%offset%)
        var-1)
  (setf (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
        var-4))
```

```
(setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
        (f2cl-lib:fref a-%data%
                       (i (f2cl-lib:int-add i 1))
                       ((1 lda) (1 *))
                       a-%offset%))
(setf (f2cl-lib:fref a-%data%
                     (i (f2cl-lib:int-add i 1))
                     ((1 lda) (1 *))
                     a-%offset%)
        one)
(dgemv "No transpose" (f2cl-lib:int-sub m i)
 (f2cl-lib:int-sub n i) one
 (f2cl-lib:array-slice a
                       double-float
                       ((+ i 1) (f2cl-lib:int-add i 1))
                       ((1 lda) (1 *)))
 lda
 (f2cl-lib:array-slice a
                       double-float
                       (i (f2cl-lib:int-add i 1))
                       ((1 lda) (1 *)))
 lda zero
 (f2cl-lib:array-slice x
                       double-float
                       ((+ i 1) i)
                       ((1 ldx) (1 *)))
 1)
(dgemv "Transpose" (f2cl-lib:int-sub n i) i one
 (f2cl-lib:array-slice y
                       double-float
                       ((+ i 1) 1)
                       ((1 ldy) (1 *)))
 ldy
 (f2cl-lib:array-slice a
                       double-float
                       (i (f2cl-lib:int-add i 1))
                       ((1 lda) (1 *)))
 lda zero
 (f2cl-lib:array-slice x double-float (1 i) ((1 ldx) (1 *)))
 1)
(dgemv "No transpose" (f2cl-lib:int-sub m i) i (- one)
 (f2cl-lib:array-slice a
                       double-float
                       ((+ i 1) 1)
                       ((1 lda) (1 *)))
 lda
```

```
                (f2cl-lib:array-slice x double-float (1 i) ((1 ldx) (1 *)))
                1 one
                (f2cl-lib:array-slice x
                                      double-float
                                      ((+ i 1) i)
                                      ((1 ldx) (1 *)))
                1)
               (dgemv "No transpose" (f2cl-lib:int-sub i 1)
                (f2cl-lib:int-sub n i) one
                (f2cl-lib:array-slice a
                                      double-float
                                      (1 (f2cl-lib:int-add i 1))
                                      ((1 lda) (1 *)))
                lda
                (f2cl-lib:array-slice a
                                      double-float
                                      (i (f2cl-lib:int-add i 1))
                                      ((1 lda) (1 *)))
                lda zero
                (f2cl-lib:array-slice x double-float (1 i) ((1 ldx) (1 *)))
                1)
               (dgemv "No transpose" (f2cl-lib:int-sub m i)
                (f2cl-lib:int-sub i 1) (- one)
                (f2cl-lib:array-slice x
                                      double-float
                                      ((+ i 1) 1)
                                      ((1 ldx) (1 *)))
                ldx
                (f2cl-lib:array-slice x double-float (1 i) ((1 ldx) (1 *)))
                1 one
                (f2cl-lib:array-slice x
                                      double-float
                                      ((+ i 1) i)
                                      ((1 ldx) (1 *)))
                1)
               (dscal (f2cl-lib:int-sub m i)
                (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
                (f2cl-lib:array-slice x
                                      double-float
                                      ((+ i 1) i)
                                      ((1 ldx) (1 *)))
                1))))))
      (t
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i nb) nil)
         (tagbody
```

```
(dgemv "No transpose"
 (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
 (f2cl-lib:int-sub i 1) (- one)
 (f2cl-lib:array-slice y double-float (i 1) ((1 ldy) (1 *))) ldy
 (f2cl-lib:array-slice a double-float (i 1) ((1 lda) (1 *))) lda
 one (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
 lda)
(dgemv "Transpose" (f2cl-lib:int-sub i 1)
 (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) (- one)
 (f2cl-lib:array-slice a double-float (1 i) ((1 lda) (1 *))) lda
 (f2cl-lib:array-slice x double-float (i 1) ((1 ldx) (1 *))) ldx
 one (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
 lda)
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlarfg (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
     (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
     (f2cl-lib:array-slice a
                           double-float
                           (i
                            (min
                             (the fixnum
                                  (f2cl-lib:int-add i 1))
                             (the fixnum n)))
                           ((1 lda) (1 *)))
     lda (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%))
  (declare (ignore var-0 var-2 var-3))
  (setf (f2cl-lib:fref a-%data%
                       (i i)
                       ((1 lda) (1 *))
                       a-%offset%)
        var-1)
  (setf (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
        var-4))
(setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
       (f2cl-lib:fref a-%data%
                      (i i)
                      ((1 lda) (1 *))
                      a-%offset%))
(cond
  ((< i m)
   (setf (f2cl-lib:fref a-%data%
                        (i i)
                        ((1 lda) (1 *))
                        a-%offset%)
         one)
   (dgemv "No transpose" (f2cl-lib:int-sub m i)
```

```
 (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) one
 (f2cl-lib:array-slice a
                       double-float
                       ((+ i 1) i)
                       ((1 lda) (1 *)))
 lda
 (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
 lda zero
 (f2cl-lib:array-slice x
                       double-float
                       ((+ i 1) i)
                       ((1 ldx) (1 *)))
 1)
(dgemv "Transpose"
 (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
 (f2cl-lib:int-sub i 1) one
 (f2cl-lib:array-slice y double-float (i 1) ((1 ldy) (1 *)))
 ldy
 (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
 lda zero
 (f2cl-lib:array-slice x double-float (1 i) ((1 ldx) (1 *)))
 1)
(dgemv "No transpose" (f2cl-lib:int-sub m i)
 (f2cl-lib:int-sub i 1) (- one)
 (f2cl-lib:array-slice a
                       double-float
                       ((+ i 1) 1)
                       ((1 lda) (1 *)))
 lda
 (f2cl-lib:array-slice x double-float (1 i) ((1 ldx) (1 *)))
 1 one
 (f2cl-lib:array-slice x
                       double-float
                       ((+ i 1) i)
                       ((1 ldx) (1 *)))
 1)
(dgemv "No transpose" (f2cl-lib:int-sub i 1)
 (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) one
 (f2cl-lib:array-slice a double-float (1 i) ((1 lda) (1 *)))
 lda
 (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
 lda zero
 (f2cl-lib:array-slice x double-float (1 i) ((1 ldx) (1 *)))
 1)
(dgemv "No transpose" (f2cl-lib:int-sub m i)
 (f2cl-lib:int-sub i 1) (- one)
```

```
 (f2cl-lib:array-slice x
                       double-float
                       ((+ i 1) 1)
                       ((1 ldx) (1 *)))
 ldx
 (f2cl-lib:array-slice x double-float (1 i) ((1 ldx) (1 *)))
 1 one
 (f2cl-lib:array-slice x
                       double-float
                       ((+ i 1) i)
                       ((1 ldx) (1 *)))
 1)
(dscal (f2cl-lib:int-sub m i)
 (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
 (f2cl-lib:array-slice x
                       double-float
                       ((+ i 1) i)
                       ((1 ldx) (1 *)))
 1)
(dgemv "No transpose" (f2cl-lib:int-sub m i)
 (f2cl-lib:int-sub i 1) (- one)
 (f2cl-lib:array-slice a
                       double-float
                       ((+ i 1) 1)
                       ((1 lda) (1 *)))
 lda
 (f2cl-lib:array-slice y double-float (i 1) ((1 ldy) (1 *)))
 ldy one
 (f2cl-lib:array-slice a
                       double-float
                       ((+ i 1) i)
                       ((1 lda) (1 *)))
 1)
(dgemv "No transpose" (f2cl-lib:int-sub m i) i (- one)
 (f2cl-lib:array-slice x
                       double-float
                       ((+ i 1) 1)
                       ((1 ldx) (1 *)))
 ldx
 (f2cl-lib:array-slice a double-float (1 i) ((1 lda) (1 *)))
 1 one
 (f2cl-lib:array-slice a
                       double-float
                       ((+ i 1) i)
                       ((1 lda) (1 *)))
 1)
```

```
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlarfg (f2cl-lib:int-sub m i)
     (f2cl-lib:fref a-%data%
                    ((f2cl-lib:int-add i 1) i)
                    ((1 lda) (1 *))
                    a-%offset%)
     (f2cl-lib:array-slice a
                           double-float
                           ((min (f2cl-lib:int-add i 2) m) i)
                           ((1 lda) (1 *)))
     1 (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%))
  (declare (ignore var-0 var-2 var-3))
  (setf (f2cl-lib:fref a-%data%
                       ((f2cl-lib:int-add i 1) i)
                       ((1 lda) (1 *))
                       a-%offset%)
          var-1)
  (setf (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
          var-4))
(setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
        (f2cl-lib:fref a-%data%
                       ((f2cl-lib:int-add i 1) i)
                       ((1 lda) (1 *))
                       a-%offset%))
(setf (f2cl-lib:fref a-%data%
                     ((f2cl-lib:int-add i 1) i)
                     ((1 lda) (1 *))
                     a-%offset%)
        one)
(dgemv "Transpose" (f2cl-lib:int-sub m i)
 (f2cl-lib:int-sub n i) one
 (f2cl-lib:array-slice a
                       double-float
                       ((+ i 1) (f2cl-lib:int-add i 1))
                       ((1 lda) (1 *)))
 lda
 (f2cl-lib:array-slice a
                       double-float
                       ((+ i 1) i)
                       ((1 lda) (1 *)))
 1 zero
 (f2cl-lib:array-slice y
                       double-float
                       ((+ i 1) i)
                       ((1 ldy) (1 *)))
 1)
```

```
(dgemv "Transpose" (f2cl-lib:int-sub m i)
 (f2cl-lib:int-sub i 1) one
 (f2cl-lib:array-slice a
                       double-float
                       ((+ i 1) 1)
                       ((1 lda) (1 *)))
 lda
 (f2cl-lib:array-slice a
                       double-float
                       ((+ i 1) i)
                       ((1 lda) (1 *)))
 1 zero
 (f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
 1)
(dgemv "No transpose" (f2cl-lib:int-sub n i)
 (f2cl-lib:int-sub i 1) (- one)
 (f2cl-lib:array-slice y
                       double-float
                       ((+ i 1) 1)
                       ((1 ldy) (1 *)))
 ldy
 (f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
 1 one
 (f2cl-lib:array-slice y
                       double-float
                       ((+ i 1) i)
                       ((1 ldy) (1 *)))
 1)
(dgemv "Transpose" (f2cl-lib:int-sub m i) i one
 (f2cl-lib:array-slice x
                       double-float
                       ((+ i 1) 1)
                       ((1 ldx) (1 *)))
 ldx
 (f2cl-lib:array-slice a
                       double-float
                       ((+ i 1) i)
                       ((1 lda) (1 *)))
 1 zero
 (f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
 1)
(dgemv "Transpose" i (f2cl-lib:int-sub n i) (- one)
 (f2cl-lib:array-slice a
                       double-float
                       (1 (f2cl-lib:int-add i 1))
                       ((1 lda) (1 *)))
```

```
                    lda
                    (f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
                    1 one
                    (f2cl-lib:array-slice y
                                          double-float
                                          ((+ i 1) i)
                                          ((1 ldy) (1 *)))
                    1)
                   (dscal (f2cl-lib:int-sub n i)
                    (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
                    (f2cl-lib:array-slice y
                                          double-float
                                          ((+ i 1) i)
                                          ((1 ldy) (1 *)))
                    1)))))))
  end_label
         (return
          (values nil nil nil nil nil nil nil nil nil nil nil nil nil))))))
```

7.25 dlacon LAPACK

⟨*dlacon.input*⟩≡

```
)set break resume
)sys rm -f dlacon.output
)spool dlacon.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlacon.help*⟩≡

```
  ======================================================================
  dlacon examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLACON - the 1-norm of a square, real matrix A

  SYNOPSIS
         SUBROUTINE DLACON( N, V, X, ISGN, EST, KASE )

             INTEGER        KASE, N

             DOUBLE         PRECISION EST

             INTEGER        ISGN( * )

             DOUBLE         PRECISION V( * ), X( * )

  PURPOSE
         DLACON estimates the 1-norm of a square, real matrix A.  Reverse commu-
         nication is used for evaluating matrix-vector products.


  ARGUMENTS
         N       (input) INTEGER
                 The order of the matrix.  N >= 1.

         V       (workspace) DOUBLE PRECISION array, dimension (N)
                 On the final return, V = A*W,  where  EST =  norm(V)/norm(W)  (W
                 is not returned).

         X       (input/output) DOUBLE PRECISION array, dimension (N)
                 On  an  intermediate  return,  X should be overwritten by A * X,
                 if KASE=1, A' * X,  if KASE=2, and DLACON must be re-called with
                 all the other parameters unchanged.

         ISGN    (workspace) INTEGER array, dimension (N)

         EST     (input/output) DOUBLE PRECISION
                 On  entry  with  KASE  =  1  or  2  and  JUMP = 3, EST should be
                 unchanged from the previous call to DLACON.  On exit, EST is  an
```

```
              estimate (a lower bound) for norm(A).

       KASE   (input/output) INTEGER
              On the initial call to DLACON, KASE should be 0.  On an interme-
              diate return, KASE will be 1 or 2, indicating whether  X  should
              be  overwritten  by  A * X  or A' * X.  On the final return from
              DLACON, KASE will again be 0.

FURTHER DETAILS
       Reference: N.J. Higham, "FORTRAN codes for estimating the one-norm of a
       real or complex matrix, with applications to condition estimation", ACM
       Trans. Math. Soft., vol. 14, no. 4, pp. 381-396, December 1988.
```

⟨*LAPACK dlacon*⟩≡

```
(let* ((itmax 5) (zero 0.0) (one 1.0) (two 2.0))
  (declare (type (fixnum 5 5) itmax)
           (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two))
  (let ((altsgn 0.0)
        (estold 0.0)
        (temp 0.0)
        (i 0)
        (iter 0)
        (j 0)
        (jlast 0)
        (jump 0))
    (declare (type fixnum itmax jump jlast j iter i)
             (type (double-float) two one zero temp estold altsgn))
    (defun dlacon (n v x isgn est kase)
      (declare (type (double-float) est)
               (type (simple-array fixnum (*)) isgn)
               (type (simple-array double-float (*)) x v)
               (type fixnum kase n))
      (f2cl-lib:with-multi-array-data
          ((v double-float v-%data% v-%offset%)
           (x double-float x-%data% x-%offset%)
           (isgn fixnum isgn-%data% isgn-%offset%))
        (prog ()
          (declare)
          (cond
            ((= kase 0)
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i n) nil)
               (tagbody
                 (setf (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%)
                         (/ one (coerce (realpart n) 'double-float)))))
             (setf kase 1)
             (setf jump 1)
             (go end_label)))
          (f2cl-lib:computed-goto (label20 label40 label70 label110 label140)
                                  jump)
  label20
          (cond
            ((= n 1)
             (setf (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%)
                     (f2cl-lib:fref x-%data% (1) ((1 *)) x-%offset%))
             (setf est (abs (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%)))
             (go label150)))
```

```
          (setf est (dasum n x 1))
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%)
                      (f2cl-lib:sign one
                                     (f2cl-lib:fref x-%data%
                                                    (i)
                                                    ((1 *))
                                                    x-%offset%)))
              (setf (f2cl-lib:fref isgn-%data% (i) ((1 *)) isgn-%offset%)
                      (values (round
                       (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%))))))
          (setf kase 2)
          (setf jump 2)
          (go end_label)
label40
          (setf j (idamax n x 1))
          (setf iter 2)
label50
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%) zero)))
          (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%) one)
          (setf kase 1)
          (setf jump 3)
          (go end_label)
label70
          (dcopy n x 1 v 1)
          (setf estold est)
          (setf est (dasum n v 1))
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i n) nil)
            (tagbody
              (if
               (/=
                (values (round
                 (f2cl-lib:sign one
                                (f2cl-lib:fref x-%data%
                                               (i)
                                               ((1 *))
                                               x-%offset%))))
                (f2cl-lib:fref isgn-%data% (i) ((1 *)) isgn-%offset%))
               (go label90))))
          (go label120)
```

```
label90
          (if (<= est estold) (go label120))
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%)
                      (f2cl-lib:sign one
                                     (f2cl-lib:fref x-%data%
                                                    (i)
                                                    ((1 *))
                                                    x-%offset%)))
              (setf (f2cl-lib:fref isgn-%data% (i) ((1 *)) isgn-%offset%)
                      (values (round
                       (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%))))))
          (setf kase 2)
          (setf jump 4)
          (go end_label)
label110
          (setf jlast j)
          (setf j (idamax n x 1))
          (cond
            ((and
              (/= (f2cl-lib:fref x (jlast) ((1 *)))
                  (abs (f2cl-lib:fref x (j) ((1 *)))))
              (< iter itmax))
             (setf iter (f2cl-lib:int-add iter 1))
             (go label50)))
label120
          (setf altsgn one)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%)
               (* altsgn
                (+ one
                 (/
                  (coerce (realpart (f2cl-lib:int-sub i 1)) 'double-float)
                  (coerce (realpart (f2cl-lib:int-sub n 1)) 'double-float)))))
              (setf altsgn (- altsgn))))
          (setf kase 1)
          (setf jump 5)
          (go end_label)
label140
          (setf temp
           (* two
            (/ (dasum n x 1)
```

```
                (coerce (realpart (f2cl-lib:int-mul 3 n)) 'double-float))))
          (cond
            ((> temp est)
             (dcopy n x 1 v 1)
             (setf est temp)))
  label150
          (setf kase 0)
  end_label
          (return (values nil nil nil nil est kase)))))))
```

7.26 dlacpy LAPACK

⟨*dlacpy.input*⟩≡

```
)set break resume
)sys rm -f dlacpy.output
)spool dlacpy.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlacpy.help*⟩≡

```
  ======================================================================
  dlacpy examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLACPY - all or part of a two-dimensional matrix A to another matrix B

  SYNOPSIS
         SUBROUTINE DLACPY( UPLO, M, N, A, LDA, B, LDB )

             CHARACTER      UPLO

             INTEGER        LDA, LDB, M, N

             DOUBLE         PRECISION A( LDA, * ), B( LDB, * )

  PURPOSE
         DLACPY  copies  all  or  part  of a two-dimensional matrix A to another
         matrix B.


  ARGUMENTS
         UPLO    (input) CHARACTER*1
                 Specifies the part of the matrix A to be copied to B.   =  'U':
                 Upper triangular part
                 = 'L':      Lower triangular part
                 Otherwise:  All of the matrix A

         M       (input) INTEGER
                 The number of rows of the matrix A.  M >= 0.

         N       (input) INTEGER
                 The number of columns of the matrix A.  N >= 0.

         A       (input) DOUBLE PRECISION array, dimension (LDA,N)
                 The m by n matrix A.  If UPLO = 'U', only the upper triangle or
                 trapezoid is accessed; if UPLO = 'L', only the  lower  triangle
                 or trapezoid is accessed.

         LDA     (input) INTEGER
                 The leading dimension of the array A.  LDA >= max(1,M).
```

```
B       (output) DOUBLE PRECISION array, dimension (LDB,N)
        On exit, B = A in the locations specified by UPLO.

LDB     (input) INTEGER
        The leading dimension of the array B.  LDB >= max(1,M).
```

⟨*LAPACK dlacpy*⟩≡

```
(defun dlacpy (uplo m n a lda b ldb$)
  (declare (type (simple-array double-float (*)) b a)
           (type fixnum ldb$ lda n m)
           (type character uplo))
  (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (a double-float a-%data% a-%offset%)
       (b double-float b-%data% b-%offset%))
    (prog ((i 0) (j 0))
      (declare (type fixnum j i))
      (cond
        ((char-equal uplo #\U)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i
                               (min (the fixnum j)
                                    (the fixnum m)))
                            nil)
               (tagbody
                 (setf (f2cl-lib:fref b-%data%
                                      (i j)
                                      ((1 ldb$) (1 *))
                                      b-%offset%)
                       (f2cl-lib:fref a-%data%
                                      (i j)
                                      ((1 lda) (1 *))
                                      a-%offset%)))))))
        ((char-equal uplo #\L)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                           ((> i m) nil)
               (tagbody
                 (setf (f2cl-lib:fref b-%data%
                                      (i j)
                                      ((1 ldb$) (1 *))
                                      b-%offset%)
                       (f2cl-lib:fref a-%data%
                                      (i j)
                                      ((1 lda) (1 *))
                                      a-%offset%)))))))
        (t
```

```
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                        ((> j n) nil)
            (tagbody
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                            ((> i m) nil)
                (tagbody
                  (setf (f2cl-lib:fref b-%data%
                                       (i j)
                                       ((1 ldb$) (1 *))
                                       b-%offset%)
                        (f2cl-lib:fref a-%data%
                                       (i j)
                                       ((1 lda) (1 *))
                                       a-%offset%))))))))
   end_label
        (return (values nil nil nil nil nil nil nil)))))
```

7.27 dladiv LAPACK

⟨*dladiv.input*⟩≡

```
)set break resume
)sys rm -f dladiv.output
)spool dladiv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dladiv.help*⟩≡

```
==========================================================================
dladiv examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

NAME
       DLADIV  -  complex  division  in  real  arithmetic   a + i*b  p + i*q =
       ---------  c + i*d  The algorithm is due to Robert L

SYNOPSIS
       SUBROUTINE DLADIV( A, B, C, D, P, Q )

           DOUBLE         PRECISION A, B, C, D, P, Q

PURPOSE
       DLADIV performs complex division in  real arithmetic in D.  Knuth,  The
       art of Computer Programming, Vol.2, p.195


ARGUMENTS
       A       (input) DOUBLE PRECISION
               B        (input) DOUBLE PRECISION C       (input) DOUBLE PRECI-
               SION D       (input) DOUBLE PRECISION The scalars a, b, c,  and
               d in the above expression.

       P       (output) DOUBLE PRECISION
               Q        (output)  DOUBLE  PRECISION The scalars p and q in the
               above expression.
```

⟨*LAPACK dladiv*⟩≡
```
(defun dladiv (a b c d p q)
  (declare (type (double-float) q p d c b a))
  (prog ((e 0.0) (f 0.0))
    (declare (type (double-float) f e))
    (cond
      ((< (abs d) (abs c))
       (setf e (/ d c))
       (setf f (+ c (* d e)))
       (setf p (/ (+ a (* b e)) f))
       (setf q (/ (- b (* a e)) f)))
      (t
       (setf e (/ c d))
       (setf f (+ d (* c e)))
       (setf p (/ (+ b (* a e)) f))
       (setf q (/ (- (* b e) a) f))))
    (return (values nil nil nil nil p q))))
```

7.28 dlaed6 LAPACK

⟨*dlaed6.input*⟩≡
```
)set break resume
)sys rm -f dlaed6.output
)spool dlaed6.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlaed6.help*⟩≡

```
  ======================================================================
  dlaed6 examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLAED6 - the positive or negative root (closest to the origin) of  z(1)
         z(2) z(3) f(x) = rho +  ---------  +  ----------  +  ---------   d(1)-x
         d(2)-x d(3)-x  It is assumed that   if ORGATI = .true

  SYNOPSIS
         SUBROUTINE DLAED6( KNITER, ORGATI, RHO, D, Z, FINIT, TAU, INFO )

             LOGICAL         ORGATI

             INTEGER         INFO, KNITER

             DOUBLE          PRECISION FINIT, RHO, TAU

             DOUBLE          PRECISION D( 3 ), Z( 3 )

  PURPOSE
         DLAED6  computes  the positive or negative root (closest to the origin)
         of
                         z(1)        z(2)        z(3) f(x) =   rho +  ---------
         + ---------- + ---------
                        d(1)-x      d(2)-x      d(3)-x
               otherwise it is between d(1) and d(2)

         This  routine  will  be called by DLAED4 when necessary. In most cases,
         the root sought is the smallest in magnitude, though it might not be in
         some extremely rare situations.


  ARGUMENTS
         KNITER       (input) INTEGER
                      Refer to DLAED4 for its significance.

         ORGATI       (input) LOGICAL
                      If  ORGATI  is  true,  the needed root is between d(2) and
                      d(3); otherwise it is between d(1) and d(2).   See  DLAED4
                      for further details.
```

```
RHO             (input) DOUBLE PRECISION
                Refer to the equation f(x) above.

D               (input) DOUBLE PRECISION array, dimension (3)
                D satisfies d(1) < d(2) < d(3).

Z               (input) DOUBLE PRECISION array, dimension (3)
                Each of the elements in z must be positive.

FINIT           (input) DOUBLE PRECISION
                The  value  of  f  at  0. It is more accurate than the one
                evaluated inside this routine (if someone wants to do so).

TAU             (output) DOUBLE PRECISION
                The root of the equation f(x).

INFO            (output) INTEGER
                = 0: successful exit
                > 0: if INFO = 1, failure to converge
```

⟨*LAPACK dlaed6*⟩≡

```
(let* ((maxit 20)
       (zero 0.0)
       (one 1.0)
       (two 2.0)
       (three 3.0)
       (four 4.0)
       (eight 8.0))
  (declare (type (fixnum 20 20) maxit)
           (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two)
           (type (double-float 3.0 3.0) three)
           (type (double-float 4.0 4.0) four)
           (type (double-float 8.0 8.0) eight))
  (let ((small1 0.0)
        (sminv1 0.0)
        (small2 0.0)
        (sminv2 0.0)
        (eps 0.0)
        (first$ nil))
    (declare (type (member t nil) first$)
             (type (double-float) eps sminv2 small2 sminv1 small1))
    (setq first$ t)
    (defun dlaed6 (kniter orgati rho d z finit tau info)
      (declare (type (simple-array double-float (*)) z d)
               (type (double-float) tau finit rho)
               (type (member t nil) orgati)
               (type fixnum info kniter))
      (f2cl-lib:with-multi-array-data
          ((d double-float d-%data% d-%offset%)
           (z double-float z-%data% z-%offset%))
        (prog ((a 0.0) (b 0.0) (base 0.0) (c 0.0) (ddf 0.0) (df 0.0)
               (erretm 0.0) (eta 0.0) (f 0.0) (fc 0.0) (sclfac 0.0)
               (sclinv 0.0) (temp 0.0) (temp1 0.0) (temp2 0.0) (temp3 0.0)
               (temp4 0.0) (i 0) (iter 0) (niter 0) (scale nil)
               (dscale (make-array 3 :element-type 'double-float))
               (zscale (make-array 3 :element-type 'double-float)))
          (declare (type (double-float) a b base c ddf df erretm eta f fc
                                        sclfac sclinv temp temp1 temp2 temp3
                                        temp4)
                   (type fixnum i iter niter)
                   (type (member t nil) scale)
                   (type (simple-array double-float (3)) dscale zscale))
          (setf info 0)
          (setf niter 1)
```

```
(setf tau zero)
(cond
  ((= kniter 2)
   (cond
     (orgati
      (setf temp
              (/
               (- (f2cl-lib:fref d-%data% (3) ((1 3)) d-%offset%)
                  (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%))
               two))
      (setf c
              (+ rho
                 (/ (f2cl-lib:fref z-%data% (1) ((1 3)) z-%offset%)
                    (-
                     (f2cl-lib:fref d-%data% (1) ((1 3)) d-%offset%)
                     (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%)
                     temp))))
      (setf a
              (+
               (* c
                  (+ (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%)
                     (f2cl-lib:fref d-%data%
                                    (3)
                                    ((1 3))
                                    d-%offset%)))
               (f2cl-lib:fref z-%data% (2) ((1 3)) z-%offset%)
               (f2cl-lib:fref z-%data% (3) ((1 3)) z-%offset%)))
      (setf b
              (+
               (* c
                  (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%)
                  (f2cl-lib:fref d-%data% (3) ((1 3)) d-%offset%))
               (* (f2cl-lib:fref z-%data% (2) ((1 3)) z-%offset%)
                  (f2cl-lib:fref d-%data% (3) ((1 3)) d-%offset%))
               (* (f2cl-lib:fref z-%data% (3) ((1 3)) z-%offset%)
                  (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%)))))
     (t
      (setf temp
              (/
               (- (f2cl-lib:fref d-%data% (1) ((1 3)) d-%offset%)
                  (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%))
               two))
      (setf c
              (+ rho
                 (/ (f2cl-lib:fref z-%data% (3) ((1 3)) z-%offset%)
                    (-
```

```
                     (f2cl-lib:fref d-%data% (3) ((1 3)) d-%offset%)
                     (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%)
                     temp))))
      (setf a
            (+
             (* c
                (+ (f2cl-lib:fref d-%data% (1) ((1 3)) d-%offset%)
                   (f2cl-lib:fref d-%data%
                                  (2)
                                  ((1 3))
                                  d-%offset%)))
             (f2cl-lib:fref z-%data% (1) ((1 3)) z-%offset%)
             (f2cl-lib:fref z-%data% (2) ((1 3)) z-%offset%)))
      (setf b
            (+
             (* c
                (f2cl-lib:fref d-%data% (1) ((1 3)) d-%offset%)
                (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%))
             (* (f2cl-lib:fref z-%data% (1) ((1 3)) z-%offset%)
                (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%))
             (* (f2cl-lib:fref z-%data% (2) ((1 3)) z-%offset%)
                (f2cl-lib:fref d-%data% (1) ((1 3)) d-%offset%))))))
(setf temp (max (abs a) (abs b) (abs c)))
(setf a (/ a temp))
(setf b (/ b temp))
(setf c (/ c temp))
(cond
  ((= c zero)
   (setf tau (/ b a)))
  ((<= a zero)
   (setf tau
         (/
          (- a
             (f2cl-lib:fsqrt
              (abs (+ (* a a) (* (- four) b c)))))
          (* two c))))
  (t
   (setf tau
         (/ (* two b)
            (+ a
               (f2cl-lib:fsqrt
                (abs (+ (* a a) (* (- four) b c)))))))))
(setf temp
      (+ rho
         (/ (f2cl-lib:fref z-%data% (1) ((1 3)) z-%offset%)
            (- (f2cl-lib:fref d-%data% (1) ((1 3)) d-%offset%)
```

```
                        tau))
                  (/ (f2cl-lib:fref z-%data% (2) ((1 3)) z-%offset%)
                     (- (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%)
                        tau))
                  (/ (f2cl-lib:fref z-%data% (3) ((1 3)) z-%offset%)
                     (- (f2cl-lib:fref d-%data% (3) ((1 3)) d-%offset%)
                        tau))))
   (if (<= (abs finit) (abs temp)) (setf tau zero))))
(cond
  (first$
   (setf eps (dlamch "Epsilon"))
   (setf base (dlamch "Base"))
   (setf small1
           (expt base
                 (f2cl-lib:int
                  (/
                   (/ (f2cl-lib:flog (dlamch "SafMin"))
                      (f2cl-lib:flog base))
                   three))))
   (setf sminv1 (/ one small1))
   (setf small2 (* small1 small1))
   (setf sminv2 (* sminv1 sminv1))
   (setf first$ nil)))
(cond
  (orgati
   (setf temp
           (min
            (abs
             (- (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%) tau))
            (abs
             (- (f2cl-lib:fref d-%data% (3) ((1 3)) d-%offset%)
                tau)))))
  (t
   (setf temp
           (min
            (abs
             (- (f2cl-lib:fref d-%data% (1) ((1 3)) d-%offset%) tau))
            (abs
             (- (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%)
                tau))))))
(setf scale nil)
(cond
  ((<= temp small1)
   (setf scale t)
   (cond
     ((<= temp small2)
```

```
          (setf sclfac sminv2)
          (setf sclinv small2))
         (t
          (setf sclfac sminv1)
          (setf sclinv small1)))
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i 3) nil)
         (tagbody
           (setf (f2cl-lib:fref dscale (i) ((1 3)))
                   (* (f2cl-lib:fref d-%data% (i) ((1 3)) d-%offset%)
                      sclfac))
           (setf (f2cl-lib:fref zscale (i) ((1 3)))
                   (* (f2cl-lib:fref z-%data% (i) ((1 3)) z-%offset%)
                      sclfac))))
       (setf tau (* tau sclfac)))
      (t
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i 3) nil)
         (tagbody
           (setf (f2cl-lib:fref dscale (i) ((1 3)))
                   (f2cl-lib:fref d-%data% (i) ((1 3)) d-%offset%))
           (setf (f2cl-lib:fref zscale (i) ((1 3)))
                   (f2cl-lib:fref z-%data% (i) ((1 3)) z-%offset%))))))
    (setf fc zero)
    (setf df zero)
    (setf ddf zero)
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i 3) nil)
      (tagbody
        (setf temp (/ one (- (f2cl-lib:fref dscale (i) ((1 3))) tau)))
        (setf temp1 (* (f2cl-lib:fref zscale (i) ((1 3))) temp))
        (setf temp2 (* temp1 temp))
        (setf temp3 (* temp2 temp))
        (setf fc (+ fc (/ temp1 (f2cl-lib:fref dscale (i) ((1 3))))))
        (setf df (+ df temp2))
        (setf ddf (+ ddf temp3))))
    (setf f (+ finit (* tau fc)))
    (if (<= (abs f) zero) (go label60))
    (setf iter (f2cl-lib:int-add niter 1))
    (f2cl-lib:fdo (niter iter (f2cl-lib:int-add niter 1))
                  ((> niter maxit) nil)
      (tagbody
        (cond
          (orgati
           (setf temp1 (- (f2cl-lib:fref dscale (2) ((1 3))) tau))
           (setf temp2 (- (f2cl-lib:fref dscale (3) ((1 3))) tau)))
```

```
  (t
   (setf temp1 (- (f2cl-lib:fref dscale (1) ((1 3))) tau))
   (setf temp2 (- (f2cl-lib:fref dscale (2) ((1 3))) tau))))
(setf a (+ (* (+ temp1 temp2) f) (* (- temp1) temp2 df)))
(setf b (* temp1 temp2 f))
(setf c (+ (- f (* (+ temp1 temp2) df)) (* temp1 temp2 ddf)))
(setf temp (max (abs a) (abs b) (abs c)))
(setf a (/ a temp))
(setf b (/ b temp))
(setf c (/ c temp))
(cond
  ((= c zero)
   (setf eta (/ b a)))
  ((<= a zero)
   (setf eta
           (/
            (- a
               (f2cl-lib:fsqrt
                (abs (+ (* a a) (* (- four) b c)))))
            (* two c))))
  (t
   (setf eta
           (/ (* two b)
              (+ a
                 (f2cl-lib:fsqrt
                  (abs (+ (* a a) (* (- four) b c)))))))))
(cond
  ((>= (* f eta) zero)
   (setf eta (/ (- f) df))))
(setf temp (+ eta tau))
(cond
  (orgati
   (if
    (and (> eta zero)
         (>= temp (f2cl-lib:fref dscale (3) ((1 3)))))
    (setf eta (/ (- (f2cl-lib:fref dscale (3) ((1 3))) tau) two)))
   (if
    (and (< eta zero)
         (<= temp (f2cl-lib:fref dscale (2) ((1 3)))))
    (setf eta
           (/ (- (f2cl-lib:fref dscale (2) ((1 3))) tau) two))))
  (t
   (if
    (and (> eta zero)
         (>= temp (f2cl-lib:fref dscale (2) ((1 3)))))
    (setf eta (/ (- (f2cl-lib:fref dscale (2) ((1 3))) tau) two)))
```

```
                  (if
                   (and (< eta zero)
                        (<= temp (f2cl-lib:fref dscale (1) ((1 3)))))
                   (setf eta
                           (/ (- (f2cl-lib:fref dscale (1) ((1 3))) tau)
                              two)))))
               (setf tau (+ tau eta))
               (setf fc zero)
               (setf erretm zero)
               (setf df zero)
               (setf ddf zero)
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i 3) nil)
                 (tagbody
                   (setf temp
                           (/ one (- (f2cl-lib:fref dscale (i) ((1 3))) tau)))
                   (setf temp1 (* (f2cl-lib:fref zscale (i) ((1 3))) temp))
                   (setf temp2 (* temp1 temp))
                   (setf temp3 (* temp2 temp))
                   (setf temp4 (/ temp1 (f2cl-lib:fref dscale (i) ((1 3)))))
                   (setf fc (+ fc temp4))
                   (setf erretm (+ erretm (abs temp4)))
                   (setf df (+ df temp2))
                   (setf ddf (+ ddf temp3))))
               (setf f (+ finit (* tau fc)))
               (setf erretm
                       (+ (* eight (+ (abs finit) (* (abs tau) erretm)))
                          (* (abs tau) df)))
               (if (<= (abs f) (* eps erretm)) (go label60))))
          (setf info 1)
label60
          (if scale (setf tau (* tau sclinv)))
end_label
          (return (values nil nil nil nil nil nil tau info)))))))
```

7.29 dlaexc LAPACK

⟨*dlaexc.input*⟩≡

```
)set break resume
)sys rm -f dlaexc.output
)spool dlaexc.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlaexc.help*⟩≡

```
  ====================================================================
  dlaexc examples
  ====================================================================

  ====================================================================
  Man Page Details
  ====================================================================

  NAME
         DLAEXC  -  adjacent  diagonal  blocks T11 and T22 of order 1 or 2 in an
         upper quasi-triangular matrix T by an orthogonal similarity transforma-
         tion

  SYNOPSIS
         SUBROUTINE DLAEXC( WANTQ, N, T, LDT, Q, LDQ, J1, N1, N2, WORK, INFO )

             LOGICAL         WANTQ

             INTEGER         INFO, J1, LDQ, LDT, N, N1, N2

             DOUBLE          PRECISION Q( LDQ, * ), T( LDT, * ), WORK( * )

  PURPOSE
         DLAEXC swaps adjacent diagonal blocks T11 and T22 of order 1 or 2 in an
         upper quasi-triangular matrix T by an orthogonal similarity transforma-
         tion.

         T must be in Schur canonical form, that is, block upper triangular with
         1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block  has  its
         diagonal  elemnts equal and its off-diagonal elements of opposite sign.


  ARGUMENTS
         WANTQ   (input) LOGICAL
                 = .TRUE. : accumulate the transformation in the matrix Q;
                 = .FALSE.: do not accumulate the transformation.

         N       (input) INTEGER
                 The order of the matrix T. N >= 0.

         T       (input/output) DOUBLE PRECISION array, dimension (LDT,N)
                 On entry, the upper quasi-triangular matrix T, in Schur canoni-
                 cal form.  On exit, the updated matrix T, again in Schur canon-
                 ical form.
```

```
LDT     (input)  INTEGER
        The leading dimension of the array T. LDT >= max(1,N).

Q       (input/output) DOUBLE PRECISION array, dimension (LDQ,N)
        On entry, if WANTQ is .TRUE., the orthogonal matrix Q.  On
        exit,  if  WANTQ  is .TRUE., the updated matrix Q.  If WANTQ is
        .FALSE., Q is not referenced.

LDQ     (input) INTEGER
        The leading dimension of the array Q.  LDQ >= 1; and  if  WANTQ
        is .TRUE., LDQ >= N.

J1      (input) INTEGER
        The index of the first row of the first block T11.

N1      (input) INTEGER
        The order of the first block T11. N1 = 0, 1 or 2.

N2      (input) INTEGER
        The order of the second block T22. N2 = 0, 1 or 2.

WORK    (workspace) DOUBLE PRECISION array, dimension (N)

INFO    (output) INTEGER
        = 0: successful exit
        = 1: the transformed matrix T would be too far from Schur form;
        the blocks are not swapped and T and Q are unchanged.
```

⟨*LAPACK dlaexc*⟩≡

```
(let* ((zero 0.0) (one 1.0) (ten 10.0) (ldd 4) (ldx 2))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 10.0 10.0) ten)
           (type (fixnum 4 4) ldd)
           (type (fixnum 2 2) ldx))
  (defun dlaexc (wantq n t$ ldt q ldq j1 n1 n2 work info)
    (declare (type (simple-array double-float (*)) work q t$)
             (type fixnum info n2 n1 j1 ldq ldt n)
             (type (member t nil) wantq))
    (f2cl-lib:with-multi-array-data
        ((t$ double-float t$-%data% t$-%offset%)
         (q double-float q-%data% q-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((d
              (make-array (the fixnum (reduce #'* (list ldd 4)))
                          :element-type 'double-float))
             (u (make-array 3 :element-type 'double-float))
             (u1 (make-array 3 :element-type 'double-float))
             (u2 (make-array 3 :element-type 'double-float))
             (x
              (make-array (the fixnum (reduce #'* (list ldx 2)))
                          :element-type 'double-float))
             (cs 0.0) (dnorm 0.0) (eps 0.0) (scale 0.0) (smlnum 0.0) (sn 0.0)
             (t11 0.0) (t22 0.0) (t33 0.0) (tau 0.0) (tau1 0.0) (tau2 0.0)
             (temp 0.0) (thresh 0.0) (wi1 0.0) (wi2 0.0) (wr1 0.0) (wr2 0.0)
             (xnorm 0.0) (ierr 0) (j2 0) (j3 0) (j4 0) (k 0) (nd 0))
        (declare (type (simple-array double-float (3)) u u1 u2)
                 (type (simple-array double-float (*)) d x)
                 (type (double-float) cs dnorm eps scale smlnum sn t11 t22 t33
                                      tau tau1 tau2 temp thresh wi1 wi2 wr1 wr2
                                      xnorm)
                 (type fixnum ierr j2 j3 j4 k nd))
        (setf info 0)
        (if (or (= n 0) (= n1 0) (= n2 0)) (go end_label))
        (if (> (f2cl-lib:int-add j1 n1) n) (go end_label))
        (setf j2 (f2cl-lib:int-add j1 1))
        (setf j3 (f2cl-lib:int-add j1 2))
        (setf j4 (f2cl-lib:int-add j1 3))
        (cond
          ((and (= n1 1) (= n2 1))
           (setf t11
                   (f2cl-lib:fref t$-%data%
                                 (j1 j1)
                                 ((1 ldt) (1 *))
```

```
                        t$-%offset%))
 (setf t22
        (f2cl-lib:fref t$-%data%
                       (j2 j2)
                       ((1 ldt) (1 *))
                       t$-%offset%))
 (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
     (dlartg
      (f2cl-lib:fref t$-%data% (j1 j2) ((1 ldt) (1 *)) t$-%offset%)
      (- t22 t11) cs sn temp)
   (declare (ignore var-0 var-1))
   (setf cs var-2)
   (setf sn var-3)
   (setf temp var-4))
 (if (<= j3 n)
     (drot (f2cl-lib:int-sub n j1 1)
      (f2cl-lib:array-slice t$ double-float (j1 j3) ((1 ldt) (1 *)))
      ldt
      (f2cl-lib:array-slice t$ double-float (j2 j3) ((1 ldt) (1 *)))
      ldt cs sn))
 (drot (f2cl-lib:int-sub j1 1)
  (f2cl-lib:array-slice t$ double-float (1 j1) ((1 ldt) (1 *))) 1
  (f2cl-lib:array-slice t$ double-float (1 j2) ((1 ldt) (1 *))) 1 cs
  sn)
 (setf (f2cl-lib:fref t$-%data% (j1 j1) ((1 ldt) (1 *)) t$-%offset%)
        t22)
 (setf (f2cl-lib:fref t$-%data% (j2 j2) ((1 ldt) (1 *)) t$-%offset%)
        t11)
 (cond
   (wantq
    (drot n
     (f2cl-lib:array-slice q double-float (1 j1) ((1 ldq) (1 *))) 1
     (f2cl-lib:array-slice q double-float (1 j2) ((1 ldq) (1 *))) 1
     cs sn))))
(t
 (tagbody
   (setf nd (f2cl-lib:int-add n1 n2))
   (dlacpy "Full" nd nd
    (f2cl-lib:array-slice t$ double-float (j1 j1) ((1 ldt) (1 *)))
    ldt d ldd)
   (setf dnorm (dlange "Max" nd nd d ldd work))
   (setf eps (dlamch "P"))
   (setf smlnum (/ (dlamch "S") eps))
   (setf thresh (max (* ten eps dnorm) smlnum))
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
```

```
                  var-10 var-11 var-12 var-13 var-14 var-15)
                (dlasy2 nil nil -1 n1 n2 d ldd
                 (f2cl-lib:array-slice d
                                       double-float
                                       ((+ n1 1) (f2cl-lib:int-add n1 1))
                                       ((1 ldd) (1 4)))
                 ldd
                 (f2cl-lib:array-slice d
                                       double-float
                                       (1 (f2cl-lib:int-add n1 1))
                                       ((1 ldd) (1 4)))
                 ldd scale x ldx xnorm ierr)
               (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                                var-8 var-9 var-10 var-12 var-13))
               (setf scale var-11)
               (setf xnorm var-14)
               (setf ierr var-15))
             (setf k (f2cl-lib:int-sub (f2cl-lib:int-add n1 n1 n2) 3))
             (f2cl-lib:computed-goto (label10 label20 label30) k)
label10
             (setf (f2cl-lib:fref u (1) ((1 3))) scale)
             (setf (f2cl-lib:fref u (2) ((1 3)))
                     (f2cl-lib:fref x (1 1) ((1 ldx) (1 2))))
             (setf (f2cl-lib:fref u (3) ((1 3)))
                     (f2cl-lib:fref x (1 2) ((1 ldx) (1 2))))
             (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                 (dlarfg 3 (f2cl-lib:fref u (3) ((1 3))) u 1 tau)
               (declare (ignore var-0 var-2 var-3))
               (setf (f2cl-lib:fref u (3) ((1 3))) var-1)
               (setf tau var-4))
             (setf (f2cl-lib:fref u (3) ((1 3))) one)
             (setf t11
                     (f2cl-lib:fref t$-%data%
                                    (j1 j1)
                                    ((1 ldt) (1 *))
                                    t$-%offset%))
             (dlarfx "L" 3 3 u tau d ldd work)
             (dlarfx "R" 3 3 u tau d ldd work)
             (if
              (>
               (max (abs (f2cl-lib:fref d (3 1) ((1 ldd) (1 4))))
                    (abs (f2cl-lib:fref d (3 2) ((1 ldd) (1 4))))
                    (abs (- (f2cl-lib:fref d (3 3) ((1 ldd) (1 4))) t11)))
               thresh)
              (go label50))
             (dlarfx "L" 3 (f2cl-lib:int-add (f2cl-lib:int-sub n j1) 1) u tau
```

```
                 (f2cl-lib:array-slice t$ double-float (j1 j1) ((1 ldt) (1 *)))
                 ldt work)
                (dlarfx "R" j2 3 u tau
                 (f2cl-lib:array-slice t$ double-float (1 j1) ((1 ldt) (1 *))) ldt
                 work)
                (setf (f2cl-lib:fref t$-%data%
                                     (j3 j1)
                                     ((1 ldt) (1 *))
                                     t$-%offset%)
                        zero)
                (setf (f2cl-lib:fref t$-%data%
                                     (j3 j2)
                                     ((1 ldt) (1 *))
                                     t$-%offset%)
                        zero)
                (setf (f2cl-lib:fref t$-%data%
                                     (j3 j3)
                                     ((1 ldt) (1 *))
                                     t$-%offset%)
                        t11)
                (cond
                  (wantq
                   (dlarfx "R" n 3 u tau
                    (f2cl-lib:array-slice q double-float (1 j1) ((1 ldq) (1 *)))
                    ldq work)))
                (go label40)
  label120
                (setf (f2cl-lib:fref u (1) ((1 3)))
                        (- (f2cl-lib:fref x (1 1) ((1 ldx) (1 2)))))
                (setf (f2cl-lib:fref u (2) ((1 3)))
                        (- (f2cl-lib:fref x (2 1) ((1 ldx) (1 2)))))
                (setf (f2cl-lib:fref u (3) ((1 3))) scale)
                (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                    (dlarfg 3 (f2cl-lib:fref u (1) ((1 3)))
                     (f2cl-lib:array-slice u double-float (2) ((1 3))) 1 tau)
                  (declare (ignore var-0 var-2 var-3))
                  (setf (f2cl-lib:fref u (1) ((1 3))) var-1)
                  (setf tau var-4))
                (setf (f2cl-lib:fref u (1) ((1 3))) one)
                (setf t33
                        (f2cl-lib:fref t$-%data%
                                       (j3 j3)
                                       ((1 ldt) (1 *))
                                       t$-%offset%))
                (dlarfx "L" 3 3 u tau d ldd work)
                (dlarfx "R" 3 3 u tau d ldd work)
```

```
              (if
               (>
                (max (abs (f2cl-lib:fref d (2 1) ((1 ldd) (1 4))))
                     (abs (f2cl-lib:fref d (3 1) ((1 ldd) (1 4))))
                     (abs (- (f2cl-lib:fref d (1 1) ((1 ldd) (1 4))) t33)))
                thresh)
               (go label50))
              (dlarfx "R" j3 3 u tau
               (f2cl-lib:array-slice t$ double-float (1 j1) ((1 ldt) (1 *))) ldt
               work)
              (dlarfx "L" 3 (f2cl-lib:int-sub n j1) u tau
               (f2cl-lib:array-slice t$ double-float (j1 j2) ((1 ldt) (1 *)))
               ldt work)
              (setf (f2cl-lib:fref t$-%data%
                                   (j1 j1)
                                   ((1 ldt) (1 *))
                                   t$-%offset%)
                    t33)
              (setf (f2cl-lib:fref t$-%data%
                                   (j2 j1)
                                   ((1 ldt) (1 *))
                                   t$-%offset%)
                    zero)
              (setf (f2cl-lib:fref t$-%data%
                                   (j3 j1)
                                   ((1 ldt) (1 *))
                                   t$-%offset%)
                    zero)
              (cond
                (wantq
                 (dlarfx "R" n 3 u tau
                  (f2cl-lib:array-slice q double-float (1 j1) ((1 ldq) (1 *)))
                  ldq work)))
              (go label40)
label30
              (setf (f2cl-lib:fref u1 (1) ((1 3)))
                    (- (f2cl-lib:fref x (1 1) ((1 ldx) (1 2)))))
              (setf (f2cl-lib:fref u1 (2) ((1 3)))
                    (- (f2cl-lib:fref x (2 1) ((1 ldx) (1 2)))))
              (setf (f2cl-lib:fref u1 (3) ((1 3))) scale)
              (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                  (dlarfg 3 (f2cl-lib:fref u1 (1) ((1 3)))
                   (f2cl-lib:array-slice u1 double-float (2) ((1 3))) 1 tau1)
                (declare (ignore var-0 var-2 var-3))
                (setf (f2cl-lib:fref u1 (1) ((1 3))) var-1)
                (setf tau1 var-4))
```

```
(setf (f2cl-lib:fref u1 (1) ((1 3))) one)
(setf temp
        (* (- tau1)
           (+ (f2cl-lib:fref x (1 2) ((1 ldx) (1 2)))
              (* (f2cl-lib:fref u1 (2) ((1 3)))
                 (f2cl-lib:fref x (2 2) ((1 ldx) (1 2)))))))
(setf (f2cl-lib:fref u2 (1) ((1 3)))
        (- (* (- temp) (f2cl-lib:fref u1 (2) ((1 3))))
           (f2cl-lib:fref x (2 2) ((1 ldx) (1 2)))))
(setf (f2cl-lib:fref u2 (2) ((1 3)))
        (* (- temp) (f2cl-lib:fref u1 (3) ((1 3)))))
(setf (f2cl-lib:fref u2 (3) ((1 3))) scale)
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlarfg 3 (f2cl-lib:fref u2 (1) ((1 3)))
     (f2cl-lib:array-slice u2 double-float (2) ((1 3))) 1 tau2)
  (declare (ignore var-0 var-2 var-3))
  (setf (f2cl-lib:fref u2 (1) ((1 3))) var-1)
  (setf tau2 var-4))
(setf (f2cl-lib:fref u2 (1) ((1 3))) one)
(dlarfx "L" 3 4 u1 tau1 d ldd work)
(dlarfx "R" 4 3 u1 tau1 d ldd work)
(dlarfx "L" 3 4 u2 tau2
 (f2cl-lib:array-slice d double-float (2 1) ((1 ldd) (1 4))) ldd
 work)
(dlarfx "R" 4 3 u2 tau2
 (f2cl-lib:array-slice d double-float (1 2) ((1 ldd) (1 4))) ldd
 work)
(if
 (>
  (max (abs (f2cl-lib:fref d (3 1) ((1 ldd) (1 4))))
       (abs (f2cl-lib:fref d (3 2) ((1 ldd) (1 4))))
       (abs (f2cl-lib:fref d (4 1) ((1 ldd) (1 4))))
       (abs (f2cl-lib:fref d (4 2) ((1 ldd) (1 4)))))
  thresh)
 (go label50))
(dlarfx "L" 3 (f2cl-lib:int-add (f2cl-lib:int-sub n j1) 1) u1 tau1
 (f2cl-lib:array-slice t$ double-float (j1 j1) ((1 ldt) (1 *)))
 ldt work)
(dlarfx "R" j4 3 u1 tau1
 (f2cl-lib:array-slice t$ double-float (1 j1) ((1 ldt) (1 *))) ldt
 work)
(dlarfx "L" 3 (f2cl-lib:int-add (f2cl-lib:int-sub n j1) 1) u2 tau2
 (f2cl-lib:array-slice t$ double-float (j2 j1) ((1 ldt) (1 *)))
 ldt work)
(dlarfx "R" j4 3 u2 tau2
 (f2cl-lib:array-slice t$ double-float (1 j2) ((1 ldt) (1 *))) ldt
```

```
                 work)
                (setf (f2cl-lib:fref t$-%data%
                                     (j3 j1)
                                     ((1 ldt) (1 *))
                                     t$-%offset%)
                      zero)
                (setf (f2cl-lib:fref t$-%data%
                                     (j3 j2)
                                     ((1 ldt) (1 *))
                                     t$-%offset%)
                      zero)
                (setf (f2cl-lib:fref t$-%data%
                                     (j4 j1)
                                     ((1 ldt) (1 *))
                                     t$-%offset%)
                      zero)
                (setf (f2cl-lib:fref t$-%data%
                                     (j4 j2)
                                     ((1 ldt) (1 *))
                                     t$-%offset%)
                      zero)
                (cond
                  (wantq
                   (dlarfx "R" n 3 u1 tau1
                    (f2cl-lib:array-slice q double-float (1 j1) ((1 ldq) (1 *)))
                    ldq work)
                   (dlarfx "R" n 3 u2 tau2
                    (f2cl-lib:array-slice q double-float (1 j2) ((1 ldq) (1 *)))
                    ldq work)))
  label40
                (cond
                  ((= n2 2)
                   (multiple-value-bind
                         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                          var-9)
                       (dlanv2
                        (f2cl-lib:fref t$-%data%
                                       (j1 j1)
                                       ((1 ldt) (1 *))
                                       t$-%offset%)
                        (f2cl-lib:fref t$-%data%
                                       (j1 j2)
                                       ((1 ldt) (1 *))
                                       t$-%offset%)
                        (f2cl-lib:fref t$-%data%
                                       (j2 j1)
```

```
                    ((1 ldt) (1 *))
                    t$-%offset%)
      (f2cl-lib:fref t$-%data%
                    (j2 j2)
                    ((1 ldt) (1 *))
                    t$-%offset%)
      wr1 wi1 wr2 wi2 cs sn)
   (declare (ignore))
   (setf (f2cl-lib:fref t$-%data%
                        (j1 j1)
                        ((1 ldt) (1 *))
                        t$-%offset%)
          var-0)
   (setf (f2cl-lib:fref t$-%data%
                        (j1 j2)
                        ((1 ldt) (1 *))
                        t$-%offset%)
          var-1)
   (setf (f2cl-lib:fref t$-%data%
                        (j2 j1)
                        ((1 ldt) (1 *))
                        t$-%offset%)
          var-2)
   (setf (f2cl-lib:fref t$-%data%
                        (j2 j2)
                        ((1 ldt) (1 *))
                        t$-%offset%)
          var-3)
   (setf wr1 var-4)
   (setf wi1 var-5)
   (setf wr2 var-6)
   (setf wi2 var-7)
   (setf cs var-8)
   (setf sn var-9))
 (drot (f2cl-lib:int-sub n j1 1)
  (f2cl-lib:array-slice t$
                        double-float
                        (j1 (f2cl-lib:int-add j1 2))
                        ((1 ldt) (1 *)))
  ldt
  (f2cl-lib:array-slice t$
                        double-float
                        (j2 (f2cl-lib:int-add j1 2))
                        ((1 ldt) (1 *)))
  ldt cs sn)
 (drot (f2cl-lib:int-sub j1 1)
```

```
          (f2cl-lib:array-slice t$ double-float (1 j1) ((1 ldt) (1 *)))
          1
          (f2cl-lib:array-slice t$ double-float (1 j2) ((1 ldt) (1 *)))
          1 cs sn)
         (if wantq
             (drot n
              (f2cl-lib:array-slice q
                                    double-float
                                    (1 j1)
                                    ((1 ldq) (1 *)))
              1
              (f2cl-lib:array-slice q
                                    double-float
                                    (1 j2)
                                    ((1 ldq) (1 *)))
              1 cs sn))))
    (cond
      ((= n1 2)
       (setf j3 (f2cl-lib:int-add j1 n2))
       (setf j4 (f2cl-lib:int-add j3 1))
       (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
             var-9)
           (dlanv2
            (f2cl-lib:fref t$-%data%
                           (j3 j3)
                           ((1 ldt) (1 *))
                           t$-%offset%)
            (f2cl-lib:fref t$-%data%
                           (j3 j4)
                           ((1 ldt) (1 *))
                           t$-%offset%)
            (f2cl-lib:fref t$-%data%
                           (j4 j3)
                           ((1 ldt) (1 *))
                           t$-%offset%)
            (f2cl-lib:fref t$-%data%
                           (j4 j4)
                           ((1 ldt) (1 *))
                           t$-%offset%)
            wr1 wi1 wr2 wi2 cs sn)
          (declare (ignore))
          (setf (f2cl-lib:fref t$-%data%
                               (j3 j3)
                               ((1 ldt) (1 *))
                               t$-%offset%)
```

```
          var-0)
  (setf (f2cl-lib:fref t$-%data%
                       (j3 j4)
                       ((1 ldt) (1 *))
                       t$-%offset%)
          var-1)
  (setf (f2cl-lib:fref t$-%data%
                       (j4 j3)
                       ((1 ldt) (1 *))
                       t$-%offset%)
          var-2)
  (setf (f2cl-lib:fref t$-%data%
                       (j4 j4)
                       ((1 ldt) (1 *))
                       t$-%offset%)
          var-3)
  (setf wr1 var-4)
  (setf wi1 var-5)
  (setf wr2 var-6)
  (setf wi2 var-7)
  (setf cs var-8)
  (setf sn var-9))
(if (<= (f2cl-lib:int-add j3 2) n)
    (drot (f2cl-lib:int-sub n j3 1)
     (f2cl-lib:array-slice t$
                           double-float
                           (j3 (f2cl-lib:int-add j3 2))
                           ((1 ldt) (1 *)))
     ldt
     (f2cl-lib:array-slice t$
                           double-float
                           (j4 (f2cl-lib:int-add j3 2))
                           ((1 ldt) (1 *)))
     ldt cs sn))
(drot (f2cl-lib:int-sub j3 1)
 (f2cl-lib:array-slice t$ double-float (1 j3) ((1 ldt) (1 *)))
 1
 (f2cl-lib:array-slice t$ double-float (1 j4) ((1 ldt) (1 *)))
 1 cs sn)
(if wantq
    (drot n
     (f2cl-lib:array-slice q
                           double-float
                           (1 j3)
                           ((1 ldq) (1 *)))
     1
```

```
                    (f2cl-lib:array-slice q
                                          double-float
                                          (1 j4)
                                          ((1 ldq) (1 *)))
                    1 cs sn)))))))
        (go end_label)
  label50
        (setf info 1)
  end_label
        (return (values nil nil nil nil nil nil nil nil nil nil info))))))
```

7.30 dlahqr LAPACK

⟨*dlahqr.input*⟩≡

```
  )set break resume
  )sys rm -f dlahqr.output
  )spool dlahqr.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dlahqr.help*⟩≡

```
==========================================================================
dlahqr examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

NAME
       DLAHQR - An auxiliary routine called by DHSEQR to update the  eigenval-
       ues and Schur decomposition already computed  by  DHSEQR,  by   dealing
       with the Hessenberg submatrix in rows and columns ILO to IHI

SYNOPSIS
       SUBROUTINE DLAHQR( WANTT,  WANTZ,  N,  ILO,  IHI, H, LDH, WR, WI, ILOZ,
                          IHIZ, Z, LDZ, INFO )

           INTEGER        IHI, IHIZ, ILO, ILOZ, INFO, LDH, LDZ, N

           LOGICAL        WANTT, WANTZ

           DOUBLE         PRECISION H( LDH, * ), WI( * ), WR( * ), Z( LDZ, * )

PURPOSE
          DLAHQR is an auxiliary routine called by DHSEQR to update the
          eigenvalues and Schur decomposition already computed by DHSEQR, by
          dealing with the Hessenberg submatrix in rows and columns ILO to
          IHI.


ARGUMENTS
       WANTT   (input) LOGICAL
               = .TRUE. : the full Schur form T is required;
               = .FALSE.: only eigenvalues are required.

       WANTZ   (input) LOGICAL
               = .TRUE. : the matrix of Schur vectors Z is required;
               = .FALSE.: Schur vectors are not required.

       N       (input) INTEGER
               The order of the matrix H.  N >= 0.

       ILO     (input) INTEGER
               IHI      (input)  INTEGER It is assumed that H is already upper
               quasi-triangular  in  rows  and  columns  IHI+1:N,   and   that
```

H(ILO,ILO-1) = 0 (unless ILO = 1). DLAHQR works primarily with the Hessenberg submatrix in rows and columns ILO to IHI, but applies transformations to all of H if WANTT is .TRUE.. 1 <= ILO <= max(1,IHI); IHI <= N.

H (input/output) DOUBLE PRECISION array, dimension (LDH,N)
On entry, the upper Hessenberg matrix H. On exit, if INFO is zero and if WANTT is .TRUE., H is upper quasi-triangular in rows and columns ILO:IHI, with any 2-by-2 diagonal blocks in standard form. If INFO is zero and WANTT is .FALSE., the contents of H are unspecified on exit. The output state of H if INFO is nonzero is given below under the description of INFO.

LDH (input) INTEGER
The leading dimension of the array H. LDH >= max(1,N).

WR (output) DOUBLE PRECISION array, dimension (N)
WI (output) DOUBLE PRECISION array, dimension (N) The real and imaginary parts, respectively, of the computed eigenvalues ILO to IHI are stored in the corresponding elements of WR and WI. If two eigenvalues are computed as a complex conjugate pair, they are stored in consecutive elements of WR and WI, say the i-th and (i+1)th, with WI(i) > 0 and WI(i+1) < 0. If WANTT is .TRUE., the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with WR(i) = H(i,i), and, if H(i:i+1,i:i+1) is a 2-by-2 diagonal block, WI(i) = sqrt(H(i+1,i)*H(i,i+1)) and WI(i+1) = -WI(i).

ILOZ (input) INTEGER
IHIZ (input) INTEGER Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 <= ILOZ <= ILO; IHI <= IHIZ <= N.

Z (input/output) DOUBLE PRECISION array, dimension (LDZ,N)
If WANTZ is .TRUE., on entry Z must contain the current matrix Z of transformations accumulated by DHSEQR, and on exit Z has been updated; transformations are applied only to the submatrix Z(ILOZ:IHIZ,ILO:IHI). If WANTZ is .FALSE., Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ >= max(1,N).

INFO (output) INTEGER
= 0: successful exit
eigenvalues ILO to IHI in a total of 30 iterations per eigen-

```
value;  elements i+1:ihi of WR and WI contain those eigenvalues
which have been successfully computed.

If INFO .GT. 0 and WANTT is .FALSE., then on exit, the  remain-
ing  unconverged  eigenvalues  are the eigenvalues of the upper
Hessenberg matrix rows and columns  ILO  thorugh  INFO  of  the
final, output value of H.

If  INFO  .GT.  0  and  WANTT  is  .TRUE., then on exit (*)
(initial value of H)*U  = U*(final value of H) where  U  is  an
orthognal  matrix.     The final value of H is upper Hessenberg
and triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit  (final  value
of  Z)  =  (initial  value  of  Z)*U where U is the orthogonal
matrix in (*) (regardless of the value of WANTT.)
```

⟨*LAPACK dlahqr*⟩≡

```
(let* ((zero 0.0) (one 1.0) (half 0.5) (dat1 0.75) (dat2 (- 0.4375)))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 0.5 0.5) half)
           (type (double-float 0.75 0.75) dat1)
           (type (double-float) dat2))
  (defun dlahqr (wantt wantz n ilo ihi h ldh wr wi iloz ihiz z ldz info)
    (declare (type (simple-array double-float (*)) z wi wr h)
             (type fixnum info ldz ihiz iloz ldh ihi ilo n)
             (type (member t nil) wantz wantt))
    (f2cl-lib:with-multi-array-data
        ((h double-float h-%data% h-%offset%)
         (wr double-float wr-%data% wr-%offset%)
         (wi double-float wi-%data% wi-%offset%)
         (z double-float z-%data% z-%offset%))
      (prog ((v (make-array 3 :element-type 'double-float))
             (work (make-array 1 :element-type 'double-float)) (ave 0.0)
             (cs 0.0) (disc 0.0) (h00 0.0) (h10 0.0) (h11 0.0) (h12 0.0)
             (h21 0.0) (h22 0.0) (h33 0.0) (h33s 0.0) (h43h34 0.0) (h44 0.0)
             (h44s 0.0) (ovfl 0.0) (s 0.0) (smlnum 0.0) (sn 0.0) (sum 0.0)
             (t1 0.0) (t2 0.0) (t3 0.0) (tst1 0.0) (ulp 0.0) (unfl 0.0)
             (v1 0.0) (v2 0.0) (v3 0.0) (i 0) (i1 0) (i2 0) (itn 0) (its 0)
             (j 0) (k 0) (l 0) (m 0) (nh 0) (nr 0) (nz 0))
        (declare (type (simple-array double-float (3)) v)
                 (type (simple-array double-float (1)) work)
                 (type (double-float) ave cs disc h00 h10 h11 h12 h21 h22 h33
                                      h33s h43h34 h44 h44s ovfl s smlnum sn sum
                                      t1 t2 t3 tst1 ulp unfl v1 v2 v3)
                 (type fixnum i i1 i2 itn its j k l m nh nr nz))
        (setf info 0)
        (if (= n 0) (go end_label))
        (cond
          ((= ilo ihi)
           (setf (f2cl-lib:fref wr-%data% (ilo) ((1 *)) wr-%offset%)
                   (f2cl-lib:fref h-%data%
                                  (ilo ilo)
                                  ((1 ldh) (1 *))
                                  h-%offset%))
           (setf (f2cl-lib:fref wi-%data% (ilo) ((1 *)) wi-%offset%) zero)
           (go end_label)))
        (setf nh (f2cl-lib:int-add (f2cl-lib:int-sub ihi ilo) 1))
        (setf nz (f2cl-lib:int-add (f2cl-lib:int-sub ihiz iloz) 1))
        (setf unfl (dlamch "Safe minimum"))
        (setf ovfl (/ one unfl))
        (multiple-value-bind (var-0 var-1)
```

```
            (dlabad unfl ovfl)
          (declare (ignore))
          (setf unfl var-0)
          (setf ovfl var-1))
        (setf ulp (dlamch "Precision"))
        (setf smlnum (* unfl (/ nh ulp)))
        (cond
          (wantt
           (setf i1 1)
           (setf i2 n)))
        (setf itn (f2cl-lib:int-mul 30 nh))
        (setf i ihi)
label10
        (setf l ilo)
        (if (< i ilo) (go end_label))
        (f2cl-lib:fdo (its 0 (f2cl-lib:int-add its 1))
                      ((> its itn) nil)
          (tagbody
            (f2cl-lib:fdo (k i (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
                          ((> k (f2cl-lib:int-add l 1)) nil)
              (tagbody
                (setf tst1
                        (+
                         (abs
                          (f2cl-lib:fref h-%data%
                                         ((f2cl-lib:int-sub k 1)
                                          (f2cl-lib:int-sub k 1))
                                         ((1 ldh) (1 *))
                                         h-%offset%))
                         (abs
                          (f2cl-lib:fref h-%data%
                                         (k k)
                                         ((1 ldh) (1 *))
                                         h-%offset%))))
                (if (= tst1 zero)
                    (setf tst1
                            (dlanhs "1"
                             (f2cl-lib:int-add (f2cl-lib:int-sub i l) 1)
                             (f2cl-lib:array-slice h
                                                   double-float
                                                   (l l)
                                                   ((1 ldh) (1 *)))
                             ldh work)))
                (if
                 (<=
                  (abs
```

```
                    (f2cl-lib:fref h-%data%
                                   (k (f2cl-lib:int-sub k 1))
                                   ((1 ldh) (1 *))
                                   h-%offset%))
                   (max (* ulp tst1) smlnum))
                  (go label30))))
  label30
              (setf l k)
              (cond
                ((> l ilo)
                 (setf (f2cl-lib:fref h-%data%
                                      (l (f2cl-lib:int-sub l 1))
                                      ((1 ldh) (1 *))
                                      h-%offset%)
                       zero)))
              (if (>= l (f2cl-lib:int-sub i 1)) (go label140))
              (cond
                ((not wantt)
                 (setf i1 l)
                 (setf i2 i)))
              (cond
                ((or (= its 10) (= its 20))
                 (setf s
                       (+
                        (abs
                         (f2cl-lib:fref h-%data%
                                        (i (f2cl-lib:int-sub i 1))
                                        ((1 ldh) (1 *))
                                        h-%offset%))
                        (abs
                         (f2cl-lib:fref h-%data%
                                        ((f2cl-lib:int-sub i 1)
                                         (f2cl-lib:int-sub i 2))
                                        ((1 ldh) (1 *))
                                        h-%offset%))))
                 (setf h44
                       (+ (* dat1 s)
                          (f2cl-lib:fref h-%data%
                                         (i i)
                                         ((1 ldh) (1 *))
                                         h-%offset%)))
                 (setf h33 h44)
                 (setf h43h34 (* dat2 s s)))
                (t
                 (setf h44
                       (f2cl-lib:fref h-%data%
```

```
                                (i i)
                                ((1 ldh) (1 *))
                                h-%offset%))
                        (setf h33
                                (f2cl-lib:fref h-%data%
                                ((f2cl-lib:int-sub i 1)
                                 (f2cl-lib:int-sub i 1))
                                ((1 ldh) (1 *))
                                h-%offset%))
                        (setf h43h34
                                (*
                                 (f2cl-lib:fref h-%data%
                                  (i (f2cl-lib:int-sub i 1))
                                  ((1 ldh) (1 *))
                                  h-%offset%)
                                 (f2cl-lib:fref h-%data%
                                  ((f2cl-lib:int-sub i 1) i)
                                  ((1 ldh) (1 *))
                                  h-%offset%)))
                        (setf s
                                (*
                                 (f2cl-lib:fref h-%data%
                                  ((f2cl-lib:int-sub i 1)
                                   (f2cl-lib:int-sub i 2))
                                  ((1 ldh) (1 *))
                                  h-%offset%)
                                 (f2cl-lib:fref h-%data%
                                  ((f2cl-lib:int-sub i 1)
                                   (f2cl-lib:int-sub i 2))
                                  ((1 ldh) (1 *))
                                  h-%offset%)))
                        (setf disc (* (- h33 h44) half))
                        (setf disc (+ (* disc disc) h43h34))
                        (cond
                          ((> disc zero)
                           (setf disc (f2cl-lib:fsqrt disc))
                           (setf ave (* half (+ h33 h44)))
                           (cond
                             ((> (+ (abs h33) (- (abs h44))) zero)
                              (setf h33 (- (* h33 h44) h43h34))
                              (setf h44 (/ h33 (+ (f2cl-lib:sign disc ave) ave))))
                             (t
                              (setf h44 (+ (f2cl-lib:sign disc ave) ave))))
                           (setf h33 h44)
                           (setf h43h34 zero)))))
                    (f2cl-lib:fdo (m (f2cl-lib:int-add i (f2cl-lib:int-sub 2))
```

```
            (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
           ((> m l) nil)
(tagbody
  (setf h11
        (f2cl-lib:fref h-%data%
                       (m m)
                       ((1 ldh) (1 *))
                       h-%offset%))
  (setf h22
        (f2cl-lib:fref h-%data%
                       ((f2cl-lib:int-add m 1)
                        (f2cl-lib:int-add m 1))
                       ((1 ldh) (1 *))
                       h-%offset%))
  (setf h21
        (f2cl-lib:fref h-%data%
                       ((f2cl-lib:int-add m 1) m)
                       ((1 ldh) (1 *))
                       h-%offset%))
  (setf h12
        (f2cl-lib:fref h-%data%
                       (m (f2cl-lib:int-add m 1))
                       ((1 ldh) (1 *))
                       h-%offset%))
  (setf h44s (- h44 h11))
  (setf h33s (- h33 h11))
  (setf v1 (+ (/ (- (* h33s h44s) h43h34) h21) h12))
  (setf v2 (- h22 h11 h33s h44s))
  (setf v3
        (f2cl-lib:fref h-%data%
                       ((f2cl-lib:int-add m 2)
                        (f2cl-lib:int-add m 1))
                       ((1 ldh) (1 *))
                       h-%offset%))
  (setf s (+ (abs v1) (abs v2) (abs v3)))
  (setf v1 (/ v1 s))
  (setf v2 (/ v2 s))
  (setf v3 (/ v3 s))
  (setf (f2cl-lib:fref v (1) ((1 3))) v1)
  (setf (f2cl-lib:fref v (2) ((1 3))) v2)
  (setf (f2cl-lib:fref v (3) ((1 3))) v3)
  (if (= m l) (go label50))
  (setf h00
        (f2cl-lib:fref h-%data%
                       ((f2cl-lib:int-sub m 1)
                        (f2cl-lib:int-sub m 1))
```

```
                                    ((1 ldh) (1 *))
                                    h-%offset%))
                (setf h10
                        (f2cl-lib:fref h-%data%
                                    (m (f2cl-lib:int-sub m 1))
                                    ((1 ldh) (1 *))
                                    h-%offset%))
                (setf tst1 (* (abs v1) (+ (abs h00) (abs h11) (abs h22))))
                (if (<= (* (abs h10) (+ (abs v2) (abs v3))) (* ulp tst1))
                    (go label50))))
  label50
            (f2cl-lib:fdo (k m (f2cl-lib:int-add k 1))
                          ((> k (f2cl-lib:int-add i (f2cl-lib:int-sub 1))) nil)
              (tagbody
                (setf nr
                        (min (the fixnum 3)
                             (the fixnum
                                  (f2cl-lib:int-add (f2cl-lib:int-sub i k)
                                                    1))))
                (if (> k m)
                    (dcopy nr
                     (f2cl-lib:array-slice h
                                           double-float
                                           (k (f2cl-lib:int-sub k 1))
                                           ((1 ldh) (1 *)))
                     1 v 1))
                (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                    (dlarfg nr (f2cl-lib:fref v (1) ((1 3)))
                     (f2cl-lib:array-slice v double-float (2) ((1 3))) 1 t1)
                  (declare (ignore var-0 var-2 var-3))
                  (setf (f2cl-lib:fref v (1) ((1 3))) var-1)
                  (setf t1 var-4))
                (cond
                  ((> k m)
                   (setf (f2cl-lib:fref h-%data%
                                        (k (f2cl-lib:int-sub k 1))
                                        ((1 ldh) (1 *))
                                        h-%offset%)
                           (f2cl-lib:fref v (1) ((1 3))))
                   (setf (f2cl-lib:fref h-%data%
                                        ((f2cl-lib:int-add k 1)
                                         (f2cl-lib:int-sub k 1))
                                        ((1 ldh) (1 *))
                                        h-%offset%)
                           zero)
                   (if (< k (f2cl-lib:int-sub i 1))
```

```
            (setf (f2cl-lib:fref h-%data%
                                 ((f2cl-lib:int-add k 2)
                                  (f2cl-lib:int-sub k 1))
                                 ((1 ldh) (1 *))
                                 h-%offset%)
                  zero)))
    ((> m l)
     (setf (f2cl-lib:fref h-%data%
                          (k (f2cl-lib:int-sub k 1))
                          ((1 ldh) (1 *))
                          h-%offset%)
            (-
             (f2cl-lib:fref h-%data%
                            (k (f2cl-lib:int-sub k 1))
                            ((1 ldh) (1 *))
                            h-%offset%)))))
  (setf v2 (f2cl-lib:fref v (2) ((1 3))))
  (setf t2 (* t1 v2))
  (cond
    ((= nr 3)
     (setf v3 (f2cl-lib:fref v (3) ((1 3))))
     (setf t3 (* t1 v3))
     (f2cl-lib:fdo (j k (f2cl-lib:int-add j 1))
                   ((> j i2) nil)
       (tagbody
         (setf sum
                 (+
                  (f2cl-lib:fref h-%data%
                                 (k j)
                                 ((1 ldh) (1 *))
                                 h-%offset%)
                  (* v2
                     (f2cl-lib:fref h-%data%
                                    ((f2cl-lib:int-add k 1) j)
                                    ((1 ldh) (1 *))
                                    h-%offset%))
                  (* v3
                     (f2cl-lib:fref h-%data%
                                    ((f2cl-lib:int-add k 2) j)
                                    ((1 ldh) (1 *))
                                    h-%offset%))))
         (setf (f2cl-lib:fref h-%data%
                              (k j)
                              ((1 ldh) (1 *))
                              h-%offset%)
                 (-
```

```
                    (f2cl-lib:fref h-%data%
                                   (k j)
                                   ((1 ldh) (1 *))
                                   h-%offset%)
                    (* sum t1)))
               (setf (f2cl-lib:fref h-%data%
                                    ((f2cl-lib:int-add k 1) j)
                                    ((1 ldh) (1 *))
                                    h-%offset%)
                     (-
                      (f2cl-lib:fref h-%data%
                                     ((f2cl-lib:int-add k 1) j)
                                     ((1 ldh) (1 *))
                                     h-%offset%)
                      (* sum t2)))
               (setf (f2cl-lib:fref h-%data%
                                    ((f2cl-lib:int-add k 2) j)
                                    ((1 ldh) (1 *))
                                    h-%offset%)
                     (-
                      (f2cl-lib:fref h-%data%
                                     ((f2cl-lib:int-add k 2) j)
                                     ((1 ldh) (1 *))
                                     h-%offset%)
                      (* sum t3)))))
          (f2cl-lib:fdo (j i1 (f2cl-lib:int-add j 1))
                        ((> j
                            (min
                             (the fixnum
                                  (f2cl-lib:int-add k 3))
                             (the fixnum i)))
                         nil)
            (tagbody
              (setf sum
                    (+
                     (f2cl-lib:fref h-%data%
                                    (j k)
                                    ((1 ldh) (1 *))
                                    h-%offset%)
                     (* v2
                        (f2cl-lib:fref h-%data%
                                       (j (f2cl-lib:int-add k 1))
                                       ((1 ldh) (1 *))
                                       h-%offset%))
                     (* v3
                        (f2cl-lib:fref h-%data%
```

```
                                    (j (f2cl-lib:int-add k 2))
                                    ((1 ldh) (1 *))
                                    h-%offset%))))
          (setf (f2cl-lib:fref h-%data%
                               (j k)
                               ((1 ldh) (1 *))
                               h-%offset%)
                  (-
                   (f2cl-lib:fref h-%data%
                                  (j k)
                                  ((1 ldh) (1 *))
                                  h-%offset%)
                   (* sum t1)))
          (setf (f2cl-lib:fref h-%data%
                               (j (f2cl-lib:int-add k 1))
                               ((1 ldh) (1 *))
                               h-%offset%)
                  (-
                   (f2cl-lib:fref h-%data%
                                  (j (f2cl-lib:int-add k 1))
                                  ((1 ldh) (1 *))
                                  h-%offset%)
                   (* sum t2)))
          (setf (f2cl-lib:fref h-%data%
                               (j (f2cl-lib:int-add k 2))
                               ((1 ldh) (1 *))
                               h-%offset%)
                  (-
                   (f2cl-lib:fref h-%data%
                                  (j (f2cl-lib:int-add k 2))
                                  ((1 ldh) (1 *))
                                  h-%offset%)
                   (* sum t3)))))
      (cond
        (wantz
         (f2cl-lib:fdo (j iloz (f2cl-lib:int-add j 1))
                       ((> j ihiz) nil)
           (tagbody
             (setf sum
                     (+
                      (f2cl-lib:fref z-%data%
                                     (j k)
                                     ((1 ldz) (1 *))
                                     z-%offset%)
                      (* v2
                         (f2cl-lib:fref z-%data%
```

```
                                        (j (f2cl-lib:int-add k 1))
                                        ((1 ldz) (1 *))
                                        z-%offset%))
                         (* v3
                            (f2cl-lib:fref z-%data%
                                        (j (f2cl-lib:int-add k 2))
                                        ((1 ldz) (1 *))
                                        z-%offset%))))
                  (setf (f2cl-lib:fref z-%data%
                                       (j k)
                                       ((1 ldz) (1 *))
                                       z-%offset%)
                        (-
                         (f2cl-lib:fref z-%data%
                                       (j k)
                                       ((1 ldz) (1 *))
                                       z-%offset%)
                         (* sum t1)))
                  (setf (f2cl-lib:fref z-%data%
                                       (j (f2cl-lib:int-add k 1))
                                       ((1 ldz) (1 *))
                                       z-%offset%)
                        (-
                         (f2cl-lib:fref z-%data%
                                       (j (f2cl-lib:int-add k 1))
                                       ((1 ldz) (1 *))
                                       z-%offset%)
                         (* sum t2)))
                  (setf (f2cl-lib:fref z-%data%
                                       (j (f2cl-lib:int-add k 2))
                                       ((1 ldz) (1 *))
                                       z-%offset%)
                        (-
                         (f2cl-lib:fref z-%data%
                                       (j (f2cl-lib:int-add k 2))
                                       ((1 ldz) (1 *))
                                       z-%offset%)
                         (* sum t3))))))))
          ((= nr 2)
           (f2cl-lib:fdo (j k (f2cl-lib:int-add j 1))
                         ((> j i2) nil)
             (tagbody
               (setf sum
                     (+
                      (f2cl-lib:fref h-%data%
                                     (k j)
```

```
                                    ((1 ldh) (1 *))
                                    h-%offset%)
                             (* v2
                                (f2cl-lib:fref h-%data%
                                               ((f2cl-lib:int-add k 1) j)
                                               ((1 ldh) (1 *))
                                               h-%offset%))))
                        (setf (f2cl-lib:fref h-%data%
                                             (k j)
                                             ((1 ldh) (1 *))
                                             h-%offset%)
                              (-
                               (f2cl-lib:fref h-%data%
                                              (k j)
                                              ((1 ldh) (1 *))
                                              h-%offset%)
                               (* sum t1)))
                        (setf (f2cl-lib:fref h-%data%
                                             ((f2cl-lib:int-add k 1) j)
                                             ((1 ldh) (1 *))
                                             h-%offset%)
                              (-
                               (f2cl-lib:fref h-%data%
                                              ((f2cl-lib:int-add k 1) j)
                                              ((1 ldh) (1 *))
                                              h-%offset%)
                               (* sum t2)))))
                    (f2cl-lib:fdo (j i1 (f2cl-lib:int-add j 1))
                                  ((> j i) nil)
                      (tagbody
                        (setf sum
                              (+
                               (f2cl-lib:fref h-%data%
                                              (j k)
                                              ((1 ldh) (1 *))
                                              h-%offset%)
                               (* v2
                                  (f2cl-lib:fref h-%data%
                                                 (j (f2cl-lib:int-add k 1))
                                                 ((1 ldh) (1 *))
                                                 h-%offset%))))
                        (setf (f2cl-lib:fref h-%data%
                                             (j k)
                                             ((1 ldh) (1 *))
                                             h-%offset%)
                              (-
```

```
                    (f2cl-lib:fref h-%data%
                                   (j k)
                                   ((1 ldh) (1 *))
                                   h-%offset%)
                    (* sum t1)))
               (setf (f2cl-lib:fref h-%data%
                                    (j (f2cl-lib:int-add k 1))
                                    ((1 ldh) (1 *))
                                    h-%offset%)
                     (-
                      (f2cl-lib:fref h-%data%
                                     (j (f2cl-lib:int-add k 1))
                                     ((1 ldh) (1 *))
                                     h-%offset%)
                      (* sum t2)))))
          (cond
            (wantz
             (f2cl-lib:fdo (j iloz (f2cl-lib:int-add j 1))
                           ((> j ihiz) nil)
               (tagbody
                 (setf sum
                       (+
                        (f2cl-lib:fref z-%data%
                                       (j k)
                                       ((1 ldz) (1 *))
                                       z-%offset%)
                        (* v2
                           (f2cl-lib:fref z-%data%
                                          (j (f2cl-lib:int-add k 1))
                                          ((1 ldz) (1 *))
                                          z-%offset%))))
                 (setf (f2cl-lib:fref z-%data%
                                      (j k)
                                      ((1 ldz) (1 *))
                                      z-%offset%)
                       (-
                        (f2cl-lib:fref z-%data%
                                       (j k)
                                       ((1 ldz) (1 *))
                                       z-%offset%)
                        (* sum t1)))
                 (setf (f2cl-lib:fref z-%data%
                                      (j (f2cl-lib:int-add k 1))
                                      ((1 ldz) (1 *))
                                      z-%offset%)
                       (-
```

```
                                    (f2cl-lib:fref z-%data%
                                                   (j (f2cl-lib:int-add k 1))
                                                   ((1 ldz) (1 *))
                                                   z-%offset%)
                                    (* sum t2)))))))))))))
        (setf info i)
        (go end_label)
  label140
        (cond
          ((= l i)
           (setf (f2cl-lib:fref wr-%data% (i) ((1 *)) wr-%offset%)
                   (f2cl-lib:fref h-%data% (i i) ((1 ldh) (1 *)) h-%offset%))
           (setf (f2cl-lib:fref wi-%data% (i) ((1 *)) wi-%offset%) zero))
          ((= l (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
           (multiple-value-bind
                 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
               (dlanv2
                (f2cl-lib:fref h-%data%
                               ((f2cl-lib:int-sub i 1) (f2cl-lib:int-sub i 1))
                               ((1 ldh) (1 *))
                               h-%offset%)
                (f2cl-lib:fref h-%data%
                               ((f2cl-lib:int-sub i 1) i)
                               ((1 ldh) (1 *))
                               h-%offset%)
                (f2cl-lib:fref h-%data%
                               (i (f2cl-lib:int-sub i 1))
                               ((1 ldh) (1 *))
                               h-%offset%)
                (f2cl-lib:fref h-%data% (i i) ((1 ldh) (1 *)) h-%offset%)
                (f2cl-lib:fref wr-%data%
                               ((f2cl-lib:int-sub i 1))
                               ((1 *))
                               wr-%offset%)
                (f2cl-lib:fref wi-%data%
                               ((f2cl-lib:int-sub i 1))
                               ((1 *))
                               wi-%offset%)
                (f2cl-lib:fref wr-%data% (i) ((1 *)) wr-%offset%)
                (f2cl-lib:fref wi-%data% (i) ((1 *)) wi-%offset%) cs sn)
             (declare (ignore))
             (setf (f2cl-lib:fref h-%data%
                                  ((f2cl-lib:int-sub i 1)
                                   (f2cl-lib:int-sub i 1))
                                  ((1 ldh) (1 *))
                                  h-%offset%)
```

```
                var-0)
          (setf (f2cl-lib:fref h-%data%
                               ((f2cl-lib:int-sub i 1) i)
                               ((1 ldh) (1 *))
                               h-%offset%)
                var-1)
          (setf (f2cl-lib:fref h-%data%
                               (i (f2cl-lib:int-sub i 1))
                               ((1 ldh) (1 *))
                               h-%offset%)
                var-2)
          (setf (f2cl-lib:fref h-%data% (i i) ((1 ldh) (1 *)) h-%offset%)
                var-3)
          (setf (f2cl-lib:fref wr-%data%
                               ((f2cl-lib:int-sub i 1))
                               ((1 *))
                               wr-%offset%)
                var-4)
          (setf (f2cl-lib:fref wi-%data%
                               ((f2cl-lib:int-sub i 1))
                               ((1 *))
                               wi-%offset%)
                var-5)
          (setf (f2cl-lib:fref wr-%data% (i) ((1 *)) wr-%offset%) var-6)
          (setf (f2cl-lib:fref wi-%data% (i) ((1 *)) wi-%offset%) var-7)
          (setf cs var-8)
          (setf sn var-9))
        (cond
          (wantt
           (if (> i2 i)
               (drot (f2cl-lib:int-sub i2 i)
                (f2cl-lib:array-slice h
                                      double-float
                                      ((+ i (f2cl-lib:int-sub 1))
                                       (f2cl-lib:int-add i 1))
                                      ((1 ldh) (1 *)))
                ldh
                (f2cl-lib:array-slice h
                                      double-float
                                      (i (f2cl-lib:int-add i 1))
                                      ((1 ldh) (1 *)))
                ldh cs sn))
           (drot (f2cl-lib:int-sub i i1 1)
            (f2cl-lib:array-slice h
                                  double-float
                                  (i1 (f2cl-lib:int-sub i 1))
```

```
                                  ((1 ldh) (1 *)))
                1 (f2cl-lib:array-slice h double-float (i1 i) ((1 ldh) (1 *))) 1
                cs sn)))
            (cond
              (wantz
               (drot nz
                (f2cl-lib:array-slice z
                                      double-float
                                      (iloz (f2cl-lib:int-sub i 1))
                                      ((1 ldz) (1 *)))
                1 (f2cl-lib:array-slice z double-float (iloz i) ((1 ldz) (1 *)))
                1 cs sn)))))
        (setf itn (f2cl-lib:int-sub itn its))
        (setf i (f2cl-lib:int-sub l 1))
        (go label10)
  end_label
        (return
         (values nil nil nil nil nil nil nil nil nil nil nil nil nil info))))))
```

7.31 dlahrd LAPACK

⟨*dlahrd.input*⟩≡

```
)set break resume
)sys rm -f dlahrd.output
)spool dlahrd.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlahrd.help*⟩≡

```
======================================================================
dlahrd examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DLAHRD  -  the first NB columns of a real general n-by-(n-k+1) matrix A
       so that elements below the k-th subdiagonal are zero

SYNOPSIS
       SUBROUTINE DLAHRD( N, K, NB, A, LDA, TAU, T, LDT, Y, LDY )

           INTEGER        K, LDA, LDT, LDY, N, NB

           DOUBLE         PRECISION A( LDA, * ), T( LDT, NB ), TAU( NB  ),  Y(
                          LDY, NB )

PURPOSE
       DLAHRD  reduces  the  first  NB  columns of a real general n-by-(n-k+1)
       matrix A so that elements below the  k-th  subdiagonal  are  zero.  The
       reduction  is performed by an orthogonal similarity transformation Q' *
       A * Q. The routine returns the matrices V and T which determine Q as  a
       block reflector I - V*T*V', and also the matrix Y = A * V * T.

       This is an OBSOLETE auxiliary routine.
       This routine will be 'deprecated' in a  future release.
       Please use the new routine DLAHR2 instead.


ARGUMENTS
       N       (input) INTEGER
               The order of the matrix A.

       K       (input) INTEGER
               The offset for the reduction. Elements below the k-th subdiago-
               nal in the first NB columns are reduced to zero.

       NB      (input) INTEGER
               The number of columns to be reduced.

       A       (input/output) DOUBLE PRECISION array, dimension (LDA,N-K+1)
               On entry, the n-by-(n-k+1) general matrix A.  On exit, the ele-
```

```
            ments on and above the k-th subdiagonal in the first NB columns
            are overwritten with the corresponding elements of the  reduced
            matrix; the elements below the k-th subdiagonal, with the array
            TAU, represent the matrix Q as a product of elementary  reflec-
            tors.  The  other  columns  of  A  are  unchanged.  See Further
            Details.  LDA     (input) INTEGER The leading dimension of  the
            array A.  LDA >= max(1,N).

        TAU     (output) DOUBLE PRECISION array, dimension (NB)
                The  scalar  factors  of the elementary reflectors. See Further
                Details.

        T       (output) DOUBLE PRECISION array, dimension (LDT,NB)
                The upper triangular matrix T.

        LDT     (input) INTEGER
                The leading dimension of the array T.  LDT >= NB.

        Y       (output) DOUBLE PRECISION array, dimension (LDY,NB)
                The n-by-nb matrix Y.

        LDY     (input) INTEGER
                The leading dimension of the array Y. LDY >= N.

    FURTHER DETAILS
           The matrix Q is represented as a product of nb elementary reflectors

              Q = H(1) H(2) . . . H(nb).

           Each H(i) has the form

              H(i) = I - tau * v * v'

           where tau is a real scalar, and v is a real vector with
           v(1:i+k-1)  =  0,  v(i+k)  =  1;  v(i+k+1:n)  is  stored  on  exit  in
           A(i+k+1:n,i), and tau in TAU(i).

           The  elements of the vectors v together form the (n-k+1)-by-nb matrix V
           which is needed, with T and Y, to apply the transformation to the unre-
           duced  part  of  the  matrix,  using  an  update of the form: A := (I -
           V*T*V') * (A - Y*V').

           The contents of A on exit are illustrated by the following example with
           n = 7, k = 3 and nb = 2:

              ( a   h   a   a   a )
```

```
( a   h   a   a   a )
( a   h   a   a   a )
( h   h   a   a   a )
( v1  h   a   a   a )
( v1  v2  a   a   a )
( v1  v2  a   a   a )
```

where a denotes an element of the original matrix A, h denotes a modified element of the upper Hessenberg matrix H, and vi denotes an element of the vector defining H(i).

⟨*LAPACK dlahrd*⟩≡

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dlahrd (n k nb a lda tau t$ ldt y ldy)
    (declare (type (simple-array double-float (*)) y t$ tau a)
             (type fixnum ldy ldt lda nb k n))
    (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (t$ double-float t$-%data% t$-%offset%)
         (y double-float y-%data% y-%offset%))
      (prog ((ei 0.0) (i 0))
        (declare (type (double-float) ei) (type fixnum i))
        (if (<= n 1) (go end_label))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i nb) nil)
          (tagbody
            (cond
              ((> i 1)
               (dgemv "No transpose" n (f2cl-lib:int-sub i 1) (- one) y ldy
                (f2cl-lib:array-slice a
                                      double-float
                                      ((+ k i (f2cl-lib:int-sub 1)) 1)
                                      ((1 lda) (1 *)))
                lda one
                (f2cl-lib:array-slice a double-float (1 i) ((1 lda) (1 *))) 1)
               (dcopy (f2cl-lib:int-sub i 1)
                (f2cl-lib:array-slice a
                                      double-float
                                      ((+ k 1) i)
                                      ((1 lda) (1 *)))
                1
                (f2cl-lib:array-slice t$ double-float (1 nb) ((1 ldt) (1 nb)))
                1)
               (dtrmv "Lower" "Transpose" "Unit" (f2cl-lib:int-sub i 1)
                (f2cl-lib:array-slice a
                                      double-float
                                      ((+ k 1) 1)
                                      ((1 lda) (1 *)))
                lda
                (f2cl-lib:array-slice t$ double-float (1 nb) ((1 ldt) (1 nb)))
                1)
               (dgemv "Transpose" (f2cl-lib:int-add (f2cl-lib:int-sub n k i) 1)
                (f2cl-lib:int-sub i 1) one
                (f2cl-lib:array-slice a
```

```
                        double-float
                        ((+ k i) 1)
                        ((1 lda) (1 *)))
 lda
 (f2cl-lib:array-slice a
                        double-float
                        ((+ k i) i)
                        ((1 lda) (1 *)))
 1 one
 (f2cl-lib:array-slice t$ double-float (1 nb) ((1 ldt) (1 nb)))
 1)
(dtrmv "Upper" "Transpose" "Non-unit" (f2cl-lib:int-sub i 1) t$
 ldt
 (f2cl-lib:array-slice t$ double-float (1 nb) ((1 ldt) (1 nb)))
 1)
(dgemv "No transpose"
 (f2cl-lib:int-add (f2cl-lib:int-sub n k i) 1)
 (f2cl-lib:int-sub i 1) (- one)
 (f2cl-lib:array-slice a
                        double-float
                        ((+ k i) 1)
                        ((1 lda) (1 *)))
 lda
 (f2cl-lib:array-slice t$ double-float (1 nb) ((1 ldt) (1 nb)))
 1 one
 (f2cl-lib:array-slice a
                        double-float
                        ((+ k i) i)
                        ((1 lda) (1 *)))
 1)
(dtrmv "Lower" "No transpose" "Unit" (f2cl-lib:int-sub i 1)
 (f2cl-lib:array-slice a
                        double-float
                        ((+ k 1) 1)
                        ((1 lda) (1 *)))
 lda
 (f2cl-lib:array-slice t$ double-float (1 nb) ((1 ldt) (1 nb)))
 1)
(daxpy (f2cl-lib:int-sub i 1) (- one)
 (f2cl-lib:array-slice t$ double-float (1 nb) ((1 ldt) (1 nb)))
 1
 (f2cl-lib:array-slice a
                        double-float
                        ((+ k 1) i)
                        ((1 lda) (1 *)))
 1)
```

```
   (setf (f2cl-lib:fref a-%data%
                        ((f2cl-lib:int-sub (f2cl-lib:int-add k i)
                                           1)
                         (f2cl-lib:int-sub i 1))
                        ((1 lda) (1 *))
                        a-%offset%)
          ei)))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlarfg (f2cl-lib:int-add (f2cl-lib:int-sub n k i) 1)
     (f2cl-lib:fref a-%data%
                    ((f2cl-lib:int-add k i) i)
                    ((1 lda) (1 *))
                    a-%offset%)
     (f2cl-lib:array-slice a
                           double-float
                           ((min (f2cl-lib:int-add k i 1) n) i)
                           ((1 lda) (1 *)))
     1 (f2cl-lib:fref tau-%data% (i) ((1 nb)) tau-%offset%))
  (declare (ignore var-0 var-2 var-3))
  (setf (f2cl-lib:fref a-%data%
                       ((f2cl-lib:int-add k i) i)
                       ((1 lda) (1 *))
                       a-%offset%)
        var-1)
  (setf (f2cl-lib:fref tau-%data% (i) ((1 nb)) tau-%offset%)
        var-4))
(setf ei
       (f2cl-lib:fref a-%data%
                      ((f2cl-lib:int-add k i) i)
                      ((1 lda) (1 *))
                      a-%offset%))
(setf (f2cl-lib:fref a-%data%
                     ((f2cl-lib:int-add k i) i)
                     ((1 lda) (1 *))
                     a-%offset%)
       one)
(dgemv "No transpose" n
 (f2cl-lib:int-add (f2cl-lib:int-sub n k i) 1) one
 (f2cl-lib:array-slice a
                       double-float
                       (1 (f2cl-lib:int-add i 1))
                       ((1 lda) (1 *)))
 lda
 (f2cl-lib:array-slice a double-float ((+ k i) i) ((1 lda) (1 *)))
 1 zero
 (f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 nb))) 1)
```

```
              (dgemv "Transpose" (f2cl-lib:int-add (f2cl-lib:int-sub n k i) 1)
               (f2cl-lib:int-sub i 1) one
               (f2cl-lib:array-slice a double-float ((+ k i) 1) ((1 lda) (1 *)))
               lda
               (f2cl-lib:array-slice a double-float ((+ k i) i) ((1 lda) (1 *)))
               1 zero
               (f2cl-lib:array-slice t$ double-float (1 i) ((1 ldt) (1 nb))) 1)
              (dgemv "No transpose" n (f2cl-lib:int-sub i 1) (- one) y ldy
               (f2cl-lib:array-slice t$ double-float (1 i) ((1 ldt) (1 nb))) 1
               one (f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 nb)))
               1)
              (dscal n (f2cl-lib:fref tau-%data% (i) ((1 nb)) tau-%offset%)
               (f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 nb))) 1)
              (dscal (f2cl-lib:int-sub i 1)
               (- (f2cl-lib:fref tau-%data% (i) ((1 nb)) tau-%offset%))
               (f2cl-lib:array-slice t$ double-float (1 i) ((1 ldt) (1 nb))) 1)
              (dtrmv "Upper" "No transpose" "Non-unit" (f2cl-lib:int-sub i 1) t$
               ldt (f2cl-lib:array-slice t$ double-float (1 i) ((1 ldt) (1 nb)))
               1)
              (setf (f2cl-lib:fref t$-%data% (i i) ((1 ldt) (1 nb)) t$-%offset%)
                      (f2cl-lib:fref tau-%data% (i) ((1 nb)) tau-%offset%))))
        (setf (f2cl-lib:fref a-%data%
                             ((f2cl-lib:int-add k nb) nb)
                             ((1 lda) (1 *))
                             a-%offset%)
                ei)
  end_label
        (return (values nil nil nil nil nil nil nil nil nil nil))))))
```

7.32 dlaln2 LAPACK

⟨*dlaln2.input*⟩≡

```
)set break resume
)sys rm -f dlaln2.output
)spool dlaln2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlaln2.help*⟩≡

```
  ======================================================================
  dlaln2 examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLALN2  - a system of the form (ca A - w D ) X = s B or (ca A' - w D) X
         = s B with possible scaling ("s") and perturbation of A

  SYNOPSIS
         SUBROUTINE DLALN2( LTRANS, NA, NW, SMIN, CA, A, LDA, D1,  D2,  B,  LDB,
                            WR, WI, X, LDX, SCALE, XNORM, INFO )

             LOGICAL        LTRANS

             INTEGER        INFO, LDA, LDB, LDX, NA, NW

             DOUBLE         PRECISION CA, D1, D2, SCALE, SMIN, WI, WR, XNORM

             DOUBLE         PRECISION A( LDA, * ), B( LDB, * ), X( LDX, * )

  PURPOSE
         DLALN2 solves a system of the form  (ca A - w D ) X = s B or (ca A' - w
         D) X = s B   with possible scaling ("s") and perturbation  of  A.   (A'
         means A-transpose.)

         A  is an NA x NA real matrix, ca is a real scalar, D is an NA x NA real
         diagonal matrix, w is a real or complex value, and X and B are NA  x  1
         matrices -- real if w is real, complex if w is complex.  NA may be 1 or
         2.

         If w is complex, X and B are represented as NA x 2 matrices, the  first
         column  of  each being the real part and the second being the imaginary
         part.

         "s" is a scaling factor (.LE. 1), computed by DLALN2, which is so  cho-
         sen  that  X  can be computed without overflow.  X is further scaled if
         necessary to assure that norm(ca A - w D)*norm(X) is  less  than  over-
         flow.

         If  both singular values of (ca A - w D) are less than SMIN, SMIN*iden-
         tity will be used instead of (ca A - w D).  If only one singular  value
```

```
       is less than SMIN, one element of (ca A - w D) will be perturbed enough
       to make the smallest singular value roughly  SMIN.  If  both  singular
       values  are  at least SMIN, (ca A - w D) will not be perturbed.  In any
       case, the perturbation will be at most  some  small  multiple  of  max(
       SMIN,  ulp*norm(ca  A  -  w  D) ).  The singular values are computed by
       infinity-norm approximations, and thus will only be correct to a factor
       of 2 or so.

       Note: all input quantities are assumed to be smaller than overflow by a
       reasonable factor.  (See BIGNUM.)


  ARGUMENTS
       LTRANS  (input) LOGICAL
               =.TRUE.:  A-transpose will be used.
               =.FALSE.: A will be used (not transposed.)

       NA      (input) INTEGER
               The size of the matrix A.  It may (only) be 1 or 2.

       NW      (input) INTEGER
               1 if "w" is real, 2 if "w" is complex.  It may only be 1 or  2.

       SMIN    (input) DOUBLE PRECISION
               The  desired  lower  bound  on  the singular values of A.  This
               should be a safe distance away from underflow or overflow, say,
               between (underflow/machine precision) and  (machine precision *
               overflow ).  (See BIGNUM and ULP.)

       CA      (input) DOUBLE PRECISION
               The coefficient c, which A is multiplied by.

       A       (input) DOUBLE PRECISION array, dimension (LDA,NA)
               The NA x NA matrix A.

       LDA     (input) INTEGER
               The leading dimension of A.  It must be at least NA.

       D1      (input) DOUBLE PRECISION
               The 1,1 element in the diagonal matrix D.

       D2      (input) DOUBLE PRECISION
               The 2,2 element in the diagonal matrix D.  Not used if NW=1.

       B       (input) DOUBLE PRECISION array, dimension (LDB,NW)
               The NA x NW matrix B (right-hand side).  If NW=2 ("w"  is  com-
```

plex), column 1 contains the real part of B and column 2 contains the imaginary part.

LDB (input) INTEGER
The leading dimension of B. It must be at least NA.

WR (input) DOUBLE PRECISION
The real part of the scalar "w".

WI (input) DOUBLE PRECISION
The imaginary part of the scalar "w". Not used if NW=1.

X (output) DOUBLE PRECISION array, dimension (LDX,NW)
The NA x NW matrix X (unknowns), as computed by DLALN2. If NW=2 ("w" is complex), on exit, column 1 will contain the real part of X and column 2 will contain the imaginary part.

LDX (input) INTEGER
The leading dimension of X. It must be at least NA.

SCALE (output) DOUBLE PRECISION
The scale factor that B must be multiplied by to insure that overflow does not occur when computing X. Thus, (ca A - w D) X will be SCALE*B, not B (ignoring perturbations of A.) It will be at most 1.

XNORM (output) DOUBLE PRECISION
The infinity-norm of X, when X is regarded as an NA x NW real matrix.

INFO (output) INTEGER
An error flag. It will be set to zero if no error occurs, a negative number if an argument is in error, or a positive number if ca A - w D had to be perturbed. The possible values are:
= 0: No error occurred, and (ca A - w D) did not have to be perturbed. = 1: (ca A - w D) had to be perturbed to make its smallest (or only) singular value greater than SMIN. NOTE: In the interests of speed, this routine does not check the inputs for errors.

⟨*LAPACK dlaln2*⟩≡

```
(let* ((zero 0.0) (one 1.0) (two 2.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two))
  (let ((zswap
         (make-array 4 :element-type 't :initial-contents '(nil nil t t)))
        (rswap
         (make-array 4 :element-type 't :initial-contents '(nil t nil t)))
        (ipivot
         (make-array 16
                     :element-type 'fixnum
                     :initial-contents '(1 2 3 4 2 1 4 3 3 4 1 2 4 3 2 1))))
    (declare (type (simple-array fixnum (16)) ipivot)
             (type (simple-array (member t nil) (4)) rswap zswap))
    (defun dlaln2
           (ltrans na nw smin ca a lda d1 d2 b ldb$ wr wi x ldx scale xnorm
            info)
      (declare (type (simple-array double-float (*)) x b a)
               (type (double-float) xnorm scale wi wr d2 d1 ca smin)
               (type fixnum info ldx ldb$ lda nw na)
               (type (member t nil) ltrans))
      (f2cl-lib:with-multi-array-data
          ((a double-float a-%data% a-%offset%)
           (b double-float b-%data% b-%offset%)
           (x double-float x-%data% x-%offset%))
        (prog ((ci (make-array 4 :element-type 'double-float))
               (civ (make-array 4 :element-type 'double-float))
               (cr (make-array 4 :element-type 'double-float))
               (crv (make-array 4 :element-type 'double-float)) (bbnd 0.0)
               (bi1 0.0) (bi2 0.0) (bignum 0.0) (bnorm 0.0) (br1 0.0) (br2 0.0)
               (ci21 0.0) (ci22 0.0) (cmax 0.0) (cnorm 0.0) (cr21 0.0)
               (cr22 0.0) (csi 0.0) (csr 0.0) (li21 0.0) (lr21 0.0) (smini 0.0)
               (smlnum 0.0) (temp 0.0) (u22abs 0.0) (ui11 0.0) (ui11r 0.0)
               (ui12 0.0) (ui12s 0.0) (ui22 0.0) (ur11 0.0) (ur11r 0.0)
               (ur12 0.0) (ur12s 0.0) (ur22 0.0) (xi1 0.0) (xi2 0.0) (xr1 0.0)
               (xr2 0.0) (icmax 0) (j 0))
          (declare (type (simple-array double-float (4)) ci civ cr crv)
                   (type (double-float) bbnd bi1 bi2 bignum bnorm br1 br2 ci21
                                        ci22 cmax cnorm cr21 cr22 csi csr li21
                                        lr21 smini smlnum temp u22abs ui11
                                        ui11r ui12 ui12s ui22 ur11 ur11r ur12
                                        ur12s ur22 xi1 xi2 xr1 xr2)
                   (type fixnum icmax j))
          (setf smlnum (* two (dlamch "Safe minimum")))
          (setf bignum (/ one smlnum))
```

```
(setf smini (max smin smlnum))
(setf info 0)
(setf scale one)
(cond
  ((= na 1)
   (cond
     ((= nw 1)
      (setf csr
              (-
               (* ca
                  (f2cl-lib:fref a-%data%
                                 (1 1)
                                 ((1 lda) (1 *))
                                 a-%offset%))
               (* wr d1)))
      (setf cnorm (abs csr))
      (cond
        ((< cnorm smini)
         (setf csr smini)
         (setf cnorm smini)
         (setf info 1)))
      (setf bnorm
              (abs
               (f2cl-lib:fref b-%data%
                              (1 1)
                              ((1 ldb$) (1 *))
                              b-%offset%)))
      (cond
        ((and (< cnorm one) (> bnorm one))
         (if (> bnorm (* bignum cnorm)) (setf scale (/ one bnorm)))))
      (setf (f2cl-lib:fref x-%data% (1 1) ((1 ldx) (1 *)) x-%offset%)
              (/
               (*
                (f2cl-lib:fref b-%data%
                               (1 1)
                               ((1 ldb$) (1 *))
                               b-%offset%)
                scale)
               csr))
      (setf xnorm
              (abs
               (f2cl-lib:fref x-%data%
                              (1 1)
                              ((1 ldx) (1 *))
                              x-%offset%))))
     (t
```

```
(setf csr
        (-
         (* ca
            (f2cl-lib:fref a-%data%
                           (1 1)
                           ((1 lda) (1 *))
                           a-%offset%))
         (* wr d1)))
(setf csi (* (- wi) d1))
(setf cnorm (+ (abs csr) (abs csi)))
(cond
  ((< cnorm smini)
   (setf csr smini)
   (setf csi zero)
   (setf cnorm smini)
   (setf info 1)))
(setf bnorm
        (+
         (abs
          (f2cl-lib:fref b-%data%
                         (1 1)
                         ((1 ldb$) (1 *))
                         b-%offset%))
         (abs
          (f2cl-lib:fref b-%data%
                         (1 2)
                         ((1 ldb$) (1 *))
                         b-%offset%))))
(cond
  ((and (< cnorm one) (> bnorm one))
   (if (> bnorm (* bignum cnorm)) (setf scale (/ one bnorm)))))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
    (dladiv
     (* scale
        (f2cl-lib:fref b-%data%
                       (1 1)
                       ((1 ldb$) (1 *))
                       b-%offset%))
     (* scale
        (f2cl-lib:fref b-%data%
                       (1 2)
                       ((1 ldb$) (1 *))
                       b-%offset%))
     csr csi
     (f2cl-lib:fref x-%data% (1 1) ((1 ldx) (1 *)) x-%offset%)
     (f2cl-lib:fref x-%data% (1 2) ((1 ldx) (1 *)) x-%offset%))
```

```
          (declare (ignore var-0 var-1 var-2 var-3))
          (setf (f2cl-lib:fref x-%data%
                               (1 1)
                               ((1 ldx) (1 *))
                               x-%offset%)
                var-4)
          (setf (f2cl-lib:fref x-%data%
                               (1 2)
                               ((1 ldx) (1 *))
                               x-%offset%)
                var-5))
       (setf xnorm
             (+
              (abs
               (f2cl-lib:fref x-%data%
                              (1 1)
                              ((1 ldx) (1 *))
                              x-%offset%))
              (abs
               (f2cl-lib:fref x-%data%
                              (1 2)
                              ((1 ldx) (1 *))
                              x-%offset%)))))))
    (t
     (setf (f2cl-lib:fref crv (1) ((1 4)))
           (-
            (* ca
               (f2cl-lib:fref a-%data%
                              (1 1)
                              ((1 lda) (1 *))
                              a-%offset%))
            (* wr d1)))
     (setf (f2cl-lib:fref crv (4) ((1 4)))
           (-
            (* ca
               (f2cl-lib:fref a-%data%
                              (2 2)
                              ((1 lda) (1 *))
                              a-%offset%))
            (* wr d2)))
     (cond
       (ltrans
        (setf (f2cl-lib:fref crv (3) ((1 4)))
              (* ca
                 (f2cl-lib:fref a-%data%
                                (2 1)
```

```
                                ((1 lda) (1 *))
                                a-%offset%)))
    (setf (f2cl-lib:fref crv (2) ((1 4)))
            (* ca
               (f2cl-lib:fref a-%data%
                                (1 2)
                                ((1 lda) (1 *))
                                a-%offset%))))
   (t
    (setf (f2cl-lib:fref crv (2) ((1 4)))
            (* ca
               (f2cl-lib:fref a-%data%
                                (2 1)
                                ((1 lda) (1 *))
                                a-%offset%)))
    (setf (f2cl-lib:fref crv (3) ((1 4)))
            (* ca
               (f2cl-lib:fref a-%data%
                                (1 2)
                                ((1 lda) (1 *))
                                a-%offset%)))))
(cond
  ((= nw 1)
   (setf cmax zero)
   (setf icmax 0)
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                 ((> j 4) nil)
     (tagbody
       (cond
         ((> (abs (f2cl-lib:fref crv (j) ((1 4)))) cmax)
          (setf cmax (abs (f2cl-lib:fref crv (j) ((1 4)))))
          (setf icmax j)))))
   (cond
     ((< cmax smini)
      (setf bnorm
              (max
               (abs
                (f2cl-lib:fref b-%data%
                               (1 1)
                               ((1 ldb$) (1 *))
                               b-%offset%))
               (abs
                (f2cl-lib:fref b-%data%
                               (2 1)
                               ((1 ldb$) (1 *))
                               b-%offset%))))
```

```
      (cond
        ((and (< smini one) (> bnorm one))
         (if (> bnorm (* bignum smini))
             (setf scale (/ one bnorm)))))
      (setf temp (/ scale smini))
      (setf (f2cl-lib:fref x-%data%
                           (1 1)
                           ((1 ldx) (1 *))
                           x-%offset%)
              (* temp
                 (f2cl-lib:fref b-%data%
                                (1 1)
                                ((1 ldb$) (1 *))
                                b-%offset%)))
      (setf (f2cl-lib:fref x-%data%
                           (2 1)
                           ((1 ldx) (1 *))
                           x-%offset%)
              (* temp
                 (f2cl-lib:fref b-%data%
                                (2 1)
                                ((1 ldb$) (1 *))
                                b-%offset%)))
      (setf xnorm (* temp bnorm))
      (setf info 1)
      (go end_label)))
  (setf ur11 (f2cl-lib:fref crv (icmax) ((1 4))))
  (setf cr21
        (f2cl-lib:fref crv
                       ((f2cl-lib:fref ipivot
                                       (2 icmax)
                                       ((1 4) (1 4))))
                       ((1 4))))
  (setf ur12
        (f2cl-lib:fref crv
                       ((f2cl-lib:fref ipivot
                                       (3 icmax)
                                       ((1 4) (1 4))))
                       ((1 4))))
  (setf cr22
        (f2cl-lib:fref crv
                       ((f2cl-lib:fref ipivot
                                       (4 icmax)
                                       ((1 4) (1 4))))
                       ((1 4))))
  (setf ur11r (/ one ur11))
```

```
(setf lr21 (* ur11r cr21))
(setf ur22 (- cr22 (* ur12 lr21)))
(cond
  ((< (abs ur22) smini)
   (setf ur22 smini)
   (setf info 1)))
(cond
  ((f2cl-lib:fref rswap (icmax) ((1 4)))
   (setf br1
           (f2cl-lib:fref b-%data%
                          (2 1)
                          ((1 ldb$) (1 *))
                          b-%offset%))
   (setf br2
           (f2cl-lib:fref b-%data%
                          (1 1)
                          ((1 ldb$) (1 *))
                          b-%offset%)))
  (t
   (setf br1
           (f2cl-lib:fref b-%data%
                          (1 1)
                          ((1 ldb$) (1 *))
                          b-%offset%))
   (setf br2
           (f2cl-lib:fref b-%data%
                          (2 1)
                          ((1 ldb$) (1 *))
                          b-%offset%))))
(setf br2 (- br2 (* lr21 br1)))
(setf bbnd (max (abs (* br1 (* ur22 ur11r))) (abs br2)))
(cond
  ((and (> bbnd one) (< (abs ur22) one))
   (if (>= bbnd (* bignum (abs ur22)))
       (setf scale (/ one bbnd)))))
(setf xr2 (/ (* br2 scale) ur22))
(setf xr1 (- (* scale br1 ur11r) (* xr2 (* ur11r ur12))))
(cond
  ((f2cl-lib:fref zswap (icmax) ((1 4)))
   (setf (f2cl-lib:fref x-%data%
                        (1 1)
                        ((1 ldx) (1 *))
                        x-%offset%)
           xr2)
   (setf (f2cl-lib:fref x-%data%
                        (2 1)
```

```
                              ((1 ldx) (1 *))
                              x-%offset%)
                    xr1))
          (t
           (setf (f2cl-lib:fref x-%data%
                              (1 1)
                              ((1 ldx) (1 *))
                              x-%offset%)
                    xr1)
           (setf (f2cl-lib:fref x-%data%
                              (2 1)
                              ((1 ldx) (1 *))
                              x-%offset%)
                    xr2)))
         (setf xnorm (max (abs xr1) (abs xr2)))
         (cond
           ((and (> xnorm one) (> cmax one))
            (cond
              ((> xnorm (f2cl-lib:f2cl/ bignum cmax))
               (setf temp (/ cmax bignum))
               (setf (f2cl-lib:fref x-%data%
                                    (1 1)
                                    ((1 ldx) (1 *))
                                    x-%offset%)
                        (* temp
                           (f2cl-lib:fref x-%data%
                                          (1 1)
                                          ((1 ldx) (1 *))
                                          x-%offset%)))
               (setf (f2cl-lib:fref x-%data%
                                    (2 1)
                                    ((1 ldx) (1 *))
                                    x-%offset%)
                        (* temp
                           (f2cl-lib:fref x-%data%
                                          (2 1)
                                          ((1 ldx) (1 *))
                                          x-%offset%)))
               (setf xnorm (* temp xnorm))
               (setf scale (* temp scale)))))))
        (t
         (setf (f2cl-lib:fref civ (1) ((1 4))) (* (- wi) d1))
         (setf (f2cl-lib:fref civ (2) ((1 4))) zero)
         (setf (f2cl-lib:fref civ (3) ((1 4))) zero)
         (setf (f2cl-lib:fref civ (4) ((1 4))) (* (- wi) d2))
         (setf cmax zero)
```

```
(setf icmax 0)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j 4) nil)
  (tagbody
    (cond
      ((>
        (+ (abs (f2cl-lib:fref crv (j) ((1 4))))
           (abs (f2cl-lib:fref civ (j) ((1 4)))))
        cmax)
       (setf cmax
               (+ (abs (f2cl-lib:fref crv (j) ((1 4))))
                  (abs (f2cl-lib:fref civ (j) ((1 4))))))
       (setf icmax j)))))
(cond
  ((< cmax smini)
   (setf bnorm
           (max
            (+
             (abs
              (f2cl-lib:fref b-%data%
                             (1 1)
                             ((1 ldb$) (1 *))
                             b-%offset%))
             (abs
              (f2cl-lib:fref b-%data%
                             (1 2)
                             ((1 ldb$) (1 *))
                             b-%offset%)))
            (+
             (abs
              (f2cl-lib:fref b-%data%
                             (2 1)
                             ((1 ldb$) (1 *))
                             b-%offset%))
             (abs
              (f2cl-lib:fref b-%data%
                             (2 2)
                             ((1 ldb$) (1 *))
                             b-%offset%)))))
   (cond
     ((and (< smini one) (> bnorm one))
      (if (> bnorm (* bignum smini))
          (setf scale (/ one bnorm)))))
   (setf temp (/ scale smini))
   (setf (f2cl-lib:fref x-%data%
                        (1 1)
```

```
                              ((1 ldx) (1 *))
                              x-%offset%)
                     (* temp
                        (f2cl-lib:fref b-%data%
                                       (1 1)
                                       ((1 ldb$) (1 *))
                                       b-%offset%)))
                  (setf (f2cl-lib:fref x-%data%
                                       (2 1)
                                       ((1 ldx) (1 *))
                                       x-%offset%)
                        (* temp
                           (f2cl-lib:fref b-%data%
                                          (2 1)
                                          ((1 ldb$) (1 *))
                                          b-%offset%)))
                  (setf (f2cl-lib:fref x-%data%
                                       (1 2)
                                       ((1 ldx) (1 *))
                                       x-%offset%)
                        (* temp
                           (f2cl-lib:fref b-%data%
                                          (1 2)
                                          ((1 ldb$) (1 *))
                                          b-%offset%)))
                  (setf (f2cl-lib:fref x-%data%
                                       (2 2)
                                       ((1 ldx) (1 *))
                                       x-%offset%)
                        (* temp
                           (f2cl-lib:fref b-%data%
                                          (2 2)
                                          ((1 ldb$) (1 *))
                                          b-%offset%)))
                  (setf xnorm (* temp bnorm))
                  (setf info 1)
                  (go end_label)))
               (setf ur11 (f2cl-lib:fref crv (icmax) ((1 4))))
               (setf ui11 (f2cl-lib:fref civ (icmax) ((1 4))))
               (setf cr21
                     (f2cl-lib:fref crv
                                    ((f2cl-lib:fref ipivot
                                                    (2 icmax)
                                                    ((1 4) (1 4))))
                                    ((1 4))))
               (setf ci21
```

```
        (f2cl-lib:fref civ
                       ((f2cl-lib:fref ipivot
                                       (2 icmax)
                                       ((1 4) (1 4))))
                       ((1 4))))
(setf ur12
        (f2cl-lib:fref crv
                       ((f2cl-lib:fref ipivot
                                       (3 icmax)
                                       ((1 4) (1 4))))
                       ((1 4))))
(setf ui12
        (f2cl-lib:fref civ
                       ((f2cl-lib:fref ipivot
                                       (3 icmax)
                                       ((1 4) (1 4))))
                       ((1 4))))
(setf cr22
        (f2cl-lib:fref crv
                       ((f2cl-lib:fref ipivot
                                       (4 icmax)
                                       ((1 4) (1 4))))
                       ((1 4))))
(setf ci22
        (f2cl-lib:fref civ
                       ((f2cl-lib:fref ipivot
                                       (4 icmax)
                                       ((1 4) (1 4))))
                       ((1 4))))
(cond
  ((or (= icmax 1) (= icmax 4))
   (cond
     ((> (abs ur11) (abs ui11))
      (setf temp (/ ui11 ur11))
      (setf ur11r (/ one (* ur11 (+ one (expt temp 2)))))
      (setf ui11r (* (- temp) ur11r)))
     (t
      (setf temp (/ ur11 ui11))
      (setf ui11r (/ (- one) (* ui11 (+ one (expt temp 2)))))
      (setf ur11r (* (- temp) ui11r))))
   (setf lr21 (* cr21 ur11r))
   (setf li21 (* cr21 ui11r))
   (setf ur12s (* ur12 ur11r))
   (setf ui12s (* ur12 ui11r))
   (setf ur22 (- cr22 (* ur12 lr21)))
   (setf ui22 (- ci22 (* ur12 li21))))
```

```
   (t
    (setf ur11r (/ one ur11))
    (setf ui11r zero)
    (setf lr21 (* cr21 ur11r))
    (setf li21 (* ci21 ur11r))
    (setf ur12s (* ur12 ur11r))
    (setf ui12s (* ui12 ur11r))
    (setf ur22 (+ (- cr22 (* ur12 lr21)) (* ui12 li21)))
    (setf ui22 (- (* (- ur12) li21) (* ui12 lr21)))))
  (setf u22abs (+ (abs ur22) (abs ui22)))
  (cond
    ((< u22abs smini)
     (setf ur22 smini)
     (setf ui22 zero)
     (setf info 1)))
  (cond
    ((f2cl-lib:fref rswap (icmax) ((1 4)))
     (setf br2
             (f2cl-lib:fref b-%data%
                            (1 1)
                            ((1 ldb$) (1 *))
                            b-%offset%))
     (setf br1
             (f2cl-lib:fref b-%data%
                            (2 1)
                            ((1 ldb$) (1 *))
                            b-%offset%))
     (setf bi2
             (f2cl-lib:fref b-%data%
                            (1 2)
                            ((1 ldb$) (1 *))
                            b-%offset%))
     (setf bi1
             (f2cl-lib:fref b-%data%
                            (2 2)
                            ((1 ldb$) (1 *))
                            b-%offset%)))
    (t
     (setf br1
             (f2cl-lib:fref b-%data%
                            (1 1)
                            ((1 ldb$) (1 *))
                            b-%offset%))
     (setf br2
             (f2cl-lib:fref b-%data%
                            (2 1)
```

```
                          ((1 ldb$) (1 *))
                          b-%offset%))
   (setf bi1
           (f2cl-lib:fref b-%data%
                          (1 2)
                          ((1 ldb$) (1 *))
                          b-%offset%))
   (setf bi2
           (f2cl-lib:fref b-%data%
                          (2 2)
                          ((1 ldb$) (1 *))
                          b-%offset%))))
(setf br2 (+ (- br2 (* lr21 br1)) (* li21 bi1)))
(setf bi2 (- bi2 (* li21 br1) (* lr21 bi1)))
(setf bbnd
        (max
         (* (+ (abs br1) (abs bi1))
            (* u22abs (+ (abs ur11r) (abs ui11r))))
         (+ (abs br2) (abs bi2))))
(cond
  ((and (> bbnd one) (< u22abs one))
   (cond
     ((>= bbnd (* bignum u22abs))
      (setf scale (/ one bbnd))
      (setf br1 (* scale br1))
      (setf bi1 (* scale bi1))
      (setf br2 (* scale br2))
      (setf bi2 (* scale bi2))))))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
    (dladiv br2 bi2 ur22 ui22 xr2 xi2)
  (declare (ignore var-0 var-1 var-2 var-3))
  (setf xr2 var-4)
  (setf xi2 var-5))
(setf xr1
        (+ (- (* ur11r br1) (* ui11r bi1) (* ur12s xr2))
           (* ui12s xi2)))
(setf xi1
        (- (+ (* ui11r br1) (* ur11r bi1))
           (* ui12s xr2)
           (* ur12s xi2)))
(cond
  ((f2cl-lib:fref zswap (icmax) ((1 4)))
   (setf (f2cl-lib:fref x-%data%
                        (1 1)
                        ((1 ldx) (1 *))
                        x-%offset%)
```

```
                xr2)
          (setf (f2cl-lib:fref x-%data%
                               (2 1)
                               ((1 ldx) (1 *))
                               x-%offset%)
                xr1)
          (setf (f2cl-lib:fref x-%data%
                               (1 2)
                               ((1 ldx) (1 *))
                               x-%offset%)
                xi2)
          (setf (f2cl-lib:fref x-%data%
                               (2 2)
                               ((1 ldx) (1 *))
                               x-%offset%)
                xi1))
         (t
          (setf (f2cl-lib:fref x-%data%
                               (1 1)
                               ((1 ldx) (1 *))
                               x-%offset%)
                xr1)
          (setf (f2cl-lib:fref x-%data%
                               (2 1)
                               ((1 ldx) (1 *))
                               x-%offset%)
                xr2)
          (setf (f2cl-lib:fref x-%data%
                               (1 2)
                               ((1 ldx) (1 *))
                               x-%offset%)
                xi1)
          (setf (f2cl-lib:fref x-%data%
                               (2 2)
                               ((1 ldx) (1 *))
                               x-%offset%)
                xi2)))
        (setf xnorm
              (max (+ (abs xr1) (abs xi1)) (+ (abs xr2) (abs xi2))))
        (cond
          ((and (> xnorm one) (> cmax one))
           (cond
             ((> xnorm (f2cl-lib:f2cl/ bignum cmax))
              (setf temp (/ cmax bignum))
              (setf (f2cl-lib:fref x-%data%
                                   (1 1)
```

```
                                        ((1 ldx) (1 *))
                                        x-%offset%)
                                (* temp
                                   (f2cl-lib:fref x-%data%
                                                  (1 1)
                                                  ((1 ldx) (1 *))
                                                  x-%offset%)))
                            (setf (f2cl-lib:fref x-%data%
                                                 (2 1)
                                                 ((1 ldx) (1 *))
                                                 x-%offset%)
                                  (* temp
                                     (f2cl-lib:fref x-%data%
                                                    (2 1)
                                                    ((1 ldx) (1 *))
                                                    x-%offset%)))
                            (setf (f2cl-lib:fref x-%data%
                                                 (1 2)
                                                 ((1 ldx) (1 *))
                                                 x-%offset%)
                                  (* temp
                                     (f2cl-lib:fref x-%data%
                                                    (1 2)
                                                    ((1 ldx) (1 *))
                                                    x-%offset%)))
                            (setf (f2cl-lib:fref x-%data%
                                                 (2 2)
                                                 ((1 ldx) (1 *))
                                                 x-%offset%)
                                  (* temp
                                     (f2cl-lib:fref x-%data%
                                                    (2 2)
                                                    ((1 ldx) (1 *))
                                                    x-%offset%)))
                            (setf xnorm (* temp xnorm))
                            (setf scale (* temp scale)))))))))
  end_label
           (return
            (values nil
                    nil
                    nil
                    nil
                    nil
                    nil
                    nil
                    nil
```

```
                    nil
                    nil
                    nil
                    nil
                    nil
                    nil
                    nil
                    scale
                    xnorm
                    info)))))))
```

7.33 dlamch LAPACK

⟨*dlamch.input*⟩≡

```
  )set break resume
  )sys rm -f dlamch.output
  )spool dlamch.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dlamch.help*⟩≡

```
==========================================================================
dlamch examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

NAME
       DLAMCH - determine double precision machine parameters

SYNOPSIS
       DOUBLE PRECISION FUNCTION DLAMCH( CMACH )

           CHARACTER    CMACH

PURPOSE
       DLAMCH determines double precision machine parameters.

ARGUMENTS
       CMACH   (input) CHARACTER*1
               Specifies the value to be returned by DLAMCH:
               = 'E' or 'e',   DLAMCH := eps
               = 'S' or 's ,   DLAMCH := sfmin
               = 'B' or 'b',   DLAMCH := base
               = 'P' or 'p',   DLAMCH := eps*base
               = 'N' or 'n',   DLAMCH := t
               = 'R' or 'r',   DLAMCH := rnd
               = 'M' or 'm',   DLAMCH := emin
               = 'U' or 'u',   DLAMCH := rmin
               = 'L' or 'l',   DLAMCH := emax
               = 'O' or 'o',   DLAMCH := rmax

               where

       eps   = relative machine precision
             sfmin = safe minimum, such that 1/sfmin does not overflow base  =
             base of the machine prec  = eps*base t      =  number  of  (base)
             digits  in the mantissa rnd   = 1.0 when rounding occurs in addi-
             tion, 0.0 otherwise emin  =  minimum  exponent  before  (gradual)
             underflow  rmin   =  underflow threshold - base**(emin-1) emax  =
             largest exponent before overflow rmax  =  overflow  threshold   -
             (base**emax)*(1-eps)
```

⟨*LAPACK dlamch*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (let ((eps 0.0)
        (sfmin 0.0)
        (base 0.0)
        (t$ 0.0f0)
        (rnd 0.0)
        (emin 0.0)
        (rmin 0.0)
        (emax 0.0)
        (rmax 0.0)
        (prec 0.0)
        (first$ nil))
    (declare (type (member t nil) first$)
             (type (single-float) t$)
             (type (double-float) prec rmax emax rmin emin rnd base sfmin eps))
    (setq first$ t)
    (defun dlamch (cmach)
      (declare (type character cmach))
      (f2cl-lib:with-multi-array-data
          ((cmach character cmach-%data% cmach-%offset%))
        (prog ((rmach 0.0) (small 0.0) (t$ 0.0) (beta 0) (imax 0) (imin 0)
               (it 0) (lrnd nil) (dlamch 0.0))
          (declare (type fixnum beta imax imin it)
                   (type (member t nil) lrnd)
                   (type (double-float) rmach small t$ dlamch))
          (cond
            (first$
             (setf first$ nil)
             (multiple-value-bind
                   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
                 (dlamc2 beta it lrnd eps imin rmin imax rmax)
               (declare (ignore))
               (setf beta var-0)
               (setf it var-1)
               (setf lrnd var-2)
               (setf eps var-3)
               (setf imin var-4)
               (setf rmin var-5)
               (setf imax var-6)
               (setf rmax var-7))
             (setf base (coerce (the fixnum beta) 'double-float))
             (setf t$ (coerce (the fixnum it) 'double-float))
             (cond
```

```
                (lrnd
                 (setf rnd one)
                 (setf eps (/ (expt base (f2cl-lib:int-sub 1 it)) 2)))
                (t
                 (setf rnd zero)
                 (setf eps (expt base (f2cl-lib:int-sub 1 it)))))
               (setf prec (* eps base))
               (setf emin (coerce (the fixnum imin) 'double-float))
               (setf emax (coerce (the fixnum imax) 'double-float))
               (setf sfmin rmin)
               (setf small (/ one rmax))
               (cond
                 ((>= small sfmin)
                  (setf sfmin (* small (+ one eps)))))))
          (cond
            ((char-equal cmach #\E)
             (setf rmach eps))
            ((char-equal cmach #\S)
             (setf rmach sfmin))
            ((char-equal cmach #\B)
             (setf rmach base))
            ((char-equal cmach #\P)
             (setf rmach prec))
            ((char-equal cmach #\N)
             (setf rmach t$))
            ((char-equal cmach #\R)
             (setf rmach rnd))
            ((char-equal cmach #\M)
             (setf rmach emin))
            ((char-equal cmach #\U)
             (setf rmach rmin))
            ((char-equal cmach #\L)
             (setf rmach emax))
            ((char-equal cmach #\O)
             (setf rmach rmax)))
          (setf dlamch rmach)
end_label
          (return (values dlamch nil)))))))
```

7.34 dlamc1 LAPACK

⟨*dlamc1.input*⟩≡

```
)set break resume
)sys rm -f dlamc1.output
)spool dlamc1.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlamc1.help*⟩≡

```
  =====================================================================
  dlamc1 examples
  =====================================================================

  =====================================================================
  Man Page Details
  =====================================================================

    -- LAPACK auxiliary routine (version 1.1) --
       Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
       Courant Institute, Argonne National Lab, and Rice University
       October 31, 1992

       .. Scalar Arguments ..
  <       LOGICAL            IEEE1, RND >
  <       INTEGER            BETA, T >
       ..

    Purpose
    =======

    DLAMC1 determines the machine parameters given by BETA, T, RND, and
    IEEE1.

    Arguments
    =========

    BETA    (output) INTEGER
            The base of the machine.

    T       (output) INTEGER
            The number of ( BETA ) digits in the mantissa.

    RND     (output) LOGICAL
            Specifies whether proper rounding  ( RND = .TRUE. )  or
            chopping  ( RND = .FALSE. )  occurs in addition. This may not
            be a reliable guide to the way in which the machine performs
            its arithmetic.

    IEEE1   (output) LOGICAL
            Specifies whether rounding appears to be done in the IEEE
            'round to nearest' style.

    Further Details
    ===============
```

```
See Malcolm M. A. (1972) Algorithms to reveal properties of
    floating-point arithmetic. Comms. of the ACM, 15, 949-951.

See Gentleman W. M. and Marovich S. B. (1974) More on algorithms
    that reveal properties of floating point arithmetic units.
    Comms. of the ACM, 17, 276-277.
```

⟨*LAPACK dlamc1*⟩≡

```
(let ((lieee1 nil) (lbeta 0) (lrnd nil) (f2cl-lib:lt 0) (first$ nil))
  (declare (type fixnum f2cl-lib:lt lbeta)
           (type (member t nil) first$ lrnd lieee1))
  (setq first$ t)
  (defun dlamc1 (beta t$ rnd ieee1)
    (declare (type (member t nil) ieee1 rnd)
             (type fixnum t$ beta))
    (prog ((a 0.0) (b 0.0) (c 0.0) (f 0.0) (one 0.0) (qtr 0.0) (savec 0.0)
           (t1 0.0) (t2 0.0))
      (declare (type (double-float) t2 t1 savec qtr one f c b a))
      (cond
        (first$
         (tagbody
           (setf first$ nil)
           (setf one (coerce (the fixnum 1) 'double-float))
           (setf a (coerce (the fixnum 1) 'double-float))
           (setf c (coerce (the fixnum 1) 'double-float))
 label10
           (cond
             ((= c one)
              (setf a (* 2 a))
              (setf c
                      (multiple-value-bind (ret-val var-0 var-1)
                          (dlamc3 a one)
                        (declare (ignore))
                        (setf a var-0)
                        (setf one var-1)
                        ret-val))
              (setf c
                      (multiple-value-bind (ret-val var-0 var-1)
                          (dlamc3 c (- a))
                        (declare (ignore var-1))
                        (setf c var-0)
                        ret-val))
              (go label10)))
           (setf b (coerce (the fixnum 1) 'double-float))
           (setf c
                   (multiple-value-bind (ret-val var-0 var-1)
                       (dlamc3 a b)
                     (declare (ignore))
                     (setf a var-0)
                     (setf b var-1)
                     ret-val))
 label20
           (cond
```

```
  ((= c a)
   (setf b (* 2 b))
   (setf c
           (multiple-value-bind (ret-val var-0 var-1)
               (dlamc3 a b)
             (declare (ignore))
             (setf a var-0)
             (setf b var-1)
             ret-val))
   (go label20)))
(setf qtr (/ one 4))
(setf savec c)
(setf c
        (multiple-value-bind (ret-val var-0 var-1)
            (dlamc3 c (- a))
          (declare (ignore var-1))
          (setf c var-0)
          ret-val))
(setf lbeta (f2cl-lib:int (+ c qtr)))
(setf b (coerce (the fixnum lbeta) 'double-float))
(setf f (dlamc3 (/ b 2) (/ (- b) 100)))
(setf c
        (multiple-value-bind (ret-val var-0 var-1)
            (dlamc3 f a)
          (declare (ignore))
          (setf f var-0)
          (setf a var-1)
          ret-val))
(cond
  ((= c a)
   (setf lrnd t))
  (t
   (setf lrnd nil)))
(setf f (dlamc3 (/ b 2) (/ b 100)))
(setf c
        (multiple-value-bind (ret-val var-0 var-1)
            (dlamc3 f a)
          (declare (ignore))
          (setf f var-0)
          (setf a var-1)
          ret-val))
(if (and lrnd (= c a)) (setf lrnd nil))
(setf t1
        (multiple-value-bind (ret-val var-0 var-1)
            (dlamc3 (/ b 2) a)
          (declare (ignore var-0))
```

```
                    (setf a var-1)
                    ret-val))
          (setf t2
                  (multiple-value-bind (ret-val var-0 var-1)
                      (dlamc3 (/ b 2) savec)
                    (declare (ignore var-0))
                    (setf savec var-1)
                    ret-val))
          (setf lieee1 (and (= t1 a) (> t2 savec) lrnd))
          (setf f2cl-lib:lt 0)
          (setf a (coerce (the fixnum 1) 'double-float))
          (setf c (coerce (the fixnum 1) 'double-float))
label30
          (cond
            ((= c one)
             (setf f2cl-lib:lt (f2cl-lib:int-add f2cl-lib:lt 1))
             (setf a (* a lbeta))
             (setf c
                     (multiple-value-bind (ret-val var-0 var-1)
                         (dlamc3 a one)
                       (declare (ignore))
                       (setf a var-0)
                       (setf one var-1)
                       ret-val))
             (setf c
                     (multiple-value-bind (ret-val var-0 var-1)
                         (dlamc3 c (- a))
                       (declare (ignore var-1))
                       (setf c var-0)
                       ret-val))
             (go label30))))))
     (setf beta lbeta)
     (setf t$ f2cl-lib:lt)
     (setf rnd lrnd)
     (setf ieee1 lieee1)
end_label
     (return (values beta t$ rnd ieee1)))))
```

7.35 dlamc2 LAPACK

⟨*dlamc2.input*⟩≡

```
)set break resume
)sys rm -f dlamc2.output
)spool dlamc2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlamc2.help*⟩≡

```
  =====================================================================
  dlamc2 examples
  =====================================================================

  =====================================================================
  Man Page Details
  =====================================================================

    -- LAPACK auxiliary routine (version 1.1) --
       Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
       Courant Institute, Argonne National Lab, and Rice University
       October 31, 1992

       .. Scalar Arguments ..
<      LOGICAL            RND >
<      INTEGER            BETA, EMAX, EMIN, T >
<      DOUBLE PRECISION   EPS, RMAX, RMIN >
       ..

    Purpose
    =======

    DLAMC2 determines the machine parameters specified in its argument
    list.

    Arguments
    =========

    BETA    (output) INTEGER
            The base of the machine.

    T       (output) INTEGER
            The number of ( BETA ) digits in the mantissa.

    RND     (output) LOGICAL
            Specifies whether proper rounding  ( RND = .TRUE. )  or
            chopping  ( RND = .FALSE. )  occurs in addition. This may not
            be a reliable guide to the way in which the machine performs
            its arithmetic.

    EPS     (output) DOUBLE PRECISION
            The smallest positive number such that

               fl( 1.0 - EPS ) .LT. 1.0,
```

```
          where fl denotes the computed value.

     EMIN    (output) INTEGER
             The minimum exponent before (gradual) underflow occurs.

     RMIN    (output) DOUBLE PRECISION
             The smallest normalized number for the machine, given by
             BASE**( EMIN - 1 ), where  BASE  is the floating point value
             of BETA.

     EMAX    (output) INTEGER
             The maximum exponent before overflow occurs.

     RMAX    (output) DOUBLE PRECISION
             The largest positive number for the machine, given by
             BASE**EMAX * ( 1 - EPS ), where  BASE  is the floating point
             value of BETA.

     Further Details
     ===============

     The computation of  EPS  is based on a routine PARANOIA by
     W. Kahan of the University of California at Berkeley.
```

⟨*LAPACK dlamc2*⟩≡

```
(let ((lbeta 0)
      (lemax 0)
      (lemin 0)
      (leps 0.0)
      (lrmax 0.0)
      (lrmin 0.0)
      (f2cl-lib:lt 0)
      (first$ nil)
      (iwarn nil))
  (declare (type (member t nil) iwarn first$)
           (type (double-float) lrmin lrmax leps)
           (type fixnum f2cl-lib:lt lemin lemax lbeta))
  (setq first$ t)
  (setq iwarn nil)
  (defun dlamc2 (beta t$ rnd eps emin rmin emax rmax)
    (declare (type (double-float) rmax rmin eps)
             (type (member t nil) rnd)
             (type fixnum emax emin t$ beta))
    (prog ((a 0.0) (b 0.0) (c 0.0) (half 0.0) (one 0.0) (rbase 0.0)
           (sixth$ 0.0) (small 0.0) (third$ 0.0) (two 0.0) (zero 0.0) (gnmin 0)
           (gpmin 0) (i 0) (ngnmin 0) (ngpmin 0) (ieee nil) (lieee1 nil)
           (lrnd nil))
      (declare (type (member t nil) lrnd lieee1 ieee)
               (type fixnum ngpmin ngnmin i gpmin gnmin)
               (type (double-float) zero two third$ small sixth$ rbase one half
                                    c b a))
      (cond
        (first$
         (tagbody
           (setf first$ nil)
           (setf zero (coerce (the fixnum 0) 'double-float))
           (setf one (coerce (the fixnum 1) 'double-float))
           (setf two (coerce (the fixnum 2) 'double-float))
           (multiple-value-bind (var-0 var-1 var-2 var-3)
               (dlamc1 lbeta f2cl-lib:lt lrnd lieee1)
             (declare (ignore))
             (setf lbeta var-0)
             (setf f2cl-lib:lt var-1)
             (setf lrnd var-2)
             (setf lieee1 var-3))
           (setf b (coerce (the fixnum lbeta) 'double-float))
           (setf a (expt b (f2cl-lib:int-sub f2cl-lib:lt)))
           (setf leps a)
           (setf b (/ two 3))
           (setf half (/ one 2))
```

```
            (setf sixth$
                    (multiple-value-bind (ret-val var-0 var-1)
                        (dlamc3 b (- half))
                      (declare (ignore var-1))
                      (setf b var-0)
                      ret-val))
            (setf third$
                    (multiple-value-bind (ret-val var-0 var-1)
                        (dlamc3 sixth$ sixth$)
                      (declare (ignore))
                      (setf sixth$ var-0)
                      (setf sixth$ var-1)
                      ret-val))
            (setf b
                    (multiple-value-bind (ret-val var-0 var-1)
                        (dlamc3 third$ (- half))
                      (declare (ignore var-1))
                      (setf third$ var-0)
                      ret-val))
            (setf b
                    (multiple-value-bind (ret-val var-0 var-1)
                        (dlamc3 b sixth$)
                      (declare (ignore))
                      (setf b var-0)
                      (setf sixth$ var-1)
                      ret-val))
            (setf b (abs b))
            (if (< b leps) (setf b leps))
            (setf leps (coerce (the fixnum 1) 'double-float))
  label10
            (cond
              ((and (> leps b) (> b zero))
               (setf leps b)
               (setf c (dlamc3 (* half leps) (* (expt two 5) (expt leps 2))))
               (setf c
                       (multiple-value-bind (ret-val var-0 var-1)
                           (dlamc3 half (- c))
                         (declare (ignore var-1))
                         (setf half var-0)
                         ret-val))
               (setf b
                       (multiple-value-bind (ret-val var-0 var-1)
                           (dlamc3 half c)
                         (declare (ignore))
                         (setf half var-0)
                         (setf c var-1)
```

```
              ret-val))
   (setf c
           (multiple-value-bind (ret-val var-0 var-1)
               (dlamc3 half (- b))
             (declare (ignore var-1))
             (setf half var-0)
             ret-val))
   (setf b
           (multiple-value-bind (ret-val var-0 var-1)
               (dlamc3 half c)
             (declare (ignore))
             (setf half var-0)
             (setf c var-1)
             ret-val))
   (go label10)))
(if (< a leps) (setf leps a))
(setf rbase (/ one lbeta))
(setf small one)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i 3) nil)
  (tagbody
    (setf small
            (multiple-value-bind (ret-val var-0 var-1)
                (dlamc3 (* small rbase) zero)
              (declare (ignore var-0))
              (setf zero var-1)
              ret-val))))
(setf a
        (multiple-value-bind (ret-val var-0 var-1)
            (dlamc3 one small)
          (declare (ignore))
          (setf one var-0)
          (setf small var-1)
          ret-val))
(multiple-value-bind (var-0 var-1 var-2)
    (dlamc4 ngpmin one lbeta)
  (declare (ignore var-1 var-2))
  (setf ngpmin var-0))
(multiple-value-bind (var-0 var-1 var-2)
    (dlamc4 ngnmin (- one) lbeta)
  (declare (ignore var-1 var-2))
  (setf ngnmin var-0))
(multiple-value-bind (var-0 var-1 var-2)
    (dlamc4 gpmin a lbeta)
  (declare (ignore var-1 var-2))
  (setf gpmin var-0))
```

```
(multiple-value-bind (var-0 var-1 var-2)
    (dlamc4 gnmin (- a) lbeta)
  (declare (ignore var-1 var-2))
  (setf gnmin var-0))
(setf ieee nil)
(cond
  ((and (= ngpmin ngnmin) (= gpmin gnmin))
   (cond
     ((= ngpmin gpmin)
      (setf lemin ngpmin))
     ((= (f2cl-lib:int-add gpmin (f2cl-lib:int-sub ngpmin)) 3)
      (setf lemin
              (f2cl-lib:int-add (f2cl-lib:int-sub ngpmin 1)
                                f2cl-lib:lt))
      (setf ieee t))
     (t
      (setf lemin
              (min (the fixnum ngpmin)
                   (the fixnum gpmin)))
      (setf iwarn t))))
  ((and (= ngpmin gpmin) (= ngnmin gnmin))
   (cond
     ((= (abs (f2cl-lib:int-add ngpmin (f2cl-lib:int-sub ngnmin))) 1)
      (setf lemin
              (max (the fixnum ngpmin)
                   (the fixnum ngnmin))))
     (t
      (setf lemin
              (min (the fixnum ngpmin)
                   (the fixnum ngnmin)))
      (setf iwarn t))))
  ((and
    (= (abs (f2cl-lib:int-add ngpmin (f2cl-lib:int-sub ngnmin))) 1)
    (= gpmin gnmin))
   (cond
     ((=
       (f2cl-lib:int-add gpmin
                         (f2cl-lib:int-sub
                          (min (the fixnum ngpmin)
                               (the fixnum ngnmin))))
       3)
      (setf lemin
              (f2cl-lib:int-add
               (f2cl-lib:int-sub
                (max (the fixnum ngpmin)
                     (the fixnum ngnmin))
```

```
                        1)
                       f2cl-lib:lt)))
              (t
               (setf lemin
                       (min (the fixnum ngpmin)
                            (the fixnum ngnmin)))
               (setf iwarn t))))
            (t
             (setf lemin
                     (min (the fixnum ngpmin)
                          (the fixnum ngnmin)
                          (the fixnum gpmin)
                          (the fixnum gnmin)))
             (setf iwarn t)))
           (cond
            (iwarn
             (setf first$ t)
             (format t "~&~s~a~s~%~s~%~s~s~%"
               "WARNING. The value EMIN may be incorrect:- EMIN = "
               lemin
               " If, after inspection, the value EMIN looks acceptable "
               "please comment out "
               " the IF block as marked within the code of routine DLAMC2,"
               " otherwise supply EMIN explicitly.")))
           (setf ieee (or ieee lieee1))
           (setf lrmin (coerce (the fixnum 1) 'double-float))
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                         ((> i (f2cl-lib:int-add 1 (f2cl-lib:int-sub lemin)))
                          nil)
            (tagbody
              (setf lrmin
                      (multiple-value-bind (ret-val var-0 var-1)
                          (dlamc3 (* lrmin rbase) zero)
                        (declare (ignore var-0))
                        (setf zero var-1)
                        ret-val))))
           (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
               (dlamc5 lbeta f2cl-lib:lt lemin ieee lemax lrmax)
             (declare (ignore var-1))
             (setf lbeta var-0)
             (setf lemin var-2)
             (setf ieee var-3)
             (setf lemax var-4)
             (setf lrmax var-5)))))
      (setf beta lbeta)
      (setf t$ f2cl-lib:lt)
```

```
        (setf rnd lrnd)
        (setf eps leps)
        (setf emin lemin)
        (setf rmin lrmin)
        (setf emax lemax)
        (setf rmax lrmax)
   end_label
        (return (values beta t$ rnd eps emin rmin emax rmax)))))
```

7.36 dlamc3 LAPACK

⟨*dlamc3.input*⟩≡

```
  )set break resume
  )sys rm -f dlamc3.output
  )spool dlamc3.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dlamc3.help*⟩≡
```
  ======================================================================
  dlamc3 examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

    -- LAPACK auxiliary routine (version 1.1) --
       Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
       Courant Institute, Argonne National Lab, and Rice University
       October 31, 1992

       .. Scalar Arguments ..
  <       DOUBLE PRECISION   A, B >
       ..

    Purpose
    =======

    DLAMC3  is intended to force  A  and  B  to be stored prior to doing
    the addition of  A  and  B ,  for use in situations where optimizers
    might hold one of these in a register.

    Arguments
    =========

    A, B    (input) DOUBLE PRECISION
            The values A and B.
```

⟨*LAPACK dlamc3*⟩≡
```
  (defun dlamc3 (a b)
    (declare (type (double-float) b a))
    (prog ((dlamc3 0.0))
      (declare (type (double-float) dlamc3))
      (setf dlamc3 (+ a b))
      (return (values dlamc3 a b))))
```

7.37 dlamc4 LAPACK

⟨*dlamc4.input*⟩≡

```
)set break resume
)sys rm -f dlamc4.output
)spool dlamc4.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlamc4.help*⟩≡

```
  =====================================================================
  dlamc4 examples
  =====================================================================

  =====================================================================
  Man Page Details
  =====================================================================

    -- LAPACK auxiliary routine (version 1.1) --
       Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
       Courant Institute, Argonne National Lab, and Rice University
       October 31, 1992

       .. Scalar Arguments ..
  <       INTEGER            BASE, EMIN >
  <       DOUBLE PRECISION   START >
       ..

    Purpose
    =======

    DLAMC4 is a service routine for DLAMC2.

    Arguments
    =========

    EMIN    (output) EMIN
            The minimum exponent before (gradual) underflow, computed by
            setting A = START and dividing by BASE until the previous A
            can not be recovered.

    START   (input) DOUBLE PRECISION
            The starting point for determining EMIN.

    BASE    (input) INTEGER
            The base of the machine.
```

⟨*LAPACK dlamc4*⟩≡

```
(defun dlamc4 (emin start base)
  (declare (type (double-float) start) (type fixnum base emin))
  (prog ((a 0.0) (b1 0.0) (b2 0.0) (c1 0.0) (c2 0.0) (d1 0.0) (d2 0.0)
         (one 0.0) (rbase 0.0) (zero 0.0) (i 0))
    (declare (type fixnum i)
             (type (double-float) zero rbase one d2 d1 c2 c1 b2 b1 a))
    (setf a start)
    (setf one (coerce (the fixnum 1) 'double-float))
    (setf rbase (/ one base))
    (setf zero (coerce (the fixnum 0) 'double-float))
    (setf emin 1)
    (setf b1
          (multiple-value-bind (ret-val var-0 var-1)
              (dlamc3 (* a rbase) zero)
            (declare (ignore var-0))
            (setf zero var-1)
            ret-val))
    (setf c1 a)
    (setf c2 a)
    (setf d1 a)
    (setf d2 a)
 label10
    (cond
      ((and (= c1 a) (= c2 a) (= d1 a) (= d2 a))
       (setf emin (f2cl-lib:int-sub emin 1))
       (setf a b1)
       (setf b1
             (multiple-value-bind (ret-val var-0 var-1)
                 (dlamc3 (/ a base) zero)
               (declare (ignore var-0))
               (setf zero var-1)
               ret-val))
       (setf c1
             (multiple-value-bind (ret-val var-0 var-1)
                 (dlamc3 (* b1 base) zero)
               (declare (ignore var-0))
               (setf zero var-1)
               ret-val))
       (setf d1 zero)
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i base) nil)
         (tagbody (setf d1 (+ d1 b1)) label20))
       (setf b2
             (multiple-value-bind (ret-val var-0 var-1)
                 (dlamc3 (* a rbase) zero)
```

```
                (declare (ignore var-0))
                (setf zero var-1)
                ret-val))
        (setf c2
              (multiple-value-bind (ret-val var-0 var-1)
                  (dlamc3 (/ b2 rbase) zero)
                (declare (ignore var-0))
                (setf zero var-1)
                ret-val))
        (setf d2 zero)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i base) nil)
          (tagbody (setf d2 (+ d2 b2)) label30))
        (go label10)))
   end_label
      (return (values emin nil nil))))
```

7.38 dlamc5 LAPACK

⟨*dlamc5.input*⟩≡

```
)set break resume
)sys rm -f dlamc5.output
)spool dlamc5.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlamc5.help*⟩≡

```
  =====================================================================
  dlamc5 examples
  =====================================================================

  =====================================================================
  Man Page Details
  =====================================================================

    -- LAPACK auxiliary routine (version 1.1) --
       Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
       Courant Institute, Argonne National Lab, and Rice University
       October 31, 1992

       .. Scalar Arguments ..
<      LOGICAL            IEEE >
<      INTEGER            BETA, EMAX, EMIN, P >
<      DOUBLE PRECISION   RMAX >
       ..

    Purpose
    =======

    DLAMC5 attempts to compute RMAX, the largest machine floating-point
    number, without overflow.  It assumes that EMAX + abs(EMIN) sum
    approximately to a power of 2.  It will fail on machines where this
    assumption does not hold, for example, the Cyber 205 (EMIN = -28625,
    EMAX = 28718).  It will also fail if the value supplied for EMIN is
    too large (i.e. too close to zero), probably with overflow.

    Arguments
    =========

    BETA    (input) INTEGER
            The base of floating-point arithmetic.

    P       (input) INTEGER
            The number of base BETA digits in the mantissa of a
            floating-point value.

    EMIN    (input) INTEGER
            The minimum exponent before (gradual) underflow.

    IEEE    (input) LOGICAL
            A logical flag specifying whether or not the arithmetic
            system is thought to comply with the IEEE standard.
```

```
    EMAX    (output) INTEGER
            The largest exponent before overflow

    RMAX    (output) DOUBLE PRECISION
            The largest machine floating-point number.
```

⟨*LAPACK dlamc5*⟩≡

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dlamc5 (beta p emin ieee emax rmax)
    (declare (type (double-float) rmax)
             (type (member t nil) ieee)
             (type fixnum emax emin p beta))
    (prog ((oldy 0.0) (recbas 0.0) (y 0.0) (z 0.0) (exbits 0) (expsum 0) (i 0)
           (lexp 0) (nbits 0) (try 0) (uexp 0))
      (declare (type (double-float) oldy recbas y z)
               (type fixnum exbits expsum i lexp nbits try uexp))
      (setf lexp 1)
      (setf exbits 1)
 label10
      (setf try (f2cl-lib:int-mul lexp 2))
      (cond
        ((<= try (f2cl-lib:int-sub emin))
         (setf lexp try)
         (setf exbits (f2cl-lib:int-add exbits 1))
         (go label10)))
      (cond
        ((= lexp (f2cl-lib:int-sub emin))
         (setf uexp lexp))
        (t
         (setf uexp try)
         (setf exbits (f2cl-lib:int-add exbits 1))))
      (cond
        ((> (f2cl-lib:int-add uexp emin)
            (f2cl-lib:int-add (f2cl-lib:int-sub lexp) (f2cl-lib:int-sub emin)))
         (setf expsum (f2cl-lib:int-mul 2 lexp)))
        (t
         (setf expsum (f2cl-lib:int-mul 2 uexp))))
      (setf emax (f2cl-lib:int-sub (f2cl-lib:int-add expsum emin) 1))
      (setf nbits (f2cl-lib:int-add 1 exbits p))
      (cond
        ((and (= (mod nbits 2) 1) (= beta 2))
         (setf emax (f2cl-lib:int-sub emax 1))))
      (cond
        (ieee
         (setf emax (f2cl-lib:int-sub emax 1))))
      (setf recbas (/ one beta))
      (setf z (- beta one))
      (setf y zero)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i p) nil)
```

```
          (tagbody
            (setf z (* z recbas))
            (if (< y one) (setf oldy y))
            (setf y
                    (multiple-value-bind (ret-val var-0 var-1)
                        (dlamc3 y z)
                      (declare (ignore))
                      (setf y var-0)
                      (setf z var-1)
                      ret-val))))
       (if (>= y one) (setf y oldy))
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i emax) nil)
         (tagbody
           (setf y
                   (multiple-value-bind (ret-val var-0 var-1)
                       (dlamc3 (* y beta) zero)
                     (declare (ignore var-0))
                     (setf zero var-1)
                     ret-val))))
       (setf rmax y)
   end_label
       (return (values beta nil emin ieee emax rmax)))))
```

7.39 dlamrg LAPACK

⟨*dlamrg.input*⟩≡

```
)set break resume
)sys rm -f dlamrg.output
)spool dlamrg.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlamrg.help*⟩≡

```
  ======================================================================
  dlamrg examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLAMRG  -  create a permutation list which will merge the elements of A
         (which is composed of two independently sorted sets) into a single  set
         which is sorted in ascending order

  SYNOPSIS
         SUBROUTINE DLAMRG( N1, N2, A, DTRD1, DTRD2, INDEX )

             INTEGER        DTRD1, DTRD2, N1, N2

             INTEGER        INDEX( * )

             DOUBLE         PRECISION A( * )

  PURPOSE
         DLAMRG  will create a permutation list which will merge the elements of
         A (which is composed of two independently sorted sets)  into  a  single
         set which is sorted in ascending order.


  ARGUMENTS
         N1     (input) INTEGER
                N2      (input)  INTEGER These arguements contain the respective
                lengths of the two sorted lists to be merged.

         A      (input) DOUBLE PRECISION array, dimension (N1+N2)
                The first N1 elements of A contain a list of numbers  which  are
                sorted  in  either  ascending or descending order.  Likewise for
                the final N2 elements.

         DTRD1  (input) INTEGER
                DTRD2  (input) INTEGER These are the strides to be taken through
                the  array  A.   Allowable  strides are 1 and -1.  They indicate
                whether a subset of A is sorted in  ascending  (DTRDx  =  1)  or
                descending (DTRDx = -1) order.

         INDEX  (output) INTEGER array, dimension (N1+N2)
```

```
On  exit this array will contain a permutation such that if B( I
) = A( INDEX( I ) ) for I=1,N1+N2, then  B  will  be  sorted  in
ascending order.
```

⟨*LAPACK dlamrg*⟩≡

```
(defun dlamrg (n1 n2 a dtrd1 dtrd2 indx)
  (declare (type (simple-array fixnum (*)) indx)
           (type (simple-array double-float (*)) a)
           (type fixnum dtrd2 dtrd1 n2 n1))
  (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (indx fixnum indx-%data% indx-%offset%))
    (prog ((i 0) (ind1 0) (ind2 0) (n1sv 0) (n2sv 0))
      (declare (type fixnum n2sv n1sv ind2 ind1 i))
      (setf n1sv n1)
      (setf n2sv n2)
      (cond
        ((> dtrd1 0)
         (setf ind1 1))
        (t
         (setf ind1 n1)))
      (cond
        ((> dtrd2 0)
         (setf ind2 (f2cl-lib:int-add 1 n1)))
        (t
         (setf ind2 (f2cl-lib:int-add n1 n2))))
      (setf i 1)
 label10
      (cond
        ((and (> n1sv 0) (> n2sv 0))
         (cond
           ((<= (f2cl-lib:fref a (ind1) ((1 *)))
                (f2cl-lib:fref a (ind2) ((1 *))))
            (setf (f2cl-lib:fref indx-%data% (i) ((1 *)) indx-%offset%) ind1)
            (setf i (f2cl-lib:int-add i 1))
            (setf ind1 (f2cl-lib:int-add ind1 dtrd1))
            (setf n1sv (f2cl-lib:int-sub n1sv 1)))
           (t
            (setf (f2cl-lib:fref indx-%data% (i) ((1 *)) indx-%offset%) ind2)
            (setf i (f2cl-lib:int-add i 1))
            (setf ind2 (f2cl-lib:int-add ind2 dtrd2))
            (setf n2sv (f2cl-lib:int-sub n2sv 1))))
         (go label10)))
      (cond
        ((= n1sv 0)
         (f2cl-lib:fdo (n1sv 1 (f2cl-lib:int-add n1sv 1))
                       ((> n1sv n2sv) nil)
           (tagbody
             (setf (f2cl-lib:fref indx-%data% (i) ((1 *)) indx-%offset%) ind2)
             (setf i (f2cl-lib:int-add i 1))
```

```
              (setf ind2 (f2cl-lib:int-add ind2 dtrd2)))))
          (t
           (f2cl-lib:fdo (n2sv 1 (f2cl-lib:int-add n2sv 1))
                         ((> n2sv n1sv) nil)
             (tagbody
               (setf (f2cl-lib:fref indx-%data% (i) ((1 *)) indx-%offset%) ind1)
               (setf i (f2cl-lib:int-add i 1))
               (setf ind1 (f2cl-lib:int-add ind1 dtrd1))))))
   end_label
        (return (values nil nil nil nil nil nil)))))
```

7.40 dlange LAPACK

⟨*dlange.input*⟩≡

```
)set break resume
)sys rm -f dlange.output
)spool dlange.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlange.help*⟩≡

```
  ========================================================================
  dlange examples
  ========================================================================

  ========================================================================
  Man Page Details
  ========================================================================

  NAME
         DLANGE  -  the  value  of  the  one norm, or the Frobenius norm, or the
         infinity norm, or the element of  largest  absolute  value  of  a  real
         matrix A

  SYNOPSIS
         DOUBLE PRECISION FUNCTION DLANGE( NORM, M, N, A, LDA, WORK )

             CHARACTER    NORM

             INTEGER      LDA, M, N

             DOUBLE       PRECISION A( LDA, * ), WORK( * )

  PURPOSE
         DLANGE   returns  the value of the one norm,  or the Frobenius norm, or
         the  infinity norm,  or the  element of  largest absolute value   of  a
         real matrix A.


  DESCRIPTION
         DLANGE returns the value

            DLANGE = ( max(abs(A(i,j))), NORM = 'M' or 'm'
                     (
                     ( norm1(A),         NORM = '1', 'O' or 'o'
                     (
                     ( normI(A),         NORM = 'I' or 'i'
                     (
                     ( normF(A),         NORM = 'F', 'f', 'E' or 'e'

         where   norm1   denotes the  one norm of a matrix (maximum column sum),
         normI  denotes the  infinity norm  of a matrix  (maximum row  sum)  and
         normF   denotes  the  Frobenius norm of a matrix (square root of sum of
         squares).  Note that  max(abs(A(i,j)))   is  not  a  consistent  matrix
         norm.
```

```
ARGUMENTS
       NORM    (input) CHARACTER*1
               Specifies the value to be returned in DLANGE as described
               above.

       M       (input) INTEGER
               The number of rows of the matrix A.  M >= 0.  When M = 0,
               DLANGE is set to zero.

       N       (input) INTEGER
               The number of columns of the matrix A.  N >= 0.  When N = 0,
               DLANGE is set to zero.

       A       (input) DOUBLE PRECISION array, dimension (LDA,N)
               The m by n matrix A.

       LDA     (input) INTEGER
               The leading dimension of the array A.  LDA >= max(M,1).

       WORK    (workspace) DOUBLE PRECISION array, dimension (MAX(1,LWORK)),
               where LWORK >= M when NORM = 'I'; otherwise, WORK is not refer-
               enced.
```

⟨*LAPACK dlange*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dlange (norm m n a lda work)
    (declare (type (simple-array double-float (*)) work a)
             (type fixnum lda n m)
             (type character norm))
    (f2cl-lib:with-multi-array-data
        ((norm character norm-%data% norm-%offset%)
         (a double-float a-%data% a-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((scale 0.0) (sum 0.0) (value 0.0) (i 0) (j 0) (dlange 0.0))
        (declare (type fixnum i j)
                 (type (double-float) scale sum value dlange))
        (cond
          ((= (min (the fixnum m) (the fixnum n)) 0)
           (setf value zero))
          ((char-equal norm #\M)
           (setf value zero)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i m) nil)
                 (tagbody
                   (setf value
                           (max value
                                (abs
                                 (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%)))))))))
          ((or (char-equal norm #\O) (f2cl-lib:fstring-= norm "1"))
           (setf value zero)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf sum zero)
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i m) nil)
                 (tagbody
                   (setf sum
                           (+ sum
                              (abs
                               (f2cl-lib:fref a-%data%
```

```
                                        (i j)
                                        ((1 lda) (1 *))
                                        a-%offset%))))))
      (setf value (max value sum)))))
 ((char-equal norm #\I)
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                ((> i m) nil)
    (tagbody
      (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
              zero)))
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                ((> j n) nil)
    (tagbody
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i m) nil)
        (tagbody
          (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
                  (+
                   (f2cl-lib:fref work-%data%
                                  (i)
                                  ((1 *))
                                  work-%offset%)
                   (abs
                    (f2cl-lib:fref a-%data%
                                   (i j)
                                   ((1 lda) (1 *))
                                   a-%offset%))))))))
  (setf value zero)
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                ((> i m) nil)
    (tagbody
      (setf value
              (max value
                   (f2cl-lib:fref work-%data%
                                  (i)
                                  ((1 *))
                                  work-%offset%))))))
 ((or (char-equal norm #\F) (char-equal norm #\E))
  (setf scale zero)
  (setf sum one)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                ((> j n) nil)
    (tagbody
      (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
          (dlassq m
           (f2cl-lib:array-slice a double-float (1 j) ((1 lda) (1 *)))
```

```
                    1 scale sum)
                  (declare (ignore var-0 var-1 var-2))
                  (setf scale var-3)
                  (setf sum var-4))))
           (setf value (* scale (f2cl-lib:fsqrt sum)))))
        (setf dlange value)
  end_label
        (return (values dlange nil nil nil nil nil nil))))))
```

7.41 dlanhs LAPACK

⟨*dlanhs.input*⟩≡

```
)set break resume
)sys rm -f dlanhs.output
)spool dlanhs.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlanhs.help*⟩≡

```
==========================================================================
dlanhs examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

NAME
       DLANHS  -  the  value  of  the  one norm, or the Frobenius norm, or the
       infinity norm, or the element of largest absolute value of a Hessenberg
       matrix A

SYNOPSIS
       DOUBLE PRECISION FUNCTION DLANHS( NORM, N, A, LDA, WORK )

           CHARACTER    NORM

           INTEGER      LDA, N

           DOUBLE       PRECISION A( LDA, * ), WORK( * )

PURPOSE
       DLANHS   returns  the value of the one norm,  or the Frobenius norm, or
       the  infinity norm,  or the  element of  largest absolute value   of  a
       Hessenberg matrix A.


DESCRIPTION
       DLANHS returns the value

          DLANHS = ( max(abs(A(i,j))), NORM = 'M' or 'm'
                   (
                   ( norm1(A),         NORM = '1', 'O' or 'o'
                   (
                   ( normI(A),         NORM = 'I' or 'i'
                   (
                   ( normF(A),         NORM = 'F', 'f', 'E' or 'e'

       where   norm1   denotes the  one norm of a matrix (maximum column sum),
       normI  denotes the  infinity norm  of a matrix  (maximum row  sum)  and
       normF   denotes  the  Frobenius norm of a matrix (square root of sum of
       squares).  Note that  max(abs(A(i,j)))   is  not  a  consistent  matrix
       norm.
```

```
ARGUMENTS
       NORM    (input) CHARACTER*1
               Specifies  the  value  to  be  returned  in DLANHS as described
               above.

       N       (input) INTEGER
               The order of the matrix A.  N >= 0.  When N = 0, DLANHS is  set
               to zero.

       A       (input) DOUBLE PRECISION array, dimension (LDA,N)
               The  n  by n upper Hessenberg matrix A; the part of A below the
               first sub-diagonal is not referenced.

       LDA     (input) INTEGER
               The leading dimension of the array A.  LDA >= max(N,1).

       WORK    (workspace) DOUBLE PRECISION array, dimension (MAX(1,LWORK)),
               where LWORK >= N when NORM = 'I'; otherwise, WORK is not refer-
               enced.
```

⟨*LAPACK dlanhs*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dlanhs (norm n a lda work)
    (declare (type (simple-array double-float (*)) work a)
             (type fixnum lda n)
             (type character norm))
    (f2cl-lib:with-multi-array-data
        ((norm character norm-%data% norm-%offset%)
         (a double-float a-%data% a-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((scale 0.0) (sum 0.0) (value 0.0) (i 0) (j 0) (dlanhs 0.0))
        (declare (type fixnum i j)
                 (type (double-float) scale sum value dlanhs))
        (cond
          ((= n 0)
           (setf value zero))
          ((char-equal norm #\M)
           (setf value zero)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i
                                 (min (the fixnum n)
                                      (the fixnum
                                           (f2cl-lib:int-add j 1))))
                              nil)
                 (tagbody
                   (setf value
                           (max value
                                (abs
                                 (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%)))))))))
          ((or (char-equal norm #\O) (f2cl-lib:fstring-= norm "1"))
           (setf value zero)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf sum zero)
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i
                                 (min (the fixnum n)
```

```
                                  (the fixnum
                                       (f2cl-lib:int-add j 1))))
                         nil)
              (tagbody
                (setf sum
                        (+ sum
                           (abs
                            (f2cl-lib:fref a-%data%
                                           (i j)
                                           ((1 lda) (1 *))
                                           a-%offset%))))))
            (setf value (max value sum)))))
        ((char-equal norm #\I)
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i n) nil)
           (tagbody
             (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
                     zero)))
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i
                               (min (the fixnum n)
                                    (the fixnum
                                         (f2cl-lib:int-add j 1))))
                            nil)
               (tagbody
                 (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
                         (+
                          (f2cl-lib:fref work-%data%
                                         (i)
                                         ((1 *))
                                         work-%offset%)
                          (abs
                           (f2cl-lib:fref a-%data%
                                          (i j)
                                          ((1 lda) (1 *))
                                          a-%offset%)))))))))
         (setf value zero)
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i n) nil)
           (tagbody
             (setf value
                     (max value
                          (f2cl-lib:fref work-%data%
```

```
                                        (i)
                                        ((1 *))
                                        work-%offset%))))))
          ((or (char-equal norm #\F) (char-equal norm #\E))
           (setf scale zero)
           (setf sum one)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                   (dlassq
                    (min (the fixnum n)
                         (the fixnum (f2cl-lib:int-add j 1)))
                    (f2cl-lib:array-slice a double-float (1 j) ((1 lda) (1 *)))
                    1 scale sum)
                 (declare (ignore var-0 var-1 var-2))
                 (setf scale var-3)
                 (setf sum var-4))))
             (setf value (* scale (f2cl-lib:fsqrt sum)))))
        (setf dlanhs value)
 end_label
        (return (values dlanhs nil nil nil nil nil))))))
```

7.42 dlanst LAPACK

⟨*dlanst.input*⟩≡

```
)set break resume
)sys rm -f dlanst.output
)spool dlanst.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlanst.help*⟩≡

```
  ======================================================================
  dlanst examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLANST  -  the  value  of  the  one norm, or the Frobenius norm, or the
         infinity norm, or the element of largest absolute value of a real  sym-
         metric tridiagonal matrix A

  SYNOPSIS
         DOUBLE PRECISION FUNCTION DLANST( NORM, N, D, E )

             CHARACTER    NORM

             INTEGER      N

             DOUBLE       PRECISION D( * ), E( * )

  PURPOSE
         DLANST   returns  the value of the one norm,  or the Frobenius norm, or
         the  infinity norm,  or the  element of  largest absolute value   of  a
         real symmetric tridiagonal matrix A.


  DESCRIPTION
         DLANST returns the value

            DLANST = ( max(abs(A(i,j))), NORM = 'M' or 'm'
                     (
                     ( norm1(A),         NORM = '1', 'O' or 'o'
                     (
                     ( normI(A),         NORM = 'I' or 'i'
                     (
                     ( normF(A),         NORM = 'F', 'f', 'E' or 'e'

         where   norm1   denotes the  one norm of a matrix (maximum column sum),
         normI  denotes the  infinity norm  of a matrix  (maximum row  sum)  and
         normF   denotes  the  Frobenius norm of a matrix (square root of sum of
         squares).  Note that  max(abs(A(i,j)))   is  not  a  consistent  matrix
         norm.
```

```
ARGUMENTS
       NORM    (input) CHARACTER*1
               Specifies the value to be returned in DLANST as described
               above.

       N       (input) INTEGER
               The order of the matrix A.  N >= 0.  When N = 0, DLANST is  set
               to zero.

       D       (input) DOUBLE PRECISION array, dimension (N)
               The diagonal elements of A.

       E       (input) DOUBLE PRECISION array, dimension (N-1)
               The (n-1) sub-diagonal or super-diagonal elements of A.
```

⟨*LAPACK dlanst*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dlanst (norm n d e)
    (declare (type (simple-array double-float (*)) e d)
             (type fixnum n)
             (type character norm))
    (f2cl-lib:with-multi-array-data
        ((norm character norm-%data% norm-%offset%)
         (d double-float d-%data% d-%offset%)
         (e double-float e-%data% e-%offset%))
      (prog ((anorm 0.0) (scale 0.0) (sum 0.0) (i 0) (dlanst 0.0))
        (declare (type fixnum i)
                 (type (double-float) anorm scale sum dlanst))
        (cond
          ((<= n 0)
           (setf anorm zero))
          ((char-equal norm #\M)
           (setf anorm (abs (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)))
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                         ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
             (tagbody
               (setf anorm
                       (max anorm
                            (abs
                             (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))))
               (setf anorm
                       (max anorm
                        (abs
                         (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)))))))
          ((or (char-equal norm #\O) (f2cl-lib:fstring-= norm "1") (char-equal norm #\I))
           (cond
             ((= n 1)
              (setf anorm
                      (abs (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))))
             (t
              (setf anorm
                      (max
                       (+ (abs (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))
                          (abs
                           (f2cl-lib:fref e-%data% (1) ((1 *)) e-%offset%)))
                       (+
                        (abs
                         (f2cl-lib:fref e-%data%
                                        ((f2cl-lib:int-sub n 1))
```

```
                                        ((1 *))
                                        e-%offset%))
                              (abs (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)))))
                 (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                               ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1)))
                                nil)
                   (tagbody
                     (setf anorm
                             (max anorm
                                  (+
                                   (abs
                                    (f2cl-lib:fref d-%data%
                                                   (i)
                                                   ((1 *))
                                                   d-%offset%))
                                   (abs
                                    (f2cl-lib:fref e-%data%
                                                   (i)
                                                   ((1 *))
                                                   e-%offset%))
                                   (abs
                                    (f2cl-lib:fref e-%data%
                                                   ((f2cl-lib:int-sub i 1))
                                                   ((1 *))
                                                   e-%offset%)))))))))
           ((or (char-equal norm #\F) (char-equal norm #\E))
            (setf scale zero)
            (setf sum one)
            (cond
              ((> n 1)
               (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                   (dlassq (f2cl-lib:int-sub n 1) e 1 scale sum)
                 (declare (ignore var-0 var-1 var-2))
                 (setf scale var-3)
                 (setf sum var-4))
               (setf sum (* 2 sum))))
            (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                (dlassq n d 1 scale sum)
              (declare (ignore var-0 var-1 var-2))
              (setf scale var-3)
              (setf sum var-4))
            (setf anorm (* scale (f2cl-lib:fsqrt sum)))))
         (setf dlanst anorm)
 end_label
         (return (values dlanst nil nil nil nil))))))
```

7.43 dlanv2 LAPACK

⟨*dlanv2.input*⟩≡

```
)set break resume
)sys rm -f dlanv2.output
)spool dlanv2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlanv2.help*⟩≡

```
==============================================================================
dlanv2 examples
==============================================================================

==============================================================================
Man Page Details
==============================================================================

NAME
       DLANV2  -  the Schur factorization of a real 2-by-2 nonsymmetric matrix
       in standard form

SYNOPSIS
       SUBROUTINE DLANV2( A, B, C, D, RT1R, RT1I, RT2R, RT2I, CS, SN )

           DOUBLE         PRECISION A, B, C, CS, D, RT1I, RT1R, RT2I, RT2R, SN

PURPOSE
       DLANV2  computes  the Schur factorization of a real 2-by-2 nonsymmetric
       matrix in standard form:

            [ A  B ] = [ CS -SN ] [ AA  BB ] [ CS  SN ]
            [ C  D ]   [ SN  CS ] [ CC  DD ] [-SN  CS ]

       where either
       1) CC = 0 so that AA and DD are real eigenvalues of the matrix,  or  2)
       AA = DD and BB*CC < 0, so that AA + or - sqrt(BB*CC) are complex conju-
       gate eigenvalues.


ARGUMENTS
       A       (input/output) DOUBLE PRECISION
               B       (input/output) DOUBLE PRECISION C        (input/output)
               DOUBLE  PRECISION  D        (input/output)  DOUBLE PRECISION On
               entry, the elements of the input matrix.   On  exit,  they  are
               overwritten by the elements of the standardised Schur form.

       RT1R    (output) DOUBLE PRECISION
               RT1I     (output) DOUBLE PRECISION RT2R    (output) DOUBLE PRE-
               CISION RT2I    (output) DOUBLE PRECISION The real and imaginary
               parts of the eigenvalues. If the eigenvalues are a complex con-
               jugate pair, RT1I > 0.

       CS      (output) DOUBLE PRECISION
               SN      (output) DOUBLE PRECISION Parameters  of  the  rotation
```

matrix.

⟨*LAPACK dlanv2*⟩≡

```
(let* ((zero 0.0) (half 0.5) (one 1.0) (multpl 4.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 0.5 0.5) half)
           (type (double-float 1.0 1.0) one)
           (type (double-float 4.0 4.0) multpl))
  (defun dlanv2 (a b c d rt1r rt1i rt2r rt2i cs sn)
    (declare (type (double-float) sn cs rt2i rt2r rt1i rt1r d c b a))
    (prog ((aa 0.0) (bb 0.0) (bcmax 0.0) (bcmis 0.0) (cc 0.0) (cs1 0.0)
           (dd 0.0) (eps 0.0) (p 0.0) (sab 0.0) (sac 0.0) (scale 0.0)
           (sigma 0.0) (sn1 0.0) (tau 0.0) (temp 0.0) (z 0.0))
      (declare (type (double-float) aa bb bcmax bcmis cc cs1 dd eps p sab sac
                                    scale sigma sn1 tau temp z))
      (setf eps (dlamch "P"))
      (cond
        ((= c zero)
         (setf cs one)
         (setf sn zero)
         (go label10))
        ((= b zero)
         (setf cs zero)
         (setf sn one)
         (setf temp d)
         (setf d a)
         (setf a temp)
         (setf b (- c))
         (setf c zero)
         (go label10))
        ((and (= (+ a (- d)) zero)
              (/= (f2cl-lib:sign one b) (f2cl-lib:sign one c)))
         (setf cs one)
         (setf sn zero)
         (go label10))
        (t
         (setf temp (- a d))
         (setf p (* half temp))
         (setf bcmax (max (abs b) (abs c)))
         (setf bcmis
                 (* (min (abs b) (abs c))
                    (f2cl-lib:sign one b)
                    (f2cl-lib:sign one c)))
         (setf scale (max (abs p) bcmax))
         (setf z (+ (* (/ p scale) p) (* (/ bcmax scale) bcmis)))
         (cond
           ((>= z (* multpl eps))
            (setf z
```

```
            (+ p
               (f2cl-lib:sign
                (* (f2cl-lib:fsqrt scale) (f2cl-lib:fsqrt z))
                p)))
      (setf a (+ d z))
      (setf d (- d (* (/ bcmax z) bcmis)))
      (setf tau (dlapy2 c z))
      (setf cs (/ z tau))
      (setf sn (/ c tau))
      (setf b (- b c))
      (setf c zero))
     (t
      (setf sigma (+ b c))
      (setf tau (dlapy2 sigma temp))
      (setf cs (f2cl-lib:fsqrt (* half (+ one (/ (abs sigma) tau)))))
      (setf sn (* (- (/ p (* tau cs))) (f2cl-lib:sign one sigma)))
      (setf aa (+ (* a cs) (* b sn)))
      (setf bb (+ (* (- a) sn) (* b cs)))
      (setf cc (+ (* c cs) (* d sn)))
      (setf dd (+ (* (- c) sn) (* d cs)))
      (setf a (+ (* aa cs) (* cc sn)))
      (setf b (+ (* bb cs) (* dd sn)))
      (setf c (+ (* (- aa) sn) (* cc cs)))
      (setf d (+ (* (- bb) sn) (* dd cs)))
      (setf temp (* half (+ a d)))
      (setf a temp)
      (setf d temp)
      (cond
        ((/= c zero)
         (cond
           ((/= b zero)
            (cond
              ((= (f2cl-lib:sign one b) (f2cl-lib:sign one c))
               (setf sab (f2cl-lib:fsqrt (abs b)))
               (setf sac (f2cl-lib:fsqrt (abs c)))
               (setf p (f2cl-lib:sign (* sab sac) c))
               (setf tau (/ one (f2cl-lib:fsqrt (abs (+ b c)))))
               (setf a (+ temp p))
               (setf d (- temp p))
               (setf b (- b c))
               (setf c zero)
               (setf cs1 (* sab tau))
               (setf sn1 (* sac tau))
               (setf temp (- (* cs cs1) (* sn sn1)))
               (setf sn (+ (* cs sn1) (* sn cs1)))
               (setf cs temp))))
```

```
                (t
                 (setf b (- c))
                 (setf c zero)
                 (setf temp cs)
                 (setf cs (- sn))
                 (setf sn temp)))))))))
  label10
       (setf rt1r a)
       (setf rt2r d)
       (cond
         ((= c zero)
          (setf rt1i zero)
          (setf rt2i zero))
         (t
          (setf rt1i (* (f2cl-lib:fsqrt (abs b)) (f2cl-lib:fsqrt (abs c))))
          (setf rt2i (- rt1i))))
  end_label
       (return (values a b c d rt1r rt1i rt2r rt2i cs sn)))))
```

7.44 dlapy2 LAPACK

⟨*dlapy2.input*⟩≡

```
)set break resume
)sys rm -f dlapy2.output
)spool dlapy2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlapy2.help*⟩≡

```
  ======================================================================
  dlapy2 examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLAPY2 - sqrt(x**2+y**2), taking care not to cause unnecessary overflow

  SYNOPSIS
         DOUBLE PRECISION FUNCTION DLAPY2( X, Y )

             DOUBLE       PRECISION X, Y

  PURPOSE
         DLAPY2 returns sqrt(x**2+y**2), taking care not  to  cause  unnecessary
         overflow.


  ARGUMENTS
         X       (input) DOUBLE PRECISION
                 Y        (input)  DOUBLE PRECISION X and Y specify the values x
                 and y.
```

⟨*LAPACK dlapy2*⟩≡
```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dlapy2 (x y)
    (declare (type (double-float) y x))
    (prog ((w 0.0) (xabs 0.0) (yabs 0.0) (z 0.0) (dlapy2 0.0))
      (declare (type (double-float) w xabs yabs z dlapy2))
      (setf xabs (abs x))
      (setf yabs (abs y))
      (setf w (max xabs yabs))
      (setf z (min xabs yabs))
      (cond
        ((= z zero)
         (setf dlapy2 w))
        (t
         (setf dlapy2 (* w (f2cl-lib:fsqrt (+ one (expt (/ z w) 2)))))))
      (return (values dlapy2 nil nil)))))
```

7.45 dlaqtr LAPACK

⟨*dlaqtr.input*⟩≡
```
)set break resume
)sys rm -f dlaqtr.output
)spool dlaqtr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlaqtr.help*⟩≡

```
  ======================================================================
  dlaqtr examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLAQTR - the real quasi-triangular system   op(T)*p = scale*c, if LREAL
         = .TRUE

  SYNOPSIS
         SUBROUTINE DLAQTR( LTRAN, LREAL, N, T, LDT, B, W, SCALE, X, WORK,  INFO
                           )

             LOGICAL         LREAL, LTRAN

             INTEGER         INFO, LDT, N

             DOUBLE          PRECISION SCALE, W

             DOUBLE          PRECISION B( * ), T( LDT, * ), WORK( * ), X( * )

  PURPOSE
         DLAQTR solves the real quasi-triangular system

         or the complex quasi-triangular systems

                    op(T + iB)*(p+iq) = scale*(c+id),  if LREAL = .FALSE.

         in real arithmetic, where T is upper quasi-triangular.
         If  LREAL = .FALSE., then the first diagonal block of T must be 1 by 1,
         B is the specially structured matrix

                        B = [ b(1) b(2) ... b(n) ]
                            [       w            ]
                            [           w        ]
                            [              .     ]
                            [                 w  ]

         op(A) = A or A', A' denotes the conjugate transpose of
         matrix A.

         On input, X = [ c ].  On output, X = [ p ].
```

[d] [q]

This subroutine is designed for the condition number estimation in routine DTRSNA.

ARGUMENTS

LTRAN (input) LOGICAL
On entry, LTRAN specifies the option of conjugate transpose: = .FALSE., op(T+i*B) = T+i*B, = .TRUE., op(T+i*B) = (T+i*B)'.

LREAL (input) LOGICAL
On entry, LREAL specifies the input matrix structure: = .FALSE., the input is complex = .TRUE., the input is real

N (input) INTEGER
On entry, N specifies the order of T+i*B. N >= 0.

T (input) DOUBLE PRECISION array, dimension (LDT,N)
On entry, T contains a matrix in Schur canonical form. If LREAL = .FALSE., then the first diagonal block of T mu be 1 by 1.

LDT (input) INTEGER
The leading dimension of the matrix T. LDT >= max(1,N).

B (input) DOUBLE PRECISION array, dimension (N)
On entry, B contains the elements to form the matrix B as described above. If LREAL = .TRUE., B is not referenced.

W (input) DOUBLE PRECISION
On entry, W is the diagonal element of the matrix B. If LREAL = .TRUE., W is not referenced.

SCALE (output) DOUBLE PRECISION
On exit, SCALE is the scale factor.

X (input/output) DOUBLE PRECISION array, dimension (2*N)
On entry, X contains the right hand side of the system. On exit, X is overwritten by the solution.

WORK (workspace) DOUBLE PRECISION array, dimension (N)

INFO (output) INTEGER

```
On exit, INFO is set to 0: successful exit.
1: the some diagonal 1 by 1 block has been perturbed by a small
number  SMIN to keep nonsingularity.  2: the some diagonal 2 by
2 block has been perturbed by a small number in DLALN2 to  keep
nonsingularity.   NOTE: In the interests of speed, this routine
does not check the inputs for errors.
```

⟨*LAPACK dlaqtr*⟩≡

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dlaqtr (ltran lreal n t$ ldt b w scale x work info)
    (declare (type (double-float) scale w)
             (type (simple-array double-float (*)) work x b t$)
             (type fixnum info ldt n)
             (type (member t nil) lreal ltran))
    (f2cl-lib:with-multi-array-data
        ((t$ double-float t$-%data% t$-%offset%)
         (b double-float b-%data% b-%offset%)
         (x double-float x-%data% x-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((d (make-array 4 :element-type 'double-float))
             (v (make-array 4 :element-type 'double-float)) (bignum 0.0)
             (eps 0.0) (rec 0.0) (scaloc 0.0) (si 0.0) (smin 0.0) (sminw 0.0)
             (smlnum 0.0) (sr 0.0) (tjj 0.0) (tmp 0.0) (xj 0.0) (xmax 0.0)
             (xnorm 0.0) (z 0.0) (i 0) (ierr 0) (j 0) (j1 0) (j2 0) (jnext 0)
             (k 0) (n1 0) (n2 0) (notran nil))
        (declare (type (simple-array double-float (4)) d v)
                 (type (double-float) bignum eps rec scaloc si smin sminw
                                      smlnum sr tjj tmp xj xmax xnorm z)
                 (type fixnum i ierr j j1 j2 jnext k n1 n2)
                 (type (member t nil) notran))
        (setf notran (not ltran))
        (setf info 0)
        (if (= n 0) (go end_label))
        (setf eps (dlamch "P"))
        (setf smlnum (/ (dlamch "S") eps))
        (setf bignum (/ one smlnum))
        (setf xnorm (dlange "M" n n t$ ldt d))
        (if (not lreal)
            (setf xnorm (max xnorm (abs w) (dlange "M" n 1 b n d))))
        (setf smin (max smlnum (* eps xnorm)))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) zero)
        (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                      ((> j n) nil)
          (tagbody
            (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
                    (dasum (f2cl-lib:int-sub j 1)
                     (f2cl-lib:array-slice t$
                                           double-float
                                           (1 j)
                                           ((1 ldt) (1 *)))
                     1))))
```

```
(cond
  ((not lreal)
   (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                 ((> i n) nil)
     (tagbody
       (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
         (+ (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
          (abs
           (f2cl-lib:fref b-%data% (i) ((1 *)) b-%offset%))))))));tpd
(setf n2 (f2cl-lib:int-mul 2 n))
(setf n1 n)
(if (not lreal) (setf n1 n2))
(setf k (idamax n1 x 1))
(setf xmax (abs (f2cl-lib:fref x-%data% (k) ((1 *)) x-%offset%)))
(setf scale one)
(cond
  ((> xmax bignum)
   (setf scale (/ bignum xmax))
   (dscal n1 scale x 1)
   (setf xmax bignum)))
(cond
  (lreal
   (cond
     (notran
      (setf jnext n)
      (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                    ((> j 1) nil)
        (tagbody
          (if (> j jnext) (go label30))
          (setf j1 j)
          (setf j2 j)
          (setf jnext (f2cl-lib:int-sub j 1))
          (cond
            ((> j 1)
             (cond
               ((/=
                 (f2cl-lib:fref t$
                                (j
                                 (f2cl-lib:int-add j
                                                   (f2cl-lib:int-sub
                                                    1)))
                                ((1 ldt) (1 *)))
                 zero)
                (setf j1 (f2cl-lib:int-sub j 1))
                (setf jnext (f2cl-lib:int-sub j 2))))))
          (cond
```

```
((= j1 j2)
 (setf xj
       (abs
        (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)))
 (setf tjj
         (abs
          (f2cl-lib:fref t$-%data%
                         (j1 j1)
                         ((1 ldt) (1 *))
                         t$-%offset%)))
 (setf tmp
         (f2cl-lib:fref t$-%data%
                        (j1 j1)
                        ((1 ldt) (1 *))
                        t$-%offset%))
 (cond
   ((< tjj smin)
    (setf tmp smin)
    (setf tjj smin)
    (setf info 1)))
 (if (= xj zero) (go label30))
 (cond
   ((< tjj one)
    (cond
      ((> xj (* bignum tjj))
       (setf rec (/ one xj))
       (dscal n rec x 1)
       (setf scale (* scale rec))
       (setf xmax (* xmax rec))))))
 (setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
         (/
          (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
          tmp))
 (setf xj
       (abs
        (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)))
 (cond
   ((> xj one)
    (setf rec (/ one xj))
    (cond
      ((> (f2cl-lib:fref work (j1) ((1 *)))
          (* (+ bignum (- xmax)) rec))
       (dscal n rec x 1)
       (setf scale (* scale rec))))))
 (cond
   ((> j1 1)
```

```
    (daxpy (f2cl-lib:int-sub j1 1)
     (- (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%))
     (f2cl-lib:array-slice t$
                           double-float
                           (1 j1)
                           ((1 ldt) (1 *)))
     1 x 1)
    (setf k (idamax (f2cl-lib:int-sub j1 1) x 1))
    (setf xmax
            (abs
             (f2cl-lib:fref x-%data%
                            (k)
                            ((1 *))
                            x-%offset%))))))
(t
 (setf (f2cl-lib:fref d (1 1) ((1 2) (1 2)))
         (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%))
 (setf (f2cl-lib:fref d (2 1) ((1 2) (1 2)))
         (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10 var-11 var-12 var-13 var-14
        var-15 var-16 var-17)
     (dlaln2 nil 2 1 smin one
      (f2cl-lib:array-slice t$
                            double-float
                            (j1 j1)
                            ((1 ldt) (1 *)))
      ldt one one d 2 zero zero v 2 scaloc xnorm ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12 var-13 var-14))
   (setf scaloc var-15)
   (setf xnorm var-16)
   (setf ierr var-17))
 (if (/= ierr 0) (setf info 2))
 (cond
   ((/= scaloc one)
    (dscal n scaloc x 1)
    (setf scale (* scale scaloc))))
 (setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
         (f2cl-lib:fref v (1 1) ((1 2) (1 2))))
 (setf (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%)
         (f2cl-lib:fref v (2 1) ((1 2) (1 2))))
 (setf xj
         (max (abs (f2cl-lib:fref v (1 1) ((1 2) (1 2))))
```

```
                                    (abs (f2cl-lib:fref v (2 1) ((1 2) (1 2))))))
                          (cond
                            ((> xj one)
                             (setf rec (/ one xj))
                             (cond
                               ((>
                                 (max (f2cl-lib:fref work (j1) ((1 *)))
                                      (f2cl-lib:fref work (j2) ((1 *))))
                                 (* (+ bignum (- xmax)) rec))
                                (dscal n rec x 1)
                                (setf scale (* scale rec))))))
                          (cond
                            ((> j1 1)
                             (daxpy (f2cl-lib:int-sub j1 1)
                              (- (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%))
                              (f2cl-lib:array-slice t$
                                                    double-float
                                                    (1 j1)
                                                    ((1 ldt) (1 *)))
                              1 x 1)
                             (daxpy (f2cl-lib:int-sub j1 1)
                              (- (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%))
                              (f2cl-lib:array-slice t$
                                                    double-float
                                                    (1 j2)
                                                    ((1 ldt) (1 *)))
                              1 x 1)
                             (setf k (idamax (f2cl-lib:int-sub j1 1) x 1))
                             (setf xmax
                                     (abs
                                      (f2cl-lib:fref x-%data%
                                                     (k)
                                                     ((1 *))
                                                     x-%offset%)))))))
        label30)))
                    (t
                     (setf jnext 1)
                     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                   ((> j n) nil)
                       (tagbody
                         (if (< j jnext) (go label40))
                         (setf j1 j)
                         (setf j2 j)
                         (setf jnext (f2cl-lib:int-add j 1))
                         (cond
                           ((< j n)
```

```
                    (cond
                      ((/=
                        (f2cl-lib:fref t$
                                       ((f2cl-lib:int-add j 1) j)
                                       ((1 ldt) (1 *)))
                        zero)
                       (setf j2 (f2cl-lib:int-add j 1))
                       (setf jnext (f2cl-lib:int-add j 2))))))
                (cond
                  ((= j1 j2)
                   (setf xj
                           (abs
                            (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)))
                   (cond
                     ((> xmax one)
                      (setf rec (/ one xmax))
                      (cond
                        ((> (f2cl-lib:fref work (j1) ((1 *)))
                            (* (+ bignum (- xj)) rec))
                         (dscal n rec x 1)
                         (setf scale (* scale rec))
                         (setf xmax (* xmax rec))))))
                   (setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
                           (-
                            (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
                            (ddot (f2cl-lib:int-sub j1 1)
                             (f2cl-lib:array-slice t$
                                                   double-float
                                                   (1 j1)
                                                   ((1 ldt) (1 *)))
                             1 x 1)))
                   (setf xj
                          (abs
                           (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)))
                   (setf tjj
                           (abs
                            (f2cl-lib:fref t$-%data%
                                           (j1 j1)
                                           ((1 ldt) (1 *))
                                           t$-%offset%)))
                   (setf tmp
                           (f2cl-lib:fref t$-%data%
                                          (j1 j1)
                                          ((1 ldt) (1 *))
                                          t$-%offset%))
                   (cond
```

```
  ((< tjj smin)
   (setf tmp smin)
   (setf tjj smin)
   (setf info 1)))
 (cond
   ((< tjj one)
    (cond
      ((> xj (* bignum tjj))
       (setf rec (/ one xj))
       (dscal n rec x 1)
       (setf scale (* scale rec))
       (setf xmax (* xmax rec))))))
 (setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
         (/
          (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
          tmp))
 (setf xmax
         (max xmax
              (abs
               (f2cl-lib:fref x-%data%
                              (j1)
                              ((1 *))
                              x-%offset%)))))
(t
 (setf xj
         (max
          (abs
           (f2cl-lib:fref x-%data%
                          (j1)
                          ((1 *))
                          x-%offset%))
          (abs
           (f2cl-lib:fref x-%data%
                          (j2)
                          ((1 *))
                          x-%offset%))))
 (cond
   ((> xmax one)
    (setf rec (/ one xmax))
    (cond
      ((>
        (max (f2cl-lib:fref work (j2) ((1 *)))
             (f2cl-lib:fref work (j1) ((1 *))))
        (* (+ bignum (- xj)) rec))
       (dscal n rec x 1)
       (setf scale (* scale rec))
```

```
      (setf xmax (* xmax rec))))))
(setf (f2cl-lib:fref d (1 1) ((1 2) (1 2)))
        (-
         (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
         (ddot (f2cl-lib:int-sub j1 1)
          (f2cl-lib:array-slice t$
                                double-float
                                (1 j1)
                                ((1 ldt) (1 *)))
          1 x 1)))
(setf (f2cl-lib:fref d (2 1) ((1 2) (1 2)))
        (-
         (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%)
         (ddot (f2cl-lib:int-sub j1 1)
          (f2cl-lib:array-slice t$
                                double-float
                                (1 j2)
                                ((1 ldt) (1 *)))
          1 x 1)))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13 var-14
       var-15 var-16 var-17)
    (dlaln2 t 2 1 smin one
     (f2cl-lib:array-slice t$
                           double-float
                           (j1 j1)
                           ((1 ldt) (1 *)))
     ldt one one d 2 zero zero v 2 scaloc xnorm ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12 var-13 var-14))
  (setf scaloc var-15)
  (setf xnorm var-16)
  (setf ierr var-17))
(if (/= ierr 0) (setf info 2))
(cond
  ((/= scaloc one)
   (dscal n scaloc x 1)
   (setf scale (* scale scaloc))))
(setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
        (f2cl-lib:fref v (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%)
        (f2cl-lib:fref v (2 1) ((1 2) (1 2))))
(setf xmax
        (max
```

```
                                (abs
                                 (f2cl-lib:fref x-%data%
                                                (j1)
                                                ((1 *))
                                                x-%offset%))
                                (abs
                                 (f2cl-lib:fref x-%data%
                                                (j2)
                                                ((1 *))
                                                x-%offset%))
                                xmax))))
  label40)))))
           (t
            (setf sminw (max (* eps (abs w)) smin))
            (cond
              (notran
               (setf jnext n)
               (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                             ((> j 1) nil)
                 (tagbody
                   (if (> j jnext) (go label70))
                   (setf j1 j)
                   (setf j2 j)
                   (setf jnext (f2cl-lib:int-sub j 1))
                   (cond
                     ((> j 1)
                      (cond
                        ((/=
                          (f2cl-lib:fref t$
                                         (j
                                          (f2cl-lib:int-add j
                                                            (f2cl-lib:int-sub
                                                             1)))
                                         ((1 ldt) (1 *)))
                          zero)
                         (setf j1 (f2cl-lib:int-sub j 1))
                         (setf jnext (f2cl-lib:int-sub j 2))))))
                   (cond
                     ((= j1 j2)
                      (setf z w)
                      (if (= j1 1)
                          (setf z
                                  (f2cl-lib:fref b-%data%
                                                 (1)
                                                 ((1 *))
                                                 b-%offset%)))
```

```
(setf xj
        (+
         (abs
          (f2cl-lib:fref x-%data%
                         (j1)
                         ((1 *))
                         x-%offset%))
         (abs
          (f2cl-lib:fref x-%data%
                         ((f2cl-lib:int-add n j1))
                         ((1 *))
                         x-%offset%))))
(setf tjj
        (+
         (abs
          (f2cl-lib:fref t$-%data%
                         (j1 j1)
                         ((1 ldt) (1 *))
                         t$-%offset%))
         (abs z)))
(setf tmp
        (f2cl-lib:fref t$-%data%
                       (j1 j1)
                       ((1 ldt) (1 *))
                       t$-%offset%))
(cond
  ((< tjj sminw)
   (setf tmp sminw)
   (setf tjj sminw)
   (setf info 1)))
(if (= xj zero) (go label70))
(cond
  ((< tjj one)
   (cond
     ((> xj (* bignum tjj))
      (setf rec (/ one xj))
      (dscal n2 rec x 1)
      (setf scale (* scale rec))
      (setf xmax (* xmax rec))))))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
    (dladiv
     (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
     (f2cl-lib:fref x-%data%
                    ((f2cl-lib:int-add n j1))
                    ((1 *))
                    x-%offset%)
```

```
     tmp z sr si)
  (declare (ignore var-0 var-1 var-2 var-3))
  (setf sr var-4)
  (setf si var-5))
(setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%) sr)
(setf (f2cl-lib:fref x-%data%
                     ((f2cl-lib:int-add n j1))
                     ((1 *))
                     x-%offset%)
        si)
(setf xj
        (+
         (abs
          (f2cl-lib:fref x-%data%
                         (j1)
                         ((1 *))
                         x-%offset%))
         (abs
          (f2cl-lib:fref x-%data%
                         ((f2cl-lib:int-add n j1))
                         ((1 *))
                         x-%offset%))))
(cond
  ((> xj one)
   (setf rec (/ one xj))
   (cond
     ((> (f2cl-lib:fref work (j1) ((1 *)))
         (* (+ bignum (- xmax)) rec))
      (dscal n2 rec x 1)
      (setf scale (* scale rec))))))
(cond
  ((> j1 1)
   (daxpy (f2cl-lib:int-sub j1 1)
    (- (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%))
    (f2cl-lib:array-slice t$
                          double-float
                          (1 j1)
                          ((1 ldt) (1 *)))
    1 x 1)
   (daxpy (f2cl-lib:int-sub j1 1)
    (-
     (f2cl-lib:fref x-%data%
                    ((f2cl-lib:int-add n j1))
                    ((1 *))
                    x-%offset%))
    (f2cl-lib:array-slice t$
```

```
                        double-float
                        (1 j1)
                        ((1 ldt) (1 *)))
 1
 (f2cl-lib:array-slice x
                        double-float
                        ((+ n 1))
                        ((1 *)))
 1)
(setf (f2cl-lib:fref x-%data% (1) ((1 *)) x-%offset%)
        (+
         (f2cl-lib:fref x-%data%
                        (1)
                        ((1 *))
                        x-%offset%)
         (*
          (f2cl-lib:fref b-%data%
                         (j1)
                         ((1 *))
                         b-%offset%)
          (f2cl-lib:fref x-%data%
                         ((f2cl-lib:int-add n j1))
                         ((1 *))
                         x-%offset%))))
(setf (f2cl-lib:fref x-%data%
                     ((f2cl-lib:int-add n 1))
                     ((1 *))
                     x-%offset%)
        (-
         (f2cl-lib:fref x-%data%
                        ((f2cl-lib:int-add n 1))
                        ((1 *))
                        x-%offset%)
         (*
          (f2cl-lib:fref b-%data%
                         (j1)
                         ((1 *))
                         b-%offset%)
          (f2cl-lib:fref x-%data%
                         (j1)
                         ((1 *))
                         x-%offset%))))
(setf xmax zero)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k
                  (f2cl-lib:int-add j1
```

```
                                                (f2cl-lib:int-sub
                                                 1)))
                              nil)
                   (tagbody
                     (setf xmax
                             (max xmax
                                  (+
                                   (abs
                                    (f2cl-lib:fref x-%data%
                                                   (k)
                                                   ((1 *))
                                                   x-%offset%))
                                   (abs
                                    (f2cl-lib:fref x-%data%
                                                   ((f2cl-lib:int-add k
                                                                      n))
                                                   ((1 *))
                                                   x-%offset%)))))))))
            (t
             (setf (f2cl-lib:fref d (1 1) ((1 2) (1 2)))
                     (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%))
             (setf (f2cl-lib:fref d (2 1) ((1 2) (1 2)))
                     (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%))
             (setf (f2cl-lib:fref d (1 2) ((1 2) (1 2)))
                     (f2cl-lib:fref x-%data%
                                    ((f2cl-lib:int-add n j1))
                                    ((1 *))
                                    x-%offset%))
             (setf (f2cl-lib:fref d (2 2) ((1 2) (1 2)))
                     (f2cl-lib:fref x-%data%
                                    ((f2cl-lib:int-add n j2))
                                    ((1 *))
                                    x-%offset%))
             (multiple-value-bind
                   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                    var-8 var-9 var-10 var-11 var-12 var-13 var-14
                    var-15 var-16 var-17)
                 (dlaln2 nil 2 2 sminw one
                  (f2cl-lib:array-slice t$
                                        double-float
                                        (j1 j1)
                                        ((1 ldt) (1 *)))
                  ldt one one d 2 zero (- w) v 2 scaloc xnorm ierr)
               (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                                var-6 var-7 var-8 var-9 var-10 var-11
                                var-12 var-13 var-14))
```

```
  (setf scaloc var-15)
  (setf xnorm var-16)
  (setf ierr var-17))
(if (/= ierr 0) (setf info 2))
(cond
  ((/= scaloc one)
   (dscal (f2cl-lib:int-mul 2 n) scaloc x 1)
   (setf scale (* scaloc scale))))
(setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
        (f2cl-lib:fref v (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%)
        (f2cl-lib:fref v (2 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref x-%data%
                     ((f2cl-lib:int-add n j1))
                     ((1 *))
                     x-%offset%)
        (f2cl-lib:fref v (1 2) ((1 2) (1 2))))
(setf (f2cl-lib:fref x-%data%
                     ((f2cl-lib:int-add n j2))
                     ((1 *))
                     x-%offset%)
        (f2cl-lib:fref v (2 2) ((1 2) (1 2))))
(setf xj
        (max
         (+ (abs (f2cl-lib:fref v (1 1) ((1 2) (1 2))))
            (abs (f2cl-lib:fref v (1 2) ((1 2) (1 2)))))
         (+ (abs (f2cl-lib:fref v (2 1) ((1 2) (1 2))))
            (abs (f2cl-lib:fref v (2 2) ((1 2) (1 2)))))))
(cond
  ((> xj one)
   (setf rec (/ one xj))
   (cond
     ((>
       (max (f2cl-lib:fref work (j1) ((1 *)))
            (f2cl-lib:fref work (j2) ((1 *))))
       (* (+ bignum (- xmax)) rec))
      (dscal n2 rec x 1)
      (setf scale (* scale rec))))))
(cond
  ((> j1 1)
   (daxpy (f2cl-lib:int-sub j1 1)
    (- (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%))
    (f2cl-lib:array-slice t$
                          double-float
                          (1 j1)
                          ((1 ldt) (1 *)))
```

```
 1 x 1)
(daxpy (f2cl-lib:int-sub j1 1)
 (- (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%))
 (f2cl-lib:array-slice t$
                       double-float
                       (1 j2)
                       ((1 ldt) (1 *)))
 1 x 1)
(daxpy (f2cl-lib:int-sub j1 1)
 (-
  (f2cl-lib:fref x-%data%
                 ((f2cl-lib:int-add n j1))
                 ((1 *))
                 x-%offset%))
 (f2cl-lib:array-slice t$
                       double-float
                       (1 j1)
                       ((1 ldt) (1 *)))
 1
 (f2cl-lib:array-slice x
                       double-float
                       ((+ n 1))
                       ((1 *)))
 1)
(daxpy (f2cl-lib:int-sub j1 1)
 (-
  (f2cl-lib:fref x-%data%
                 ((f2cl-lib:int-add n j2))
                 ((1 *))
                 x-%offset%))
 (f2cl-lib:array-slice t$
                       double-float
                       (1 j2)
                       ((1 ldt) (1 *)))
 1
 (f2cl-lib:array-slice x
                       double-float
                       ((+ n 1))
                       ((1 *)))
 1)
(setf (f2cl-lib:fref x-%data% (1) ((1 *)) x-%offset%)
       (+
        (f2cl-lib:fref x-%data%
                       (1)
                       ((1 *))
                       x-%offset%)
```

```
                    (*
                     (f2cl-lib:fref b-%data%
                                    (j1)
                                    ((1 *))
                                    b-%offset%)
                     (f2cl-lib:fref x-%data%
                                    ((f2cl-lib:int-add n j1))
                                    ((1 *))
                                    x-%offset%))
                    (*
                     (f2cl-lib:fref b-%data%
                                    (j2)
                                    ((1 *))
                                    b-%offset%)
                     (f2cl-lib:fref x-%data%
                                    ((f2cl-lib:int-add n j2))
                                    ((1 *))
                                    x-%offset%))))
            (setf (f2cl-lib:fref x-%data%
                                 ((f2cl-lib:int-add n 1))
                                 ((1 *))
                                 x-%offset%)
                  (-
                   (f2cl-lib:fref x-%data%
                                  ((f2cl-lib:int-add n 1))
                                  ((1 *))
                                  x-%offset%)
                   (*
                    (f2cl-lib:fref b-%data%
                                   (j1)
                                   ((1 *))
                                   b-%offset%)
                    (f2cl-lib:fref x-%data%
                                   (j1)
                                   ((1 *))
                                   x-%offset%))
                   (*
                    (f2cl-lib:fref b-%data%
                                   (j2)
                                   ((1 *))
                                   b-%offset%)
                    (f2cl-lib:fref x-%data%
                                   (j2)
                                   ((1 *))
                                   x-%offset%))))
            (setf xmax zero)
```

```
                        (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                                      ((> k
                                          (f2cl-lib:int-add j1
                                                            (f2cl-lib:int-sub
                                                             1)))
                                       nil)
                          (tagbody
                            (setf xmax
                                    (max
                                     (+
                                      (abs
                                       (f2cl-lib:fref x-%data%
                                                      (k)
                                                      ((1 *))
                                                      x-%offset%))
                                      (abs
                                       (f2cl-lib:fref x-%data%
                                                      ((f2cl-lib:int-add k n))
                                                      ((1 *))
                                                      x-%offset%)))
                                     xmax))))))))
   label70)))
              (t
               (setf jnext 1)
               (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                             ((> j n) nil)
                 (tagbody
                   (if (< j jnext) (go label80))
                   (setf j1 j)
                   (setf j2 j)
                   (setf jnext (f2cl-lib:int-add j 1))
                   (cond
                     ((< j n)
                      (cond
                        ((/=
                          (f2cl-lib:fref t$
                                         ((f2cl-lib:int-add j 1) j)
                                         ((1 ldt) (1 *)))
                          zero)
                         (setf j2 (f2cl-lib:int-add j 1))
                         (setf jnext (f2cl-lib:int-add j 2))))))
                   (cond
                     ((= j1 j2)
                      (setf xj
                              (+
                               (abs
```

```
                (f2cl-lib:fref x-%data%
                               (j1)
                               ((1 *))
                               x-%offset%))
               (abs
                (f2cl-lib:fref x-%data%
                               ((f2cl-lib:int-add j1 n))
                               ((1 *))
                               x-%offset%))))
(cond
  ((> xmax one)
   (setf rec (/ one xmax))
   (cond
     ((> (f2cl-lib:fref work (j1) ((1 *)))
         (* (+ bignum (- xj)) rec))
      (dscal n2 rec x 1)
      (setf scale (* scale rec))
      (setf xmax (* xmax rec))))))
(setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
        (-
         (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
         (ddot (f2cl-lib:int-sub j1 1)
          (f2cl-lib:array-slice t$
                                double-float
                                (1 j1)
                                ((1 ldt) (1 *)))
          1 x 1)))
(setf (f2cl-lib:fref x-%data%
                     ((f2cl-lib:int-add n j1))
                     ((1 *))
                     x-%offset%)
        (-
         (f2cl-lib:fref x-%data%
                        ((f2cl-lib:int-add n j1))
                        ((1 *))
                        x-%offset%)
         (ddot (f2cl-lib:int-sub j1 1)
          (f2cl-lib:array-slice t$
                                double-float
                                (1 j1)
                                ((1 ldt) (1 *)))
          1
          (f2cl-lib:array-slice x
                                double-float
                                ((+ n 1))
                                ((1 *)))
```

```
          1)))
(cond
  ((> j1 1)
   (setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
         (-
          (f2cl-lib:fref x-%data%
                         (j1)
                         ((1 *))
                         x-%offset%)
          (*
           (f2cl-lib:fref b-%data%
                          (j1)
                          ((1 *))
                          b-%offset%)
           (f2cl-lib:fref x-%data%
                          ((f2cl-lib:int-add n 1))
                          ((1 *))
                          x-%offset%))))
   (setf (f2cl-lib:fref x-%data%
                        ((f2cl-lib:int-add n j1))
                        ((1 *))
                        x-%offset%)
         (+
          (f2cl-lib:fref x-%data%
                         ((f2cl-lib:int-add n j1))
                         ((1 *))
                         x-%offset%)
          (*
           (f2cl-lib:fref b-%data%
                          (j1)
                          ((1 *))
                          b-%offset%)
           (f2cl-lib:fref x-%data%
                          (1)
                          ((1 *))
                          x-%offset%))))))
(setf xj
        (+
         (abs
          (f2cl-lib:fref x-%data%
                         (j1)
                         ((1 *))
                         x-%offset%))
         (abs
          (f2cl-lib:fref x-%data%
                         ((f2cl-lib:int-add j1 n))
```

```
                          ((1 *))
                          x-%offset%))))
(setf z w)
(if (= j1 1)
    (setf z
            (f2cl-lib:fref b-%data%
                           (1)
                           ((1 *))
                           b-%offset%)))
(setf tjj
        (+
         (abs
          (f2cl-lib:fref t$-%data%
                         (j1 j1)
                         ((1 ldt) (1 *))
                         t$-%offset%))
         (abs z)))
(setf tmp
        (f2cl-lib:fref t$-%data%
                       (j1 j1)
                       ((1 ldt) (1 *))
                       t$-%offset%))
(cond
  ((< tjj sminw)
   (setf tmp sminw)
   (setf tjj sminw)
   (setf info 1)))
(cond
  ((< tjj one)
   (cond
     ((> xj (* bignum tjj))
      (setf rec (/ one xj))
      (dscal n2 rec x 1)
      (setf scale (* scale rec))
      (setf xmax (* xmax rec))))))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
    (dladiv
     (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
     (f2cl-lib:fref x-%data%
                    ((f2cl-lib:int-add n j1))
                    ((1 *))
                    x-%offset%)
     tmp (- z) sr si)
  (declare (ignore var-0 var-1 var-2 var-3))
  (setf sr var-4)
  (setf si var-5))
```

```
(setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%) sr)
(setf (f2cl-lib:fref x-%data%
                     ((f2cl-lib:int-add j1 n))
                     ((1 *))
                     x-%offset%)
      si)
(setf xmax
      (max
       (+
        (abs
         (f2cl-lib:fref x-%data%
                        (j1)
                        ((1 *))
                        x-%offset%))
        (abs
         (f2cl-lib:fref x-%data%
                        ((f2cl-lib:int-add j1 n))
                        ((1 *))
                        x-%offset%)))
       xmax)))
(t
 (setf xj
       (max
        (+
         (abs
          (f2cl-lib:fref x-%data%
                         (j1)
                         ((1 *))
                         x-%offset%))
         (abs
          (f2cl-lib:fref x-%data%
                         ((f2cl-lib:int-add n j1))
                         ((1 *))
                         x-%offset%)))
        (+
         (abs
          (f2cl-lib:fref x-%data%
                         (j2)
                         ((1 *))
                         x-%offset%))
         (abs
          (f2cl-lib:fref x-%data%
                         ((f2cl-lib:int-add n j2))
                         ((1 *))
                         x-%offset%)))))
 (cond
```

```
  ((> xmax one)
   (setf rec (/ one xmax))
   (cond
     ((>
       (max (f2cl-lib:fref work (j1) ((1 *)))
            (f2cl-lib:fref work (j2) ((1 *))))
       (f2cl-lib:f2cl/ (+ bignum (- xj)) xmax))
      (dscal n2 rec x 1)
      (setf scale (* scale rec))
      (setf xmax (* xmax rec))))))
(setf (f2cl-lib:fref d (1 1) ((1 2) (1 2)))
        (-
         (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
         (ddot (f2cl-lib:int-sub j1 1)
          (f2cl-lib:array-slice t$
                                double-float
                                (1 j1)
                                ((1 ldt) (1 *)))
          1 x 1)))
(setf (f2cl-lib:fref d (2 1) ((1 2) (1 2)))
        (-
         (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%)
         (ddot (f2cl-lib:int-sub j1 1)
          (f2cl-lib:array-slice t$
                                double-float
                                (1 j2)
                                ((1 ldt) (1 *)))
          1 x 1)))
(setf (f2cl-lib:fref d (1 2) ((1 2) (1 2)))
        (-
         (f2cl-lib:fref x-%data%
                        ((f2cl-lib:int-add n j1))
                        ((1 *))
                        x-%offset%)
         (ddot (f2cl-lib:int-sub j1 1)
          (f2cl-lib:array-slice t$
                                double-float
                                (1 j1)
                                ((1 ldt) (1 *)))
          1
          (f2cl-lib:array-slice x
                                double-float
                                ((+ n 1))
                                ((1 *)))
          1)))
(setf (f2cl-lib:fref d (2 2) ((1 2) (1 2)))
```

```
                (-
                 (f2cl-lib:fref x-%data%
                                ((f2cl-lib:int-add n j2))
                                ((1 *))
                                x-%offset%)
                 (ddot (f2cl-lib:int-sub j1 1)
                  (f2cl-lib:array-slice t$
                                        double-float
                                        (1 j2)
                                        ((1 ldt) (1 *)))
                  1
                  (f2cl-lib:array-slice x
                                        double-float
                                        ((+ n 1))
                                        ((1 *)))
                  1)))
          (setf (f2cl-lib:fref d (1 1) ((1 2) (1 2)))
                (- (f2cl-lib:fref d (1 1) ((1 2) (1 2)))
                   (*
                    (f2cl-lib:fref b-%data%
                                   (j1)
                                   ((1 *))
                                   b-%offset%)
                    (f2cl-lib:fref x-%data%
                                   ((f2cl-lib:int-add n 1))
                                   ((1 *))
                                   x-%offset%))))
          (setf (f2cl-lib:fref d (2 1) ((1 2) (1 2)))
                (- (f2cl-lib:fref d (2 1) ((1 2) (1 2)))
                   (*
                    (f2cl-lib:fref b-%data%
                                   (j2)
                                   ((1 *))
                                   b-%offset%)
                    (f2cl-lib:fref x-%data%
                                   ((f2cl-lib:int-add n 1))
                                   ((1 *))
                                   x-%offset%))))
          (setf (f2cl-lib:fref d (1 2) ((1 2) (1 2)))
                (+ (f2cl-lib:fref d (1 2) ((1 2) (1 2)))
                   (*
                    (f2cl-lib:fref b-%data%
                                   (j1)
                                   ((1 *))
                                   b-%offset%)
                    (f2cl-lib:fref x-%data%
```

```
                              (1)
                              ((1 *))
                              x-%offset%))))
(setf (f2cl-lib:fref d (2 2) ((1 2) (1 2)))
       (+ (f2cl-lib:fref d (2 2) ((1 2) (1 2)))
          (*
           (f2cl-lib:fref b-%data%
                              (j2)
                              ((1 *))
                              b-%offset%)
           (f2cl-lib:fref x-%data%
                              (1)
                              ((1 *))
                              x-%offset%))))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13 var-14
       var-15 var-16 var-17)
    (dlaln2 t 2 2 sminw one
     (f2cl-lib:array-slice t$
                              double-float
                              (j1 j1)
                              ((1 ldt) (1 *)))
     ldt one one d 2 zero w v 2 scaloc xnorm ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12 var-13 var-14))
  (setf scaloc var-15)
  (setf xnorm var-16)
  (setf ierr var-17))
(if (/= ierr 0) (setf info 2))
(cond
  ((/= scaloc one)
   (dscal n2 scaloc x 1)
   (setf scale (* scaloc scale))))
(setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
       (f2cl-lib:fref v (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%)
       (f2cl-lib:fref v (2 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref x-%data%
                     ((f2cl-lib:int-add n j1))
                     ((1 *))
                     x-%offset%)
       (f2cl-lib:fref v (1 2) ((1 2) (1 2))))
(setf (f2cl-lib:fref x-%data%
                     ((f2cl-lib:int-add n j2))
```

```
                                        ((1 *))
                                        x-%offset%)
                                (f2cl-lib:fref v (2 2) ((1 2) (1 2))))
                        (setf xmax
                                (max
                                 (+
                                  (abs
                                   (f2cl-lib:fref x-%data%
                                                  (j1)
                                                  ((1 *))
                                                  x-%offset%))
                                  (abs
                                   (f2cl-lib:fref x-%data%
                                                  ((f2cl-lib:int-add n j1))
                                                  ((1 *))
                                                  x-%offset%)))
                                 (+
                                  (abs
                                   (f2cl-lib:fref x-%data%
                                                  (j2)
                                                  ((1 *))
                                                  x-%offset%))
                                  (abs
                                   (f2cl-lib:fref x-%data%
                                                  ((f2cl-lib:int-add n j2))
                                                  ((1 *))
                                                  x-%offset%)))
                                 xmax))))
    label80))))))
    end_label
           (return (values nil nil nil nil nil nil nil scale nil nil info))))))
```

7.46 dlarfb LAPACK

⟨*dlarfb.input*⟩≡

```
)set break resume
)sys rm -f dlarfb.output
)spool dlarfb.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlarfb.help*⟩≡

```
==========================================================================
dlarfb examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

NAME
       DLARFB  - a real block reflector H or its transpose H' to a real m by n
       matrix C, from either the left or the right

SYNOPSIS
       SUBROUTINE DLARFB( SIDE, TRANS, DIRECT, STOREV, M, N,  K,  V,  LDV,  T,
                          LDT, C, LDC, WORK, LDWORK )

           CHARACTER      DIRECT, SIDE, STOREV, TRANS

           INTEGER        K, LDC, LDT, LDV, LDWORK, M, N

           DOUBLE         PRECISION  C(  LDC,  *  ), T( LDT, * ), V( LDV, * ),
                          WORK( LDWORK, * )

PURPOSE
       DLARFB applies a real block reflector H or its transpose H' to a real m
       by n matrix C, from either the left or the right.


ARGUMENTS
       SIDE    (input) CHARACTER*1
               = 'L': apply H or H' from the Left
               = 'R': apply H or H' from the Right

       TRANS   (input) CHARACTER*1
               = 'N': apply H (No transpose)
               = 'T': apply H' (Transpose)

       DIRECT  (input) CHARACTER*1
               Indicates  how H is formed from a product of elementary reflec-
               tors = 'F': H = H(1) H(2) . . . H(k) (Forward)
               = 'B': H = H(k) . . . H(2) H(1) (Backward)

       STOREV  (input) CHARACTER*1
               Indicates how the vectors which define the  elementary  reflec-
               tors are stored:
```

```
          = 'C': Columnwise
          = 'R': Rowwise

M         (input) INTEGER
          The number of rows of the matrix C.

N         (input) INTEGER
          The number of columns of the matrix C.

K         (input) INTEGER
          The  order  of the matrix T (= the number of elementary reflec-
          tors whose product defines the block reflector).

V         (input) DOUBLE PRECISION array, dimension
          (LDV,K) if STOREV = 'C' (LDV,M) if STOREV = 'R' and SIDE =  'L'
          (LDV,N)  if  STOREV = 'R' and SIDE = 'R' The matrix V. See fur-
          ther details.

LDV       (input) INTEGER
          The leading dimension of the array V.  If STOREV = 'C' and SIDE
          =  'L', LDV >= max(1,M); if STOREV = 'C' and SIDE = 'R', LDV >=
          max(1,N); if STOREV = 'R', LDV >= K.

T         (input) DOUBLE PRECISION array, dimension (LDT,K)
          The triangular k by k matrix T in  the  representation  of  the
          block reflector.

LDT       (input) INTEGER
          The leading dimension of the array T. LDT >= K.

C         (input/output) DOUBLE PRECISION array, dimension (LDC,N)
          On  entry,  the  m by n matrix C.  On exit, C is overwritten by
          H*C or H'*C or C*H or C*H'.

LDC       (input) INTEGER
          The leading dimension of the array C. LDA >= max(1,M).

WORK      (workspace) DOUBLE PRECISION array, dimension (LDWORK,K)

LDWORK    (input) INTEGER
          The leading dimension of the array WORK.  If SIDE = 'L', LDWORK
          >= max(1,N); if SIDE = 'R', LDWORK >= max(1,M).
```

⟨*LAPACK dlarfb*⟩≡

```
(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dlarfb (side trans direct storev m n k v ldv t$ ldt c ldc work ldwork)
    (declare (type (simple-array double-float (*)) work c t$ v)
             (type fixnum ldwork ldc ldt ldv k n m)
             (type character storev direct trans side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (trans character trans-%data% trans-%offset%)
         (direct character direct-%data% direct-%offset%)
         (storev character storev-%data% storev-%offset%)
         (v double-float v-%data% v-%offset%)
         (t$ double-float t$-%data% t$-%offset%)
         (c double-float c-%data% c-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((i 0) (j 0)
             (transt
              (make-array '(1) :element-type 'character :initial-element #\ )))
        (declare (type fixnum i j)
                 (type (simple-array character (1)) transt))
        (if (or (<= m 0) (<= n 0)) (go end_label))
        (cond
          ((char-equal trans #\N)
           (f2cl-lib:f2cl-set-string transt "T" (string 1)))
          (t
           (f2cl-lib:f2cl-set-string transt "N" (string 1))))
        (cond
          ((char-equal storev #\C)
           (cond
             ((char-equal direct #\F)
              (cond
                ((char-equal side #\L)
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j k) nil)
                   (tagbody
                     (dcopy n
                      (f2cl-lib:array-slice c
                                            double-float
                                            (j 1)
                                            ((1 ldc) (1 *)))
                      ldc
                      (f2cl-lib:array-slice work
                                            double-float
                                            (1 j)
                                            ((1 ldwork) (1 *)))
```

```
     1)))
(dtrmm "Right" "Lower" "No transpose" "Unit" n k one v ldv
 work ldwork)
(cond
  ((> m k)
   (dgemm "Transpose" "No transpose" n k
    (f2cl-lib:int-sub m k) one
    (f2cl-lib:array-slice c
                          double-float
                          ((+ k 1) 1)
                          ((1 ldc) (1 *)))
    ldc
    (f2cl-lib:array-slice v
                          double-float
                          ((+ k 1) 1)
                          ((1 ldv) (1 *)))
    ldv one work ldwork)))
(dtrmm "Right" "Upper" transt "Non-unit" n k one t$ ldt work
 ldwork)
(cond
  ((> m k)
   (dgemm "No transpose" "Transpose" (f2cl-lib:int-sub m k) n
    k (- one)
    (f2cl-lib:array-slice v
                          double-float
                          ((+ k 1) 1)
                          ((1 ldv) (1 *)))
    ldv work ldwork one
    (f2cl-lib:array-slice c
                          double-float
                          ((+ k 1) 1)
                          ((1 ldc) (1 *)))
    ldc)))
(dtrmm "Right" "Lower" "Transpose" "Unit" n k one v ldv work
 ldwork)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j k) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data%
                             (j i)
                             ((1 ldc) (1 *))
                             c-%offset%)
                (-
```

```
                    (f2cl-lib:fref c-%data%
                                   (j i)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                    (f2cl-lib:fref work-%data%
                                   (i j)
                                   ((1 ldwork) (1 *))
                                   work-%offset%))))))))
((char-equal side #\R)
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j k) nil)
   (tagbody
     (dcopy m
      (f2cl-lib:array-slice c
                            double-float
                            (1 j)
                            ((1 ldc) (1 *)))
      1
      (f2cl-lib:array-slice work
                            double-float
                            (1 j)
                            ((1 ldwork) (1 *)))
      1)))
 (dtrmm "Right" "Lower" "No transpose" "Unit" m k one v ldv
  work ldwork)
 (cond
   ((> n k)
    (dgemm "No transpose" "No transpose" m k
     (f2cl-lib:int-sub n k) one
     (f2cl-lib:array-slice c
                           double-float
                           (1 (f2cl-lib:int-add k 1))
                           ((1 ldc) (1 *)))
     ldc
     (f2cl-lib:array-slice v
                           double-float
                           ((+ k 1) 1)
                           ((1 ldv) (1 *)))
     ldv one work ldwork)))
 (dtrmm "Right" "Upper" trans "Non-unit" m k one t$ ldt work
  ldwork)
 (cond
   ((> n k)
    (dgemm "No transpose" "Transpose" m (f2cl-lib:int-sub n k)
     k (- one) work ldwork
     (f2cl-lib:array-slice v
```

```
                                  double-float
                                  ((+ k 1) 1)
                                  ((1 ldv) (1 *)))
              ldv one
              (f2cl-lib:array-slice c
                                  double-float
                                  (1 (f2cl-lib:int-add k 1))
                                  ((1 ldc) (1 *)))
              ldc)))
          (dtrmm "Right" "Lower" "Transpose" "Unit" m k one v ldv work
           ldwork)
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                        ((> j k) nil)
            (tagbody
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                            ((> i m) nil)
                (tagbody
                  (setf (f2cl-lib:fref c-%data%
                                       (i j)
                                       ((1 ldc) (1 *))
                                       c-%offset%)
                        (-
                         (f2cl-lib:fref c-%data%
                                        (i j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                         (f2cl-lib:fref work-%data%
                                        (i j)
                                        ((1 ldwork) (1 *))
                                        work-%offset%)))))))))
        (t
         (cond
           ((char-equal side #\L)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                          ((> j k) nil)
              (tagbody
                (dcopy n
                 (f2cl-lib:array-slice c
                                       double-float
                                       ((+ m (f2cl-lib:int-sub k) j) 1)
                                       ((1 ldc) (1 *)))
                 ldc
                 (f2cl-lib:array-slice work
                                       double-float
                                       (1 j)
                                       ((1 ldwork) (1 *)))
```

```
     1)))
(dtrmm "Right" "Upper" "No transpose" "Unit" n k one
 (f2cl-lib:array-slice v
                       double-float
                       ((+ m (f2cl-lib:int-sub k) 1) 1)
                       ((1 ldv) (1 *)))
 ldv work ldwork)
(cond
  ((> m k)
   (dgemm "Transpose" "No transpose" n k
    (f2cl-lib:int-sub m k) one c ldc v ldv one work ldwork)))
(dtrmm "Right" "Lower" transt "Non-unit" n k one t$ ldt work
 ldwork)
(cond
  ((> m k)
   (dgemm "No transpose" "Transpose" (f2cl-lib:int-sub m k) n
    k (- one) v ldv work ldwork one c ldc)))
(dtrmm "Right" "Upper" "Transpose" "Unit" n k one
 (f2cl-lib:array-slice v
                       double-float
                       ((+ m (f2cl-lib:int-sub k) 1) 1)
                       ((1 ldv) (1 *)))
 ldv work ldwork)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j k) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data%
                             ((f2cl-lib:int-add
                               (f2cl-lib:int-sub m k)
                               j)
                              i)
                             ((1 ldc) (1 *))
                             c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                ((f2cl-lib:int-add
                                  (f2cl-lib:int-sub m k)
                                  j)
                                 i)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (f2cl-lib:fref work-%data%
                                (i j)
```

```
                                    ((1 ldwork) (1 *))
                                    work-%offset%))))))))
          ((char-equal side #\R)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j k) nil)
             (tagbody
               (dcopy m
                (f2cl-lib:array-slice c
                                      double-float
                                      (1
                                       (f2cl-lib:int-add
                                        (f2cl-lib:int-sub n k)
                                        j))
                                      ((1 ldc) (1 *)))
                1
                (f2cl-lib:array-slice work
                                      double-float
                                      (1 j)
                                      ((1 ldwork) (1 *)))
                1)))
           (dtrmm "Right" "Upper" "No transpose" "Unit" m k one
            (f2cl-lib:array-slice v
                                  double-float
                                  ((+ n (f2cl-lib:int-sub k) 1) 1)
                                  ((1 ldv) (1 *)))
            ldv work ldwork)
           (cond
             ((> n k)
              (dgemm "No transpose" "No transpose" m k
               (f2cl-lib:int-sub n k) one c ldc v ldv one work ldwork)))
           (dtrmm "Right" "Lower" trans "Non-unit" m k one t$ ldt work
            ldwork)
           (cond
             ((> n k)
              (dgemm "No transpose" "Transpose" m (f2cl-lib:int-sub n k)
               k (- one) work ldwork v ldv one c ldc)))
           (dtrmm "Right" "Upper" "Transpose" "Unit" m k one
            (f2cl-lib:array-slice v
                                  double-float
                                  ((+ n (f2cl-lib:int-sub k) 1) 1)
                                  ((1 ldv) (1 *)))
            ldv work ldwork)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j k) nil)
             (tagbody
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
```

```
                                ((> i m) nil)
                              (tagbody
                                (setf (f2cl-lib:fref c-%data%
                                                     (i
                                                      (f2cl-lib:int-add
                                                       (f2cl-lib:int-sub n k)
                                                       j))
                                                     ((1 ldc) (1 *))
                                                     c-%offset%)
                                      (-
                                       (f2cl-lib:fref c-%data%
                                                      (i
                                                       (f2cl-lib:int-add
                                                        (f2cl-lib:int-sub n k)
                                                        j))
                                                      ((1 ldc) (1 *))
                                                      c-%offset%)
                                       (f2cl-lib:fref work-%data%
                                                      (i j)
                                                      ((1 ldwork) (1 *))
                                                      work-%offset%))))))))))))
          ((char-equal storev #\R)
           (cond
             ((char-equal direct #\F)
              (cond
                ((char-equal side #\L)
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                               ((> j k) nil)
                   (tagbody
                     (dcopy n
                      (f2cl-lib:array-slice c
                                            double-float
                                            (j 1)
                                            ((1 ldc) (1 *)))
                      ldc
                      (f2cl-lib:array-slice work
                                            double-float
                                            (1 j)
                                            ((1 ldwork) (1 *)))
                      1)))
                 (dtrmm "Right" "Upper" "Transpose" "Unit" n k one v ldv work
                  ldwork)
                 (cond
                   ((> m k)
                    (dgemm "Transpose" "Transpose" n k (f2cl-lib:int-sub m k)
                     one
```

```
    (f2cl-lib:array-slice c
                          double-float
                          ((+ k 1) 1)
                          ((1 ldc) (1 *)))
    ldc
    (f2cl-lib:array-slice v
                          double-float
                          (1 (f2cl-lib:int-add k 1))
                          ((1 ldv) (1 *)))
    ldv one work ldwork)))
(dtrmm "Right" "Upper" transt "Non-unit" n k one t$ ldt work
 ldwork)
(cond
  ((> m k)
   (dgemm "Transpose" "Transpose" (f2cl-lib:int-sub m k) n k
    (- one)
    (f2cl-lib:array-slice v
                          double-float
                          (1 (f2cl-lib:int-add k 1))
                          ((1 ldv) (1 *)))
    ldv work ldwork one
    (f2cl-lib:array-slice c
                          double-float
                          ((+ k 1) 1)
                          ((1 ldc) (1 *)))
    ldc)))
(dtrmm "Right" "Upper" "No transpose" "Unit" n k one v ldv
 work ldwork)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j k) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data%
                             (j i)
                             ((1 ldc) (1 *))
                             c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j i)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (f2cl-lib:fref work-%data%
                                (i j)
                                ((1 ldwork) (1 *))
```

```
                                        work-%offset%))))))))
          ((char-equal side #\R)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j k) nil)
             (tagbody
               (dcopy m
                (f2cl-lib:array-slice c
                                      double-float
                                      (1 j)
                                      ((1 ldc) (1 *)))
                1
                (f2cl-lib:array-slice work
                                      double-float
                                      (1 j)
                                      ((1 ldwork) (1 *)))
                1)))
           (dtrmm "Right" "Upper" "Transpose" "Unit" m k one v ldv work
            ldwork)
           (cond
             ((> n k)
              (dgemm "No transpose" "Transpose" m k
               (f2cl-lib:int-sub n k) one
               (f2cl-lib:array-slice c
                                     double-float
                                     (1 (f2cl-lib:int-add k 1))
                                     ((1 ldc) (1 *)))
               ldc
               (f2cl-lib:array-slice v
                                     double-float
                                     (1 (f2cl-lib:int-add k 1))
                                     ((1 ldv) (1 *)))
               ldv one work ldwork)))
           (dtrmm "Right" "Upper" trans "Non-unit" m k one t$ ldt work
            ldwork)
           (cond
             ((> n k)
              (dgemm "No transpose" "No transpose" m
               (f2cl-lib:int-sub n k) k (- one) work ldwork
               (f2cl-lib:array-slice v
                                     double-float
                                     (1 (f2cl-lib:int-add k 1))
                                     ((1 ldv) (1 *)))
               ldv one
               (f2cl-lib:array-slice c
                                     double-float
                                     (1 (f2cl-lib:int-add k 1))
```

```
                                  ((1 ldc) (1 *)))
           ldc)))
      (dtrmm "Right" "Upper" "No transpose" "Unit" m k one v ldv
       work ldwork)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j k) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                    (-
                     (f2cl-lib:fref c-%data%
                                    (i j)
                                    ((1 ldc) (1 *))
                                    c-%offset%)
                     (f2cl-lib:fref work-%data%
                                    (i j)
                                    ((1 ldwork) (1 *))
                                    work-%offset%)))))))))
(t
 (cond
   ((char-equal side #\L)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  ((> j k) nil)
      (tagbody
        (dcopy n
         (f2cl-lib:array-slice c
                               double-float
                               ((+ m (f2cl-lib:int-sub k) j) 1)
                               ((1 ldc) (1 *)))
         ldc
         (f2cl-lib:array-slice work
                               double-float
                               (1 j)
                               ((1 ldwork) (1 *)))
         1)))
    (dtrmm "Right" "Lower" "Transpose" "Unit" n k one
     (f2cl-lib:array-slice v
                           double-float
                           (1
                            (f2cl-lib:int-add
                             (f2cl-lib:int-sub m k)
```

```
                        1))
                       ((1 ldv) (1 *)))
 ldv work ldwork)
(cond
  ((> m k)
   (dgemm "Transpose" "Transpose" n k (f2cl-lib:int-sub m k)
    one c ldc v ldv one work ldwork)))
(dtrmm "Right" "Lower" transt "Non-unit" n k one t$ ldt work
 ldwork)
(cond
  ((> m k)
   (dgemm "Transpose" "Transpose" (f2cl-lib:int-sub m k) n k
    (- one) v ldv work ldwork one c ldc)))
(dtrmm "Right" "Lower" "No transpose" "Unit" n k one
 (f2cl-lib:array-slice v
                       double-float
                       (1
                        (f2cl-lib:int-add
                         (f2cl-lib:int-sub m k)
                         1))
                       ((1 ldv) (1 *)))
 ldv work ldwork)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j k) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data%
                             ((f2cl-lib:int-add
                               (f2cl-lib:int-sub m k)
                               j)
                              i)
                             ((1 ldc) (1 *))
                             c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                ((f2cl-lib:int-add
                                  (f2cl-lib:int-sub m k)
                                  j)
                                 i)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (f2cl-lib:fref work-%data%
                                (i j)
                                ((1 ldwork) (1 *))
```

```
                                        work-%offset%))))))))
          ((char-equal side #\R)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j k) nil)
             (tagbody
               (dcopy m
                (f2cl-lib:array-slice c
                                      double-float
                                      (1
                                       (f2cl-lib:int-add
                                        (f2cl-lib:int-sub n k)
                                        j))
                                      ((1 ldc) (1 *)))
                1
                (f2cl-lib:array-slice work
                                      double-float
                                      (1 j)
                                      ((1 ldwork) (1 *)))
                1)))
           (dtrmm "Right" "Lower" "Transpose" "Unit" m k one
            (f2cl-lib:array-slice v
                                  double-float
                                  (1
                                   (f2cl-lib:int-add
                                    (f2cl-lib:int-sub n k)
                                    1))
                                  ((1 ldv) (1 *)))
            ldv work ldwork)
           (cond
             ((> n k)
              (dgemm "No transpose" "Transpose" m k
               (f2cl-lib:int-sub n k) one c ldc v ldv one work ldwork)))
           (dtrmm "Right" "Lower" trans "Non-unit" m k one t$ ldt work
            ldwork)
           (cond
             ((> n k)
              (dgemm "No transpose" "No transpose" m
               (f2cl-lib:int-sub n k) k (- one) work ldwork v ldv one c
               ldc)))
           (dtrmm "Right" "Lower" "No transpose" "Unit" m k one
            (f2cl-lib:array-slice v
                                  double-float
                                  (1
                                   (f2cl-lib:int-add
                                    (f2cl-lib:int-sub n k)
                                    1))
```

```
                                ((1 ldv) (1 *)))
                       ldv work ldwork)
                      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                    ((> j k) nil)
                        (tagbody
                          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                        ((> i m) nil)
                            (tagbody
                              (setf (f2cl-lib:fref c-%data%
                                                   (i
                                                    (f2cl-lib:int-add
                                                     (f2cl-lib:int-sub n k)
                                                     j))
                                                   ((1 ldc) (1 *))
                                                   c-%offset%)
                                    (-
                                     (f2cl-lib:fref c-%data%
                                                    (i
                                                     (f2cl-lib:int-add
                                                      (f2cl-lib:int-sub n k)
                                                      j))
                                                    ((1 ldc) (1 *))
                                                    c-%offset%)
                                     (f2cl-lib:fref work-%data%
                                                    (i j)
                                                    ((1 ldwork) (1 *))
                                                    work-%offset%)))))))))))))
  end_label
         (return
          (values nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil))))))
```

7.47 dlarfg LAPACK

⟨*dlarfg.input*⟩≡

```
)set break resume
)sys rm -f dlarfg.output
)spool dlarfg.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlarfg.help*⟩≡

```
==========================================================================
dlarfg examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

NAME
       DLARFG  -  a  real elementary reflector H of order n, such that   H * (
       alpha ) = ( beta ), H' * H = I

SYNOPSIS
       SUBROUTINE DLARFG( N, ALPHA, X, INCX, TAU )

           INTEGER        INCX, N

           DOUBLE         PRECISION ALPHA, TAU

           DOUBLE         PRECISION X( * )

PURPOSE
       DLARFG generates a real elementary reflector H of order n, such that
                 (   x   )   (   0  )

       where alpha and beta are scalars, and x is an (n-1)-element  real  vec-
       tor. H is represented in the form

             H = I - tau * ( 1 ) * ( 1 v' ) ,
                           ( v )

       where tau is a real scalar and v is a real (n-1)-element
       vector.

       If  the  elements  of x are all zero, then tau = 0 and H is taken to be
       the unit matrix.

       Otherwise  1 <= tau <= 2.


ARGUMENTS
       N       (input) INTEGER
               The order of the elementary reflector.

       ALPHA   (input/output) DOUBLE PRECISION
```

On entry, the value alpha. On exit, it is overwritten with the value beta.

X (input/output) DOUBLE PRECISION array, dimension (1+(N-2)*abs(INCX)) On entry, the vector x. On exit, it is overwritten with the vector v.

INCX (input) INTEGER
The increment between elements of X. INCX > 0.

TAU (output) DOUBLE PRECISION
The value tau.

⟨*LAPACK dlarfg*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dlarfg (n alpha x incx tau)
    (declare (type (simple-array double-float (*)) x)
             (type (double-float) tau alpha)
             (type fixnum incx n))
    (f2cl-lib:with-multi-array-data
        ((x double-float x-%data% x-%offset%))
      (prog ((beta 0.0) (rsafmn 0.0) (safmin 0.0) (xnorm 0.0) (j 0) (knt 0))
        (declare (type (double-float) beta rsafmn safmin xnorm)
                 (type fixnum j knt))
        (cond
          ((<= n 1)
           (setf tau zero)
           (go end_label)))
        (setf xnorm (dnrm2 (f2cl-lib:int-sub n 1) x incx))
        (cond
          ((= xnorm zero)
           (setf tau zero))
          (t
           (setf beta (- (f2cl-lib:sign (dlapy2 alpha xnorm) alpha)))
           (setf safmin (/ (dlamch "S") (dlamch "E")))
           (cond
             ((< (abs beta) safmin)
              (tagbody
                (setf rsafmn (/ one safmin))
                (setf knt 0)
label10
                (setf knt (f2cl-lib:int-add knt 1))
                (dscal (f2cl-lib:int-sub n 1) rsafmn x incx)
                (setf beta (* beta rsafmn))
                (setf alpha (* alpha rsafmn))
                (if (< (abs beta) safmin) (go label10))
                (setf xnorm (dnrm2 (f2cl-lib:int-sub n 1) x incx))
                (setf beta (- (f2cl-lib:sign (dlapy2 alpha xnorm) alpha)))
                (setf tau (/ (- beta alpha) beta))
                (dscal (f2cl-lib:int-sub n 1) (/ one (- alpha beta)) x incx)
                (setf alpha beta)
                (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                              ((> j knt) nil)
                  (tagbody (setf alpha (* alpha safmin)) label20))))
             (t
              (setf tau (/ (- beta alpha) beta))
              (dscal (f2cl-lib:int-sub n 1) (/ one (- alpha beta)) x incx)
```

```
                (setf alpha beta)))))
   end_label
          (return (values nil alpha nil nil tau))))))
```

7.48 dlarf LAPACK

⟨*dlarf.input*⟩≡

```
  )set break resume
  )sys rm -f dlarf.output
  )spool dlarf.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dlarf.help*⟩≡

```
======================================================================
dlarf examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DLARF  -  a real elementary reflector H to a real m by n matrix C, from
       either the left or the right

SYNOPSIS
       SUBROUTINE DLARF( SIDE, M, N, V, INCV, TAU, C, LDC, WORK )

           CHARACTER     SIDE

           INTEGER       INCV, LDC, M, N

           DOUBLE        PRECISION TAU

           DOUBLE        PRECISION C( LDC, * ), V( * ), WORK( * )

PURPOSE
       DLARF applies a real elementary reflector H to a real m by n matrix  C,
       from either the left or the right. H is represented in the form

             H = I - tau * v * v'

       where tau is a real scalar and v is a real vector.

       If tau = 0, then H is taken to be the unit matrix.


ARGUMENTS
       SIDE    (input) CHARACTER*1
               = 'L': form  H * C
               = 'R': form  C * H

       M       (input) INTEGER
               The number of rows of the matrix C.

       N       (input) INTEGER
               The number of columns of the matrix C.
```

```
V       (input) DOUBLE PRECISION array, dimension
        (1 + (M-1)*abs(INCV)) if SIDE = 'L' or (1 + (N-1)*abs(INCV)) if
        SIDE = 'R' The vector v in the representation of H.  V  is  not
        used if TAU = 0.

INCV    (input) INTEGER
        The increment between elements of v. INCV <> 0.

TAU     (input) DOUBLE PRECISION
        The value tau in the representation of H.

C       (input/output) DOUBLE PRECISION array, dimension (LDC,N)
        On  entry,  the  m by n matrix C.  On exit, C is overwritten by
        the matrix H * C if SIDE = 'L', or C * H if SIDE = 'R'.

LDC     (input) INTEGER
        The leading dimension of the array C. LDC >= max(1,M).

WORK    (workspace) DOUBLE PRECISION array, dimension
        (N) if SIDE = 'L' or (M) if SIDE = 'R'
```

⟨*LAPACK dlarf*⟩≡
```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dlarf (side m n v incv tau c ldc work)
    (declare (type (double-float) tau)
             (type (simple-array double-float (*)) work c v)
             (type fixnum ldc incv n m)
             (type character side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (v double-float v-%data% v-%offset%)
         (c double-float c-%data% c-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ()
        (declare)
        (cond
          ((char-equal side #\L)
           (cond
             ((/= tau zero)
              (dgemv "Transpose" m n one c ldc v incv zero work 1)
              (dger m n (- tau) v incv work 1 c ldc))))
          (t
           (cond
             ((/= tau zero)
              (dgemv "No transpose" m n one c ldc v incv zero work 1)
              (dger m n (- tau) work 1 v incv c ldc)))))
        (return (values nil nil nil nil nil nil nil nil nil))))))
```

7.49 dlarft LAPACK

⟨*dlarft.input*⟩≡
```
)set break resume
)sys rm -f dlarft.output
)spool dlarft.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlarft.help*⟩≡

```
  ==========================================================================
  dlarft examples
  ==========================================================================

  ==========================================================================
  Man Page Details
  ==========================================================================

  NAME
         DLARFT  -  the triangular factor T of a real block reflector H of order
         n, which is defined as a product of k elementary reflectors

  SYNOPSIS
         SUBROUTINE DLARFT( DIRECT, STOREV, N, K, V, LDV, TAU, T, LDT )

             CHARACTER      DIRECT, STOREV

             INTEGER        K, LDT, LDV, N

             DOUBLE         PRECISION T( LDT, * ), TAU( * ), V( LDV, * )

  PURPOSE
         DLARFT forms the triangular factor T of a real  block  reflector  H  of
         order n, which is defined as a product of k elementary reflectors.

         If DIRECT = 'F', H = H(1) H(2) . . . H(k) and T is upper triangular;

         If DIRECT = 'B', H = H(k) . . . H(2) H(1) and T is lower triangular.

         If STOREV = 'C', the vector which defines the elementary reflector H(i)
         is stored in the i-th column of the array V, and

            H  =  I - V * T * V'

         If STOREV = 'R', the vector which defines the elementary reflector H(i)
         is stored in the i-th row of the array V, and

            H  =  I - V' * T * V


  ARGUMENTS
         DIRECT  (input) CHARACTER*1
                 Specifies the order in which the elementary reflectors are mul-
                 tiplied to form the block reflector:
                 = 'F': H = H(1) H(2) . . . H(k) (Forward)
```

```
          = 'B': H = H(k) . . . H(2) H(1) (Backward)

      STOREV  (input) CHARACTER*1
              Specifies how the vectors which define the  elementary  reflec-
              tors are stored (see also Further Details):
              = 'R': rowwise

      N       (input) INTEGER
              The order of the block reflector H. N >= 0.

      K       (input) INTEGER
              The  order  of the triangular factor T (= the number of elemen-
              tary reflectors). K >= 1.

      V       (input/output) DOUBLE PRECISION array, dimension
              (LDV,K) if STOREV = 'C' (LDV,N) if STOREV = 'R' The  matrix  V.
              See further details.

      LDV     (input) INTEGER
              The  leading dimension of the array V.  If STOREV = 'C', LDV >=
              max(1,N); if STOREV = 'R', LDV >= K.

      TAU     (input) DOUBLE PRECISION array, dimension (K)
              TAU(i) must contain the scalar factor of the elementary reflec-
              tor H(i).

      T       (output) DOUBLE PRECISION array, dimension (LDT,K)
              The  k  by  k  triangular  factor T of the block reflector.  If
              DIRECT = 'F', T is upper triangular; if  DIRECT  =  'B',  T  is
              lower triangular. The rest of the array is not used.

      LDT     (input) INTEGER
              The leading dimension of the array T. LDT >= K.

  FURTHER DETAILS
      The  shape  of the matrix V and the storage of the vectors which define
      the H(i) is best illustrated by the following example with n = 5 and  k
      =  3.  The  elements equal to 1 are not stored; the corresponding array
      elements are modified but restored on exit. The rest of  the  array  is
      not used.

      DIRECT = 'F' and STOREV = 'C':         DIRECT = 'F' and STOREV = 'R':

                   V = (  1       )                 V = (  1 v1 v1 v1 v1 )
                       ( v1  1    )                     (     1 v2 v2 v2 )
                       ( v1 v2  1 )                     (        1 v3 v3 )
```

```
                    ( v1 v2 v3 )
                    ( v1 v2 v3 )

DIRECT = 'B' and STOREV = 'C':          DIRECT = 'B' and STOREV = 'R':

             V = ( v1 v2 v3 )                        V = ( v1 v1  1       )
                 ( v1 v2 v3 )                            ( v2 v2 v2  1    )
                 (  1 v2 v3 )                            ( v3 v3 v3 v3  1 )
                 (     1 v3 )
                 (        1 )
```

⟨*LAPACK dlarft*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dlarft (direct storev n k v ldv tau t$ ldt)
    (declare (type (simple-array double-float (*)) t$ tau v)
             (type fixnum ldt ldv k n)
             (type character storev direct))
    (f2cl-lib:with-multi-array-data
        ((direct character direct-%data% direct-%offset%)
         (storev character storev-%data% storev-%offset%)
         (v double-float v-%data% v-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (t$ double-float t$-%data% t$-%offset%))
      (prog ((vii 0.0) (i 0) (j 0))
        (declare (type (double-float) vii) (type fixnum i j))
        (if (= n 0) (go end_label))
        (cond
          ((char-equal direct #\F)
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                         ((> i k) nil)
             (tagbody
               (cond
                 ((= (f2cl-lib:fref tau (i) ((1 *))) zero)
                  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                ((> j i) nil)
                    (tagbody
                      (setf (f2cl-lib:fref t$-%data%
                                           (j i)
                                           ((1 ldt) (1 *))
                                           t$-%offset%)
                                zero))))
                 (t
                  (setf vii
                        (f2cl-lib:fref v-%data%
                                       (i i)
                                       ((1 ldv) (1 *))
                                       v-%offset%))
                  (setf (f2cl-lib:fref v-%data%
                                       (i i)
                                       ((1 ldv) (1 *))
                                       v-%offset%)
                        one)
                  (cond
                    ((char-equal storev #\C)
                     (dgemv "Transpose"
```

```
                (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
                (f2cl-lib:int-sub i 1)
                (- (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
                (f2cl-lib:array-slice v
                                      double-float
                                      (i 1)
                                      ((1 ldv) (1 *)))
                ldv
                (f2cl-lib:array-slice v
                                      double-float
                                      (i i)
                                      ((1 ldv) (1 *)))
                1 zero
                (f2cl-lib:array-slice t$
                                      double-float
                                      (1 i)
                                      ((1 ldt) (1 *)))
                1))
              (t
               (dgemv "No transpose" (f2cl-lib:int-sub i 1)
                (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
                (- (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
                (f2cl-lib:array-slice v
                                      double-float
                                      (1 i)
                                      ((1 ldv) (1 *)))
                ldv
                (f2cl-lib:array-slice v
                                      double-float
                                      (i i)
                                      ((1 ldv) (1 *)))
                ldv zero
                (f2cl-lib:array-slice t$
                                      double-float
                                      (1 i)
                                      ((1 ldt) (1 *)))
                1)))
            (setf (f2cl-lib:fref v-%data%
                                 (i i)
                                 ((1 ldv) (1 *))
                                 v-%offset%)
                  vii)
            (dtrmv "Upper" "No transpose" "Non-unit"
             (f2cl-lib:int-sub i 1) t$ ldt
             (f2cl-lib:array-slice t$ double-float (1 i) ((1 ldt) (1 *)))
             1)
```

```
          (setf (f2cl-lib:fref t$-%data%
                               (i i)
                               ((1 ldt) (1 *))
                               t$-%offset%)
                (f2cl-lib:fref tau-%data%
                               (i)
                               ((1 *))
                               tau-%offset%)))))))
    (t
     (f2cl-lib:fdo (i k (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                   ((> i 1) nil)
       (tagbody
         (cond
           ((= (f2cl-lib:fref tau (i) ((1 *))) zero)
            (f2cl-lib:fdo (j i (f2cl-lib:int-add j 1))
                          ((> j k) nil)
              (tagbody
                (setf (f2cl-lib:fref t$-%data%
                                     (j i)
                                     ((1 ldt) (1 *))
                                     t$-%offset%)
                      zero))))
           (t
            (cond
              ((< i k)
               (cond
                 ((char-equal storev #\C)
                  (setf vii
                        (f2cl-lib:fref v-%data%
                                       ((f2cl-lib:int-add
                                         (f2cl-lib:int-sub n k)
                                         i)
                                        i)
                                       ((1 ldv) (1 *))
                                       v-%offset%))
                  (setf (f2cl-lib:fref v-%data%
                                       ((f2cl-lib:int-add
                                         (f2cl-lib:int-sub n k)
                                         i)
                                        i)
                                       ((1 ldv) (1 *))
                                       v-%offset%)
                        one)
                  (dgemv "Transpose"
                   (f2cl-lib:int-add (f2cl-lib:int-sub n k) i)
                   (f2cl-lib:int-sub k i)
```

```
 (-
  (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
 (f2cl-lib:array-slice v
                       double-float
                       (1 (f2cl-lib:int-add i 1))
                       ((1 ldv) (1 *)))
 ldv
 (f2cl-lib:array-slice v
                       double-float
                       (1 i)
                       ((1 ldv) (1 *)))
 1 zero
 (f2cl-lib:array-slice t$
                       double-float
                       ((+ i 1) i)
                       ((1 ldt) (1 *)))
 1)
(setf (f2cl-lib:fref v-%data%
                     ((f2cl-lib:int-add
                       (f2cl-lib:int-sub n k)
                       i)
                      i)
                     ((1 ldv) (1 *))
                     v-%offset%)
       vii))
(t
 (setf vii
       (f2cl-lib:fref v-%data%
                      (i
                       (f2cl-lib:int-add
                        (f2cl-lib:int-sub n k)
                        i))
                      ((1 ldv) (1 *))
                      v-%offset%))
 (setf (f2cl-lib:fref v-%data%
                      (i
                       (f2cl-lib:int-add
                        (f2cl-lib:int-sub n k)
                        i))
                      ((1 ldv) (1 *))
                      v-%offset%)
       one)
 (dgemv "No transpose" (f2cl-lib:int-sub k i)
  (f2cl-lib:int-add (f2cl-lib:int-sub n k) i)
  (-
   (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
```

```
                        (f2cl-lib:array-slice v
                                              double-float
                                              ((+ i 1) 1)
                                              ((1 ldv) (1 *)))
                        ldv
                        (f2cl-lib:array-slice v
                                              double-float
                                              (i 1)
                                              ((1 ldv) (1 *)))
                        ldv zero
                        (f2cl-lib:array-slice t$
                                              double-float
                                              ((+ i 1) i)
                                              ((1 ldt) (1 *)))
                        1)
                       (setf (f2cl-lib:fref v-%data%
                                            (i
                                             (f2cl-lib:int-add
                                              (f2cl-lib:int-sub n k)
                                              i))
                                            ((1 ldv) (1 *))
                                            v-%offset%)
                             vii)))
                   (dtrmv "Lower" "No transpose" "Non-unit"
                    (f2cl-lib:int-sub k i)
                    (f2cl-lib:array-slice t$
                                          double-float
                                          ((+ i 1) (f2cl-lib:int-add i 1))
                                          ((1 ldt) (1 *)))
                    ldt
                    (f2cl-lib:array-slice t$
                                          double-float
                                          ((+ i 1) i)
                                          ((1 ldt) (1 *)))
                    1)))
                (setf (f2cl-lib:fref t$-%data%
                                     (i i)
                                     ((1 ldt) (1 *))
                                     t$-%offset%)
                      (f2cl-lib:fref tau-%data%
                                     (i)
                                     ((1 *))
                                     tau-%offset%))))))))
  end_label
       (return (values nil nil nil nil nil nil nil nil nil))))))
```

7.50 dlarfx LAPACK

⟨*dlarfx.input*⟩≡

```
)set break resume
)sys rm -f dlarfx.output
)spool dlarfx.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlarfx.help*⟩≡

```
=====================================================================
dlarfx examples
=====================================================================

=====================================================================
Man Page Details
=====================================================================

NAME
       DLARFX  - a real elementary reflector H to a real m by n matrix C, from
       either the left or the right

SYNOPSIS
       SUBROUTINE DLARFX( SIDE, M, N, V, TAU, C, LDC, WORK )

           CHARACTER      SIDE

           INTEGER        LDC, M, N

           DOUBLE         PRECISION TAU

           DOUBLE         PRECISION C( LDC, * ), V( * ), WORK( * )

PURPOSE
       DLARFX applies a real elementary reflector H to a real m by n matrix C,
       from either the left or the right. H is represented in the form

             H = I - tau * v * v'

       where tau is a real scalar and v is a real vector.

       If tau = 0, then H is taken to be the unit matrix

       This version uses inline code if H has order < 11.


ARGUMENTS
       SIDE    (input) CHARACTER*1
               = 'L': form  H * C
               = 'R': form  C * H

       M       (input) INTEGER
               The number of rows of the matrix C.

       N       (input) INTEGER
```

```
          The number of columns of the matrix C.

V         (input) DOUBLE PRECISION array, dimension (M) if SIDE = 'L'
          or (N) if SIDE = 'R' The vector v in the representation of H.

TAU       (input) DOUBLE PRECISION
          The value tau in the representation of H.

C         (input/output) DOUBLE PRECISION array, dimension (LDC,N)
          On  entry,  the  m by n matrix C.  On exit, C is overwritten by
          the matrix H * C if SIDE = 'L', or C * H if SIDE = 'R'.

LDC       (input) INTEGER
          The leading dimension of the array C. LDA >= (1,M).

WORK      (workspace) DOUBLE PRECISION array, dimension
          (N) if SIDE = 'L' or (M) if SIDE = 'R' WORK is  not  referenced
          if H has order < 11.
```

⟨*LAPACK dlarfx*⟩≡

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dlarfx (side m n v tau c ldc work)
    (declare (type (double-float) tau)
             (type (simple-array double-float (*)) work c v)
             (type fixnum ldc n m)
             (type character side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (v double-float v-%data% v-%offset%)
         (c double-float c-%data% c-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((sum 0.0) (t1 0.0) (t10 0.0) (t2 0.0) (t3 0.0) (t4 0.0) (t5 0.0)
             (t6 0.0) (t7 0.0) (t8 0.0) (t9 0.0) (v1 0.0) (v10 0.0) (v2 0.0)
             (v3 0.0) (v4 0.0) (v5 0.0) (v6 0.0) (v7 0.0) (v8 0.0) (v9 0.0)
             (j 0))
        (declare (type (double-float) sum t1 t10 t2 t3 t4 t5 t6 t7 t8 t9 v1 v10
                                      v2 v3 v4 v5 v6 v7 v8 v9)
                 (type fixnum j))
        (if (= tau zero) (go end_label))
        (cond
          ((char-equal side #\L)
           (tagbody
             (f2cl-lib:computed-goto
              (label10 label30 label50 label70 label90 label110 label130
               label150 label170 label190)
              m)
             (dgemv "Transpose" m n one c ldc v 1 zero work 1)
             (dger m n (- tau) v 1 work 1 c ldc)
             (go end_label)
 label10
             (setf t1
                     (+ one
                        (* (- tau)
                           (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%)
                           (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))))
             (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                           ((> j n) nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                      (1 j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       (* t1
```

```
                        (f2cl-lib:fref c-%data%
                                      (1 j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)))))
              (go end_label)
  label30
              (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
              (setf t1 (* tau v1))
              (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
              (setf t2 (* tau v2))
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (setf sum
                          (+
                           (* v1
                              (f2cl-lib:fref c-%data%
                                            (1 j)
                                            ((1 ldc) (1 *))
                                            c-%offset%))
                           (* v2
                              (f2cl-lib:fref c-%data%
                                            (2 j)
                                            ((1 ldc) (1 *))
                                            c-%offset%))))
                  (setf (f2cl-lib:fref c-%data%
                                       (1 j)
                                       ((1 ldc) (1 *))
                                       c-%offset%)
                          (-
                           (f2cl-lib:fref c-%data%
                                          (1 j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                           (* sum t1)))
                  (setf (f2cl-lib:fref c-%data%
                                       (2 j)
                                       ((1 ldc) (1 *))
                                       c-%offset%)
                          (-
                           (f2cl-lib:fref c-%data%
                                          (2 j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                           (* sum t2)))))
              (go end_label)
```

```
label50
            (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
            (setf t1 (* tau v1))
            (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
            (setf t2 (* tau v2))
            (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
            (setf t3 (* tau v3))
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                          ((> j n) nil)
              (tagbody
                (setf sum
                        (+
                         (* v1
                            (f2cl-lib:fref c-%data%
                                           (1 j)
                                           ((1 ldc) (1 *))
                                           c-%offset%))
                         (* v2
                            (f2cl-lib:fref c-%data%
                                           (2 j)
                                           ((1 ldc) (1 *))
                                           c-%offset%))
                         (* v3
                            (f2cl-lib:fref c-%data%
                                           (3 j)
                                           ((1 ldc) (1 *))
                                           c-%offset%))))
                (setf (f2cl-lib:fref c-%data%
                                     (1 j)
                                     ((1 ldc) (1 *))
                                     c-%offset%)
                        (-
                         (f2cl-lib:fref c-%data%
                                        (1 j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                         (* sum t1)))
                (setf (f2cl-lib:fref c-%data%
                                     (2 j)
                                     ((1 ldc) (1 *))
                                     c-%offset%)
                        (-
                         (f2cl-lib:fref c-%data%
                                        (2 j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
```

```
                        (* sum t2)))
                    (setf (f2cl-lib:fref c-%data%
                                         (3 j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                          (-
                           (f2cl-lib:fref c-%data%
                                          (3 j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                           (* sum t3)))))
              (go end_label)
  label70
              (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
              (setf t1 (* tau v1))
              (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
              (setf t2 (* tau v2))
              (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
              (setf t3 (* tau v3))
              (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
              (setf t4 (* tau v4))
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (setf sum
                          (+
                           (* v1
                              (f2cl-lib:fref c-%data%
                                             (1 j)
                                             ((1 ldc) (1 *))
                                             c-%offset%))
                           (* v2
                              (f2cl-lib:fref c-%data%
                                             (2 j)
                                             ((1 ldc) (1 *))
                                             c-%offset%))
                           (* v3
                              (f2cl-lib:fref c-%data%
                                             (3 j)
                                             ((1 ldc) (1 *))
                                             c-%offset%))
                           (* v4
                              (f2cl-lib:fref c-%data%
                                             (4 j)
                                             ((1 ldc) (1 *))
                                             c-%offset%))))
```

```
                    (setf (f2cl-lib:fref c-%data%
                                         (1 j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                            (-
                             (f2cl-lib:fref c-%data%
                                            (1 j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)
                             (* sum t1)))
                    (setf (f2cl-lib:fref c-%data%
                                         (2 j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                            (-
                             (f2cl-lib:fref c-%data%
                                            (2 j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)
                             (* sum t2)))
                    (setf (f2cl-lib:fref c-%data%
                                         (3 j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                            (-
                             (f2cl-lib:fref c-%data%
                                            (3 j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)
                             (* sum t3)))
                    (setf (f2cl-lib:fref c-%data%
                                         (4 j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                            (-
                             (f2cl-lib:fref c-%data%
                                            (4 j)
                                            ((1 ldc) (1 *))
                                            c-%offset%)
                             (* sum t4)))))
                (go end_label)
  label90
                (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
                (setf t1 (* tau v1))
                (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
                (setf t2 (* tau v2))
```

```
(setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
(setf t3 (* tau v3))
(setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
(setf t4 (* tau v4))
(setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
(setf t5 (* tau v5))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j n) nil)
  (tagbody
    (setf sum
            (+
             (* v1
                (f2cl-lib:fref c-%data%
                               (1 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v2
                (f2cl-lib:fref c-%data%
                               (2 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v3
                (f2cl-lib:fref c-%data%
                               (3 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v4
                (f2cl-lib:fref c-%data%
                               (4 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v5
                (f2cl-lib:fref c-%data%
                               (5 j)
                               ((1 ldc) (1 *))
                               c-%offset%))))
    (setf (f2cl-lib:fref c-%data%
                         (1 j)
                         ((1 ldc) (1 *))
                         c-%offset%)
            (-
             (f2cl-lib:fref c-%data%
                            (1 j)
                            ((1 ldc) (1 *))
                            c-%offset%)
             (* sum t1)))
```

```
                    (setf (f2cl-lib:fref c-%data%
                                         (2 j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                            (-
                             (f2cl-lib:fref c-%data%
                                           (2 j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                             (* sum t2)))
                    (setf (f2cl-lib:fref c-%data%
                                         (3 j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                            (-
                             (f2cl-lib:fref c-%data%
                                           (3 j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                             (* sum t3)))
                    (setf (f2cl-lib:fref c-%data%
                                         (4 j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                            (-
                             (f2cl-lib:fref c-%data%
                                           (4 j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                             (* sum t4)))
                    (setf (f2cl-lib:fref c-%data%
                                         (5 j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                            (-
                             (f2cl-lib:fref c-%data%
                                           (5 j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                             (* sum t5)))))
                (go end_label)
  label110
                (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
                (setf t1 (* tau v1))
                (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
                (setf t2 (* tau v2))
```

```
(setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
(setf t3 (* tau v3))
(setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
(setf t4 (* tau v4))
(setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
(setf t5 (* tau v5))
(setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
(setf t6 (* tau v6))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j n) nil)
  (tagbody
    (setf sum
            (+
             (* v1
                (f2cl-lib:fref c-%data%
                               (1 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v2
                (f2cl-lib:fref c-%data%
                               (2 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v3
                (f2cl-lib:fref c-%data%
                               (3 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v4
                (f2cl-lib:fref c-%data%
                               (4 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v5
                (f2cl-lib:fref c-%data%
                               (5 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v6
                (f2cl-lib:fref c-%data%
                               (6 j)
                               ((1 ldc) (1 *))
                               c-%offset%))))
    (setf (f2cl-lib:fref c-%data%
                         (1 j)
                         ((1 ldc) (1 *))
```

```
                        c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (1 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t1)))
(setf (f2cl-lib:fref c-%data%
                     (2 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (2 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t2)))
(setf (f2cl-lib:fref c-%data%
                     (3 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (3 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t3)))
(setf (f2cl-lib:fref c-%data%
                     (4 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (4 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t4)))
(setf (f2cl-lib:fref c-%data%
                     (5 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (5 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
```

```
                        (* sum t5)))
                    (setf (f2cl-lib:fref c-%data%
                                         (6 j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                          (-
                           (f2cl-lib:fref c-%data%
                                          (6 j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                           (* sum t6)))))
                (go end_label)
  label130
                (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
                (setf t1 (* tau v1))
                (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
                (setf t2 (* tau v2))
                (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
                (setf t3 (* tau v3))
                (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
                (setf t4 (* tau v4))
                (setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
                (setf t5 (* tau v5))
                (setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
                (setf t6 (* tau v6))
                (setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
                (setf t7 (* tau v7))
                (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                              ((> j n) nil)
                  (tagbody
                    (setf sum
                          (+
                           (* v1
                              (f2cl-lib:fref c-%data%
                                             (1 j)
                                             ((1 ldc) (1 *))
                                             c-%offset%))
                           (* v2
                              (f2cl-lib:fref c-%data%
                                             (2 j)
                                             ((1 ldc) (1 *))
                                             c-%offset%))
                           (* v3
                              (f2cl-lib:fref c-%data%
                                             (3 j)
                                             ((1 ldc) (1 *))
```

```
                                c-%offset%))
                  (* v4
                     (f2cl-lib:fref c-%data%
                                    (4 j)
                                    ((1 ldc) (1 *))
                                    c-%offset%))
                  (* v5
                     (f2cl-lib:fref c-%data%
                                    (5 j)
                                    ((1 ldc) (1 *))
                                    c-%offset%))
                  (* v6
                     (f2cl-lib:fref c-%data%
                                    (6 j)
                                    ((1 ldc) (1 *))
                                    c-%offset%))
                  (* v7
                     (f2cl-lib:fref c-%data%
                                    (7 j)
                                    ((1 ldc) (1 *))
                                    c-%offset%))))
            (setf (f2cl-lib:fref c-%data%
                                 (1 j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (-
                   (f2cl-lib:fref c-%data%
                                  (1 j)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                   (* sum t1)))
            (setf (f2cl-lib:fref c-%data%
                                 (2 j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (-
                   (f2cl-lib:fref c-%data%
                                  (2 j)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                   (* sum t2)))
            (setf (f2cl-lib:fref c-%data%
                                 (3 j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (-
```

```
                (f2cl-lib:fref c-%data%
                               (3 j)
                               ((1 ldc) (1 *))
                               c-%offset%)
                (* sum t3)))
           (setf (f2cl-lib:fref c-%data%
                                (4 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (-
                  (f2cl-lib:fref c-%data%
                                 (4 j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (* sum t4)))
           (setf (f2cl-lib:fref c-%data%
                                (5 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (-
                  (f2cl-lib:fref c-%data%
                                 (5 j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (* sum t5)))
           (setf (f2cl-lib:fref c-%data%
                                (6 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (-
                  (f2cl-lib:fref c-%data%
                                 (6 j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (* sum t6)))
           (setf (f2cl-lib:fref c-%data%
                                (7 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (-
                  (f2cl-lib:fref c-%data%
                                 (7 j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (* sum t7)))))
      (go end_label)
```

```
label150
              (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
              (setf t1 (* tau v1))
              (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
              (setf t2 (* tau v2))
              (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
              (setf t3 (* tau v3))
              (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
              (setf t4 (* tau v4))
              (setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
              (setf t5 (* tau v5))
              (setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
              (setf t6 (* tau v6))
              (setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
              (setf t7 (* tau v7))
              (setf v8 (f2cl-lib:fref v-%data% (8) ((1 *)) v-%offset%))
              (setf t8 (* tau v8))
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (setf sum
                          (+
                           (* v1
                              (f2cl-lib:fref c-%data%
                                             (1 j)
                                             ((1 ldc) (1 *))
                                             c-%offset%))
                           (* v2
                              (f2cl-lib:fref c-%data%
                                             (2 j)
                                             ((1 ldc) (1 *))
                                             c-%offset%))
                           (* v3
                              (f2cl-lib:fref c-%data%
                                             (3 j)
                                             ((1 ldc) (1 *))
                                             c-%offset%))
                           (* v4
                              (f2cl-lib:fref c-%data%
                                             (4 j)
                                             ((1 ldc) (1 *))
                                             c-%offset%))
                           (* v5
                              (f2cl-lib:fref c-%data%
                                             (5 j)
                                             ((1 ldc) (1 *))
```

```
                                c-%offset%))
                  (* v6
                     (f2cl-lib:fref c-%data%
                                    (6 j)
                                    ((1 ldc) (1 *))
                                    c-%offset%))
                  (* v7
                     (f2cl-lib:fref c-%data%
                                    (7 j)
                                    ((1 ldc) (1 *))
                                    c-%offset%))
                  (* v8
                     (f2cl-lib:fref c-%data%
                                    (8 j)
                                    ((1 ldc) (1 *))
                                    c-%offset%))))
           (setf (f2cl-lib:fref c-%data%
                                (1 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (-
                  (f2cl-lib:fref c-%data%
                                 (1 j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (* sum t1)))
           (setf (f2cl-lib:fref c-%data%
                                (2 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (-
                  (f2cl-lib:fref c-%data%
                                 (2 j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (* sum t2)))
           (setf (f2cl-lib:fref c-%data%
                                (3 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (-
                  (f2cl-lib:fref c-%data%
                                 (3 j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (* sum t3)))
```

```
(setf (f2cl-lib:fref c-%data%
                     (4 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
        (-
         (f2cl-lib:fref c-%data%
                        (4 j)
                        ((1 ldc) (1 *))
                        c-%offset%)
         (* sum t4)))
(setf (f2cl-lib:fref c-%data%
                     (5 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
        (-
         (f2cl-lib:fref c-%data%
                        (5 j)
                        ((1 ldc) (1 *))
                        c-%offset%)
         (* sum t5)))
(setf (f2cl-lib:fref c-%data%
                     (6 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
        (-
         (f2cl-lib:fref c-%data%
                        (6 j)
                        ((1 ldc) (1 *))
                        c-%offset%)
         (* sum t6)))
(setf (f2cl-lib:fref c-%data%
                     (7 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
        (-
         (f2cl-lib:fref c-%data%
                        (7 j)
                        ((1 ldc) (1 *))
                        c-%offset%)
         (* sum t7)))
(setf (f2cl-lib:fref c-%data%
                     (8 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
        (-
         (f2cl-lib:fref c-%data%
```

```
                                        (8 j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                          (* sum t8)))))
                (go end_label)
  label170
                (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
                (setf t1 (* tau v1))
                (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
                (setf t2 (* tau v2))
                (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
                (setf t3 (* tau v3))
                (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
                (setf t4 (* tau v4))
                (setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
                (setf t5 (* tau v5))
                (setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
                (setf t6 (* tau v6))
                (setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
                (setf t7 (* tau v7))
                (setf v8 (f2cl-lib:fref v-%data% (8) ((1 *)) v-%offset%))
                (setf t8 (* tau v8))
                (setf v9 (f2cl-lib:fref v-%data% (9) ((1 *)) v-%offset%))
                (setf t9 (* tau v9))
                (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                              ((> j n) nil)
                  (tagbody
                    (setf sum
                            (+
                             (* v1
                                (f2cl-lib:fref c-%data%
                                               (1 j)
                                               ((1 ldc) (1 *))
                                               c-%offset%))
                             (* v2
                                (f2cl-lib:fref c-%data%
                                               (2 j)
                                               ((1 ldc) (1 *))
                                               c-%offset%))
                             (* v3
                                (f2cl-lib:fref c-%data%
                                               (3 j)
                                               ((1 ldc) (1 *))
                                               c-%offset%))
                             (* v4
                                (f2cl-lib:fref c-%data%
```

```
                                (4 j)
                                ((1 ldc) (1 *))
                                c-%offset%))
                        (* v5
                           (f2cl-lib:fref c-%data%
                                (5 j)
                                ((1 ldc) (1 *))
                                c-%offset%))
                        (* v6
                           (f2cl-lib:fref c-%data%
                                (6 j)
                                ((1 ldc) (1 *))
                                c-%offset%))
                        (* v7
                           (f2cl-lib:fref c-%data%
                                (7 j)
                                ((1 ldc) (1 *))
                                c-%offset%))
                        (* v8
                           (f2cl-lib:fref c-%data%
                                (8 j)
                                ((1 ldc) (1 *))
                                c-%offset%))
                        (* v9
                           (f2cl-lib:fref c-%data%
                                (9 j)
                                ((1 ldc) (1 *))
                                c-%offset%))))
                (setf (f2cl-lib:fref c-%data%
                                     (1 j)
                                     ((1 ldc) (1 *))
                                     c-%offset%)
                      (-
                       (f2cl-lib:fref c-%data%
                                      (1 j)
                                      ((1 ldc) (1 *))
                                      c-%offset%)
                       (* sum t1)))
                (setf (f2cl-lib:fref c-%data%
                                     (2 j)
                                     ((1 ldc) (1 *))
                                     c-%offset%)
                      (-
                       (f2cl-lib:fref c-%data%
                                      (2 j)
                                      ((1 ldc) (1 *))
```

```
                          c-%offset%)
            (* sum t2)))
(setf (f2cl-lib:fref c-%data%
                     (3 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
      (-
       (f2cl-lib:fref c-%data%
                      (3 j)
                      ((1 ldc) (1 *))
                      c-%offset%)
       (* sum t3)))
(setf (f2cl-lib:fref c-%data%
                     (4 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
      (-
       (f2cl-lib:fref c-%data%
                      (4 j)
                      ((1 ldc) (1 *))
                      c-%offset%)
       (* sum t4)))
(setf (f2cl-lib:fref c-%data%
                     (5 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
      (-
       (f2cl-lib:fref c-%data%
                      (5 j)
                      ((1 ldc) (1 *))
                      c-%offset%)
       (* sum t5)))
(setf (f2cl-lib:fref c-%data%
                     (6 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
      (-
       (f2cl-lib:fref c-%data%
                      (6 j)
                      ((1 ldc) (1 *))
                      c-%offset%)
       (* sum t6)))
(setf (f2cl-lib:fref c-%data%
                     (7 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
```

```
                        (-
                         (f2cl-lib:fref c-%data%
                                        (7 j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                         (* sum t7)))
                    (setf (f2cl-lib:fref c-%data%
                                         (8 j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                        (-
                         (f2cl-lib:fref c-%data%
                                        (8 j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                         (* sum t8)))
                    (setf (f2cl-lib:fref c-%data%
                                         (9 j)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                        (-
                         (f2cl-lib:fref c-%data%
                                        (9 j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                         (* sum t9)))))
                (go end_label)
label190
                (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
                (setf t1 (* tau v1))
                (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
                (setf t2 (* tau v2))
                (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
                (setf t3 (* tau v3))
                (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
                (setf t4 (* tau v4))
                (setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
                (setf t5 (* tau v5))
                (setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
                (setf t6 (* tau v6))
                (setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
                (setf t7 (* tau v7))
                (setf v8 (f2cl-lib:fref v-%data% (8) ((1 *)) v-%offset%))
                (setf t8 (* tau v8))
                (setf v9 (f2cl-lib:fref v-%data% (9) ((1 *)) v-%offset%))
                (setf t9 (* tau v9))
```

```
(setf v10 (f2cl-lib:fref v-%data% (10) ((1 *)) v-%offset%))
(setf t10 (* tau v10))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j n) nil)
  (tagbody
    (setf sum
            (+
             (* v1
                (f2cl-lib:fref c-%data%
                               (1 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v2
                (f2cl-lib:fref c-%data%
                               (2 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v3
                (f2cl-lib:fref c-%data%
                               (3 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v4
                (f2cl-lib:fref c-%data%
                               (4 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v5
                (f2cl-lib:fref c-%data%
                               (5 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v6
                (f2cl-lib:fref c-%data%
                               (6 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v7
                (f2cl-lib:fref c-%data%
                               (7 j)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v8
                (f2cl-lib:fref c-%data%
                               (8 j)
                               ((1 ldc) (1 *))
```

```
                                c-%offset%))
                  (* v9
                     (f2cl-lib:fref c-%data%
                                    (9 j)
                                    ((1 ldc) (1 *))
                                    c-%offset%))
                  (* v10
                     (f2cl-lib:fref c-%data%
                                    (10 j)
                                    ((1 ldc) (1 *))
                                    c-%offset%))))
          (setf (f2cl-lib:fref c-%data%
                               (1 j)
                               ((1 ldc) (1 *))
                               c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (1 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t1)))
          (setf (f2cl-lib:fref c-%data%
                               (2 j)
                               ((1 ldc) (1 *))
                               c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (2 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t2)))
          (setf (f2cl-lib:fref c-%data%
                               (3 j)
                               ((1 ldc) (1 *))
                               c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (3 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t3)))
          (setf (f2cl-lib:fref c-%data%
                               (4 j)
                               ((1 ldc) (1 *))
                               c-%offset%)
                (-
```

```
          (f2cl-lib:fref c-%data%
                         (4 j)
                         ((1 ldc) (1 *))
                         c-%offset%)
          (* sum t4)))
(setf (f2cl-lib:fref c-%data%
                     (5 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
        (-
          (f2cl-lib:fref c-%data%
                         (5 j)
                         ((1 ldc) (1 *))
                         c-%offset%)
          (* sum t5)))
(setf (f2cl-lib:fref c-%data%
                     (6 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
        (-
          (f2cl-lib:fref c-%data%
                         (6 j)
                         ((1 ldc) (1 *))
                         c-%offset%)
          (* sum t6)))
(setf (f2cl-lib:fref c-%data%
                     (7 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
        (-
          (f2cl-lib:fref c-%data%
                         (7 j)
                         ((1 ldc) (1 *))
                         c-%offset%)
          (* sum t7)))
(setf (f2cl-lib:fref c-%data%
                     (8 j)
                     ((1 ldc) (1 *))
                     c-%offset%)
        (-
          (f2cl-lib:fref c-%data%
                         (8 j)
                         ((1 ldc) (1 *))
                         c-%offset%)
          (* sum t8)))
(setf (f2cl-lib:fref c-%data%
```

```
                                        (9 j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                                (-
                                 (f2cl-lib:fref c-%data%
                                                (9 j)
                                                ((1 ldc) (1 *))
                                                c-%offset%)
                                 (* sum t9)))
                            (setf (f2cl-lib:fref c-%data%
                                                 (10 j)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (-
                                   (f2cl-lib:fref c-%data%
                                                  (10 j)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)
                                   (* sum t10)))))
                        (go end_label)))
                    (t
                     (tagbody
                       (f2cl-lib:computed-goto
                        (label210 label230 label250 label270 label290 label310 label330
                         label350 label370 label390)
                        n)
                       (dgemv "No transpose" m n one c ldc v 1 zero work 1)
                       (dger m n (- tau) work 1 v 1 c ldc)
                       (go end_label)
          label210
                       (setf t1
                             (+ one
                                (* (- tau)
                                   (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%)
                                   (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))))
                       (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                     ((> j m) nil)
                         (tagbody
                           (setf (f2cl-lib:fref c-%data%
                                                (j 1)
                                                ((1 ldc) (1 *))
                                                c-%offset%)
                                 (* t1
                                    (f2cl-lib:fref c-%data%
                                                   (j 1)
                                                   ((1 ldc) (1 *))
```

```
                                        c-%offset%)))))
              (go end_label)
   label230
              (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
              (setf t1 (* tau v1))
              (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
              (setf t2 (* tau v2))
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j m) nil)
                (tagbody
                  (setf sum
                          (+
                           (* v1
                              (f2cl-lib:fref c-%data%
                                             (j 1)
                                             ((1 ldc) (1 *))
                                             c-%offset%))
                           (* v2
                              (f2cl-lib:fref c-%data%
                                             (j 2)
                                             ((1 ldc) (1 *))
                                             c-%offset%))))
                  (setf (f2cl-lib:fref c-%data%
                                       (j 1)
                                       ((1 ldc) (1 *))
                                       c-%offset%)
                          (-
                           (f2cl-lib:fref c-%data%
                                          (j 1)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                           (* sum t1)))
                  (setf (f2cl-lib:fref c-%data%
                                       (j 2)
                                       ((1 ldc) (1 *))
                                       c-%offset%)
                          (-
                           (f2cl-lib:fref c-%data%
                                          (j 2)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                           (* sum t2)))))
              (go end_label)
   label250
              (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
              (setf t1 (* tau v1))
```

```
(setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
(setf t2 (* tau v2))
(setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
(setf t3 (* tau v3))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j m) nil)
  (tagbody
    (setf sum
            (+
             (* v1
                (f2cl-lib:fref c-%data%
                               (j 1)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v2
                (f2cl-lib:fref c-%data%
                               (j 2)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v3
                (f2cl-lib:fref c-%data%
                               (j 3)
                               ((1 ldc) (1 *))
                               c-%offset%))))
    (setf (f2cl-lib:fref c-%data%
                         (j 1)
                         ((1 ldc) (1 *))
                         c-%offset%)
            (-
             (f2cl-lib:fref c-%data%
                            (j 1)
                            ((1 ldc) (1 *))
                            c-%offset%)
             (* sum t1)))
    (setf (f2cl-lib:fref c-%data%
                         (j 2)
                         ((1 ldc) (1 *))
                         c-%offset%)
            (-
             (f2cl-lib:fref c-%data%
                            (j 2)
                            ((1 ldc) (1 *))
                            c-%offset%)
             (* sum t2)))
    (setf (f2cl-lib:fref c-%data%
                         (j 3)
```

```
                                    ((1 ldc) (1 *))
                                    c-%offset%)
                            (-
                             (f2cl-lib:fref c-%data%
                                            (j 3)
                                            ((1 ldc) (1 *))
                                            c-%offset%)
                             (* sum t3)))))
                    (go end_label)
  label270
                    (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
                    (setf t1 (* tau v1))
                    (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
                    (setf t2 (* tau v2))
                    (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
                    (setf t3 (* tau v3))
                    (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
                    (setf t4 (* tau v4))
                    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                  ((> j m) nil)
                      (tagbody
                        (setf sum
                                (+
                                 (* v1
                                    (f2cl-lib:fref c-%data%
                                                   (j 1)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%))
                                 (* v2
                                    (f2cl-lib:fref c-%data%
                                                   (j 2)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%))
                                 (* v3
                                    (f2cl-lib:fref c-%data%
                                                   (j 3)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%))
                                 (* v4
                                    (f2cl-lib:fref c-%data%
                                                   (j 4)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%))))
                        (setf (f2cl-lib:fref c-%data%
                                             (j 1)
                                             ((1 ldc) (1 *))
```

```
                                        c-%offset%)
                                (-
                                 (f2cl-lib:fref c-%data%
                                               (j 1)
                                               ((1 ldc) (1 *))
                                               c-%offset%)
                                 (* sum t1)))
                            (setf (f2cl-lib:fref c-%data%
                                                 (j 2)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (-
                                   (f2cl-lib:fref c-%data%
                                                  (j 2)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)
                                   (* sum t2)))
                            (setf (f2cl-lib:fref c-%data%
                                                 (j 3)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (-
                                   (f2cl-lib:fref c-%data%
                                                  (j 3)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)
                                   (* sum t3)))
                            (setf (f2cl-lib:fref c-%data%
                                                 (j 4)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (-
                                   (f2cl-lib:fref c-%data%
                                                  (j 4)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)
                                   (* sum t4)))))
                        (go end_label)
label290
                        (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
                        (setf t1 (* tau v1))
                        (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
                        (setf t2 (* tau v2))
                        (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
                        (setf t3 (* tau v3))
                        (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
```

```
(setf t4 (* tau v4))
(setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
(setf t5 (* tau v5))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j m) nil)
  (tagbody
    (setf sum
            (+
             (* v1
                (f2cl-lib:fref c-%data%
                               (j 1)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v2
                (f2cl-lib:fref c-%data%
                               (j 2)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v3
                (f2cl-lib:fref c-%data%
                               (j 3)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v4
                (f2cl-lib:fref c-%data%
                               (j 4)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v5
                (f2cl-lib:fref c-%data%
                               (j 5)
                               ((1 ldc) (1 *))
                               c-%offset%))))
    (setf (f2cl-lib:fref c-%data%
                         (j 1)
                         ((1 ldc) (1 *))
                         c-%offset%)
            (-
             (f2cl-lib:fref c-%data%
                            (j 1)
                            ((1 ldc) (1 *))
                            c-%offset%)
             (* sum t1)))
    (setf (f2cl-lib:fref c-%data%
                         (j 2)
                         ((1 ldc) (1 *))
```

```
                                        c-%offset%)
                                (-
                                 (f2cl-lib:fref c-%data%
                                               (j 2)
                                               ((1 ldc) (1 *))
                                               c-%offset%)
                                 (* sum t2)))
                            (setf (f2cl-lib:fref c-%data%
                                                 (j 3)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (-
                                   (f2cl-lib:fref c-%data%
                                                  (j 3)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)
                                   (* sum t3)))
                            (setf (f2cl-lib:fref c-%data%
                                                 (j 4)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (-
                                   (f2cl-lib:fref c-%data%
                                                  (j 4)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)
                                   (* sum t4)))
                            (setf (f2cl-lib:fref c-%data%
                                                 (j 5)
                                                 ((1 ldc) (1 *))
                                                 c-%offset%)
                                  (-
                                   (f2cl-lib:fref c-%data%
                                                  (j 5)
                                                  ((1 ldc) (1 *))
                                                  c-%offset%)
                                   (* sum t5)))))
                      (go end_label)
label310
                      (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
                      (setf t1 (* tau v1))
                      (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
                      (setf t2 (* tau v2))
                      (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
                      (setf t3 (* tau v3))
                      (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
```

```
(setf t4 (* tau v4))
(setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
(setf t5 (* tau v5))
(setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
(setf t6 (* tau v6))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j m) nil)
  (tagbody
    (setf sum
            (+
             (* v1
                (f2cl-lib:fref c-%data%
                               (j 1)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v2
                (f2cl-lib:fref c-%data%
                               (j 2)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v3
                (f2cl-lib:fref c-%data%
                               (j 3)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v4
                (f2cl-lib:fref c-%data%
                               (j 4)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v5
                (f2cl-lib:fref c-%data%
                               (j 5)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v6
                (f2cl-lib:fref c-%data%
                               (j 6)
                               ((1 ldc) (1 *))
                               c-%offset%))))
    (setf (f2cl-lib:fref c-%data%
                         (j 1)
                         ((1 ldc) (1 *))
                         c-%offset%)
            (-
             (f2cl-lib:fref c-%data%
```

```
                          (j 1)
                          ((1 ldc) (1 *))
                          c-%offset%)
           (* sum t1)))
(setf (f2cl-lib:fref c-%data%
                     (j 2)
                     ((1 ldc) (1 *))
                     c-%offset%)
        (-
         (f2cl-lib:fref c-%data%
                        (j 2)
                        ((1 ldc) (1 *))
                        c-%offset%)
         (* sum t2)))
(setf (f2cl-lib:fref c-%data%
                     (j 3)
                     ((1 ldc) (1 *))
                     c-%offset%)
        (-
         (f2cl-lib:fref c-%data%
                        (j 3)
                        ((1 ldc) (1 *))
                        c-%offset%)
         (* sum t3)))
(setf (f2cl-lib:fref c-%data%
                     (j 4)
                     ((1 ldc) (1 *))
                     c-%offset%)
        (-
         (f2cl-lib:fref c-%data%
                        (j 4)
                        ((1 ldc) (1 *))
                        c-%offset%)
         (* sum t4)))
(setf (f2cl-lib:fref c-%data%
                     (j 5)
                     ((1 ldc) (1 *))
                     c-%offset%)
        (-
         (f2cl-lib:fref c-%data%
                        (j 5)
                        ((1 ldc) (1 *))
                        c-%offset%)
         (* sum t5)))
(setf (f2cl-lib:fref c-%data%
                     (j 6)
```

```
                                    ((1 ldc) (1 *))
                                    c-%offset%)
                           (-
                            (f2cl-lib:fref c-%data%
                                           (j 6)
                                           ((1 ldc) (1 *))
                                           c-%offset%)
                            (* sum t6)))))
                    (go end_label)
  label330
                    (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
                    (setf t1 (* tau v1))
                    (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
                    (setf t2 (* tau v2))
                    (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
                    (setf t3 (* tau v3))
                    (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
                    (setf t4 (* tau v4))
                    (setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
                    (setf t5 (* tau v5))
                    (setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
                    (setf t6 (* tau v6))
                    (setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
                    (setf t7 (* tau v7))
                    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                  ((> j m) nil)
                      (tagbody
                        (setf sum
                                (+
                                 (* v1
                                    (f2cl-lib:fref c-%data%
                                                   (j 1)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%))
                                 (* v2
                                    (f2cl-lib:fref c-%data%
                                                   (j 2)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%))
                                 (* v3
                                    (f2cl-lib:fref c-%data%
                                                   (j 3)
                                                   ((1 ldc) (1 *))
                                                   c-%offset%))
                                 (* v4
                                    (f2cl-lib:fref c-%data%
```

```
                                (j 4)
                                ((1 ldc) (1 *))
                                c-%offset%))
                   (* v5
                      (f2cl-lib:fref c-%data%
                                (j 5)
                                ((1 ldc) (1 *))
                                c-%offset%))
                   (* v6
                      (f2cl-lib:fref c-%data%
                                (j 6)
                                ((1 ldc) (1 *))
                                c-%offset%))
                   (* v7
                      (f2cl-lib:fref c-%data%
                                (j 7)
                                ((1 ldc) (1 *))
                                c-%offset%))))
             (setf (f2cl-lib:fref c-%data%
                                  (j 1)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                   (-
                    (f2cl-lib:fref c-%data%
                                   (j 1)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                    (* sum t1)))
             (setf (f2cl-lib:fref c-%data%
                                  (j 2)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                   (-
                    (f2cl-lib:fref c-%data%
                                   (j 2)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                    (* sum t2)))
             (setf (f2cl-lib:fref c-%data%
                                  (j 3)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                   (-
                    (f2cl-lib:fref c-%data%
                                   (j 3)
                                   ((1 ldc) (1 *))
```

```
                                    c-%offset%)
                          (* sum t3)))
                    (setf (f2cl-lib:fref c-%data%
                                         (j 4)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                          (-
                           (f2cl-lib:fref c-%data%
                                          (j 4)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                           (* sum t4)))
                    (setf (f2cl-lib:fref c-%data%
                                         (j 5)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                          (-
                           (f2cl-lib:fref c-%data%
                                          (j 5)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                           (* sum t5)))
                    (setf (f2cl-lib:fref c-%data%
                                         (j 6)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                          (-
                           (f2cl-lib:fref c-%data%
                                          (j 6)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                           (* sum t6)))
                    (setf (f2cl-lib:fref c-%data%
                                         (j 7)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                          (-
                           (f2cl-lib:fref c-%data%
                                          (j 7)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                           (* sum t7)))))
              (go end_label)
label350
              (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
              (setf t1 (* tau v1))
```

```
(setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
(setf t2 (* tau v2))
(setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
(setf t3 (* tau v3))
(setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
(setf t4 (* tau v4))
(setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
(setf t5 (* tau v5))
(setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
(setf t6 (* tau v6))
(setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
(setf t7 (* tau v7))
(setf v8 (f2cl-lib:fref v-%data% (8) ((1 *)) v-%offset%))
(setf t8 (* tau v8))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j m) nil)
  (tagbody
    (setf sum
            (+
             (* v1
                (f2cl-lib:fref c-%data%
                               (j 1)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v2
                (f2cl-lib:fref c-%data%
                               (j 2)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v3
                (f2cl-lib:fref c-%data%
                               (j 3)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v4
                (f2cl-lib:fref c-%data%
                               (j 4)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v5
                (f2cl-lib:fref c-%data%
                               (j 5)
                               ((1 ldc) (1 *))
                               c-%offset%))
             (* v6
                (f2cl-lib:fref c-%data%
```

```
                                (j 6)
                                ((1 ldc) (1 *))
                                c-%offset%))
                  (* v7
                     (f2cl-lib:fref c-%data%
                                (j 7)
                                ((1 ldc) (1 *))
                                c-%offset%))
                  (* v8
                     (f2cl-lib:fref c-%data%
                                (j 8)
                                ((1 ldc) (1 *))
                                c-%offset%))))
            (setf (f2cl-lib:fref c-%data%
                                 (j 1)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (-
                   (f2cl-lib:fref c-%data%
                                  (j 1)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                   (* sum t1)))
            (setf (f2cl-lib:fref c-%data%
                                 (j 2)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (-
                   (f2cl-lib:fref c-%data%
                                  (j 2)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                   (* sum t2)))
            (setf (f2cl-lib:fref c-%data%
                                 (j 3)
                                 ((1 ldc) (1 *))
                                 c-%offset%)
                  (-
                   (f2cl-lib:fref c-%data%
                                  (j 3)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                   (* sum t3)))
            (setf (f2cl-lib:fref c-%data%
                                 (j 4)
                                 ((1 ldc) (1 *))
```

```
                        c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 4)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t4)))
(setf (f2cl-lib:fref c-%data%
                     (j 5)
                     ((1 ldc) (1 *))
                     c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 5)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t5)))
(setf (f2cl-lib:fref c-%data%
                     (j 6)
                     ((1 ldc) (1 *))
                     c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 6)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t6)))
(setf (f2cl-lib:fref c-%data%
                     (j 7)
                     ((1 ldc) (1 *))
                     c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 7)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t7)))
(setf (f2cl-lib:fref c-%data%
                     (j 8)
                     ((1 ldc) (1 *))
                     c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 8)
                                ((1 ldc) (1 *))
                                c-%offset%)
```

```
                    (* sum t8)))))
              (go end_label)
   label370
              (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
              (setf t1 (* tau v1))
              (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
              (setf t2 (* tau v2))
              (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
              (setf t3 (* tau v3))
              (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
              (setf t4 (* tau v4))
              (setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
              (setf t5 (* tau v5))
              (setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
              (setf t6 (* tau v6))
              (setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
              (setf t7 (* tau v7))
              (setf v8 (f2cl-lib:fref v-%data% (8) ((1 *)) v-%offset%))
              (setf t8 (* tau v8))
              (setf v9 (f2cl-lib:fref v-%data% (9) ((1 *)) v-%offset%))
              (setf t9 (* tau v9))
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j m) nil)
                (tagbody
                  (setf sum
                          (+
                           (* v1
                              (f2cl-lib:fref c-%data%
                                             (j 1)
                                             ((1 ldc) (1 *))
                                             c-%offset%))
                           (* v2
                              (f2cl-lib:fref c-%data%
                                             (j 2)
                                             ((1 ldc) (1 *))
                                             c-%offset%))
                           (* v3
                              (f2cl-lib:fref c-%data%
                                             (j 3)
                                             ((1 ldc) (1 *))
                                             c-%offset%))
                           (* v4
                              (f2cl-lib:fref c-%data%
                                             (j 4)
                                             ((1 ldc) (1 *))
                                             c-%offset%))
```

```
                (* v5
                   (f2cl-lib:fref c-%data%
                                  (j 5)
                                  ((1 ldc) (1 *))
                                  c-%offset%))
                (* v6
                   (f2cl-lib:fref c-%data%
                                  (j 6)
                                  ((1 ldc) (1 *))
                                  c-%offset%))
                (* v7
                   (f2cl-lib:fref c-%data%
                                  (j 7)
                                  ((1 ldc) (1 *))
                                  c-%offset%))
                (* v8
                   (f2cl-lib:fref c-%data%
                                  (j 8)
                                  ((1 ldc) (1 *))
                                  c-%offset%))
                (* v9
                   (f2cl-lib:fref c-%data%
                                  (j 9)
                                  ((1 ldc) (1 *))
                                  c-%offset%))))
          (setf (f2cl-lib:fref c-%data%
                               (j 1)
                               ((1 ldc) (1 *))
                               c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 1)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t1)))
          (setf (f2cl-lib:fref c-%data%
                               (j 2)
                               ((1 ldc) (1 *))
                               c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 2)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t2)))
          (setf (f2cl-lib:fref c-%data%
```

```
                        (j 3)
                        ((1 ldc) (1 *))
                        c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 3)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t3)))
        (setf (f2cl-lib:fref c-%data%
                        (j 4)
                        ((1 ldc) (1 *))
                        c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 4)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t4)))
        (setf (f2cl-lib:fref c-%data%
                        (j 5)
                        ((1 ldc) (1 *))
                        c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 5)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t5)))
        (setf (f2cl-lib:fref c-%data%
                        (j 6)
                        ((1 ldc) (1 *))
                        c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 6)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t6)))
        (setf (f2cl-lib:fref c-%data%
                        (j 7)
                        ((1 ldc) (1 *))
                        c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 7)
```

```
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                              (* sum t7)))
                    (setf (f2cl-lib:fref c-%data%
                                         (j 8)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                          (-
                           (f2cl-lib:fref c-%data%
                                         (j 8)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                           (* sum t8)))
                    (setf (f2cl-lib:fref c-%data%
                                         (j 9)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                          (-
                           (f2cl-lib:fref c-%data%
                                         (j 9)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                           (* sum t9)))))
              (go end_label)
label390
              (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
              (setf t1 (* tau v1))
              (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
              (setf t2 (* tau v2))
              (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
              (setf t3 (* tau v3))
              (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
              (setf t4 (* tau v4))
              (setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
              (setf t5 (* tau v5))
              (setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
              (setf t6 (* tau v6))
              (setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
              (setf t7 (* tau v7))
              (setf v8 (f2cl-lib:fref v-%data% (8) ((1 *)) v-%offset%))
              (setf t8 (* tau v8))
              (setf v9 (f2cl-lib:fref v-%data% (9) ((1 *)) v-%offset%))
              (setf t9 (* tau v9))
              (setf v10 (f2cl-lib:fref v-%data% (10) ((1 *)) v-%offset%))
              (setf t10 (* tau v10))
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
```

```
            ((> j m) nil)
(tagbody
  (setf sum
          (+
           (* v1
              (f2cl-lib:fref c-%data%
                             (j 1)
                             ((1 ldc) (1 *))
                             c-%offset%))
           (* v2
              (f2cl-lib:fref c-%data%
                             (j 2)
                             ((1 ldc) (1 *))
                             c-%offset%))
           (* v3
              (f2cl-lib:fref c-%data%
                             (j 3)
                             ((1 ldc) (1 *))
                             c-%offset%))
           (* v4
              (f2cl-lib:fref c-%data%
                             (j 4)
                             ((1 ldc) (1 *))
                             c-%offset%))
           (* v5
              (f2cl-lib:fref c-%data%
                             (j 5)
                             ((1 ldc) (1 *))
                             c-%offset%))
           (* v6
              (f2cl-lib:fref c-%data%
                             (j 6)
                             ((1 ldc) (1 *))
                             c-%offset%))
           (* v7
              (f2cl-lib:fref c-%data%
                             (j 7)
                             ((1 ldc) (1 *))
                             c-%offset%))
           (* v8
              (f2cl-lib:fref c-%data%
                             (j 8)
                             ((1 ldc) (1 *))
                             c-%offset%))
           (* v9
              (f2cl-lib:fref c-%data%
```

```
                                (j 9)
                                ((1 ldc) (1 *))
                                c-%offset%))
                 (* v10
                    (f2cl-lib:fref c-%data%
                                (j 10)
                                ((1 ldc) (1 *))
                                c-%offset%))))
          (setf (f2cl-lib:fref c-%data%
                               (j 1)
                               ((1 ldc) (1 *))
                               c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 1)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t1)))
          (setf (f2cl-lib:fref c-%data%
                               (j 2)
                               ((1 ldc) (1 *))
                               c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 2)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t2)))
          (setf (f2cl-lib:fref c-%data%
                               (j 3)
                               ((1 ldc) (1 *))
                               c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 3)
                                ((1 ldc) (1 *))
                                c-%offset%)
                 (* sum t3)))
          (setf (f2cl-lib:fref c-%data%
                               (j 4)
                               ((1 ldc) (1 *))
                               c-%offset%)
                (-
                 (f2cl-lib:fref c-%data%
                                (j 4)
                                ((1 ldc) (1 *))
```

```
                          c-%offset%)
            (* sum t4)))
(setf (f2cl-lib:fref c-%data%
                     (j 5)
                     ((1 ldc) (1 *))
                     c-%offset%)
      (-
       (f2cl-lib:fref c-%data%
                      (j 5)
                      ((1 ldc) (1 *))
                      c-%offset%)
       (* sum t5)))
(setf (f2cl-lib:fref c-%data%
                     (j 6)
                     ((1 ldc) (1 *))
                     c-%offset%)
      (-
       (f2cl-lib:fref c-%data%
                      (j 6)
                      ((1 ldc) (1 *))
                      c-%offset%)
       (* sum t6)))
(setf (f2cl-lib:fref c-%data%
                     (j 7)
                     ((1 ldc) (1 *))
                     c-%offset%)
      (-
       (f2cl-lib:fref c-%data%
                      (j 7)
                      ((1 ldc) (1 *))
                      c-%offset%)
       (* sum t7)))
(setf (f2cl-lib:fref c-%data%
                     (j 8)
                     ((1 ldc) (1 *))
                     c-%offset%)
      (-
       (f2cl-lib:fref c-%data%
                      (j 8)
                      ((1 ldc) (1 *))
                      c-%offset%)
       (* sum t8)))
(setf (f2cl-lib:fref c-%data%
                     (j 9)
                     ((1 ldc) (1 *))
                     c-%offset%)
```

```
                          (-
                           (f2cl-lib:fref c-%data%
                                          (j 9)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                           (* sum t9)))
                    (setf (f2cl-lib:fref c-%data%
                                         (j 10)
                                         ((1 ldc) (1 *))
                                         c-%offset%)
                          (-
                           (f2cl-lib:fref c-%data%
                                          (j 10)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                           (* sum t10)))))
               (go end_label))))
  end_label
         (return (values nil nil nil nil nil nil nil nil))))))
```

7.51 dlartg LAPACK

⟨*dlartg.input*⟩≡

```
)set break resume
)sys rm -f dlartg.output
)spool dlartg.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlartg.help*⟩≡

```
  ======================================================================
  dlartg examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLARTG - a plane rotation so that   [ CS SN ]

  SYNOPSIS
         SUBROUTINE DLARTG( F, G, CS, SN, R )

             DOUBLE         PRECISION CS, F, G, R, SN

  PURPOSE
         DLARTG generate a plane rotation so that
            [ -SN  CS  ]     [ G ]     [ 0 ]

         This  is  a  slower,  more accurate version of the BLAS1 routine DROTG,
         with the following other differences:
            F and G are unchanged on return.
            If G=0, then CS=1 and SN=0.
            If F=0 and (G .ne. 0), then CS=0 and SN=1 without doing any
               floating point operations (saves work in DBDSQR when
               there are zeros on the diagonal).

         If F exceeds G in magnitude, CS will be positive.


  ARGUMENTS
         F       (input) DOUBLE PRECISION
                 The first component of vector to be rotated.

         G       (input) DOUBLE PRECISION
                 The second component of vector to be rotated.

         CS      (output) DOUBLE PRECISION
                 The cosine of the rotation.

         SN      (output) DOUBLE PRECISION
                 The sine of the rotation.

         R       (output) DOUBLE PRECISION
```

```
                The nonzero component of the rotated vector.
```

⟨*LAPACK dlartg*⟩≡

```
(let* ((zero 0.0) (one 1.0) (two 2.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two))
  (let ((safmx2 0.0) (safmin 0.0) (safmn2 0.0) (first$ nil))
    (declare (type (member t nil) first$)
             (type (double-float) safmn2 safmin safmx2))
    (setq first$ t)
    (defun dlartg (f g cs sn r)
      (declare (type (double-float) r sn cs g f))
      (prog ((eps 0.0) (f1 0.0) (g1 0.0) (scale 0.0) (i 0) (count$ 0))
        (declare (type (double-float) eps f1 g1 scale)
                 (type fixnum count$ i))
        (cond
          (first$
           (setf first$ nil)
           (setf safmin (dlamch "S"))
           (setf eps (dlamch "E"))
           (setf safmn2
                   (expt (dlamch "B")
                         (f2cl-lib:int
                          (/
                           (/ (f2cl-lib:flog (/ safmin eps))
                              (f2cl-lib:flog (dlamch "B")))
                           two))))
           (setf safmx2 (/ one safmn2))))
        (cond
          ((= g zero)
           (setf cs one)
           (setf sn zero)
           (setf r f))
          ((= f zero)
           (setf cs zero)
           (setf sn one)
           (setf r g))
          (t
           (setf f1 f)
           (setf g1 g)
           (setf scale (max (abs f1) (abs g1)))
           (cond
             ((>= scale safmx2)
              (tagbody
                (setf count$ 0)
label10
                (setf count$ (f2cl-lib:int-add count$ 1))
```

```
                (setf f1 (* f1 safmn2))
                (setf g1 (* g1 safmn2))
                (setf scale (max (abs f1) (abs g1)))
                (if (>= scale safmx2) (go label10))
                (setf r (f2cl-lib:fsqrt (+ (expt f1 2) (expt g1 2))))
                (setf cs (/ f1 r))
                (setf sn (/ g1 r))
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                              ((> i count$) nil)
                  (tagbody (setf r (* r safmx2)) label20))))
              ((<= scale safmn2)
               (tagbody
                (setf count$ 0)
label30
                (setf count$ (f2cl-lib:int-add count$ 1))
                (setf f1 (* f1 safmx2))
                (setf g1 (* g1 safmx2))
                (setf scale (max (abs f1) (abs g1)))
                (if (<= scale safmn2) (go label30))
                (setf r (f2cl-lib:fsqrt (+ (expt f1 2) (expt g1 2))))
                (setf cs (/ f1 r))
                (setf sn (/ g1 r))
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                              ((> i count$) nil)
                  (tagbody (setf r (* r safmn2)) label40))))
              (t
               (setf r (f2cl-lib:fsqrt (+ (expt f1 2) (expt g1 2))))
               (setf cs (/ f1 r))
               (setf sn (/ g1 r))))
            (cond
              ((and (> (abs f) (abs g)) (< cs zero))
               (setf cs (- cs))
               (setf sn (- sn))
               (setf r (- r))))))
end_label
       (return (values nil nil cs sn r))))))
```

7.52 dlas2 LAPACK

⟨*dlas2.input*⟩≡

```
)set break resume
)sys rm -f dlas2.output
)spool dlas2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlas2.help*⟩≡

```
======================================================================
dlas2 examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DLAS2 - the singular values of the 2-by-2 matrix  [ F G ]  [ 0 H ]

SYNOPSIS
       SUBROUTINE DLAS2( F, G, H, SSMIN, SSMAX )

           DOUBLE        PRECISION F, G, H, SSMAX, SSMIN

PURPOSE
       DLAS2  computes the singular values of the 2-by-2 matrix
          [  F   G  ]
          [  0   H  ].  On return, SSMIN is the smaller singular value and
       SSMAX is the larger singular value.


ARGUMENTS
       F       (input) DOUBLE PRECISION
               The (1,1) element of the 2-by-2 matrix.

       G       (input) DOUBLE PRECISION
               The (1,2) element of the 2-by-2 matrix.

       H       (input) DOUBLE PRECISION
               The (2,2) element of the 2-by-2 matrix.

       SSMIN   (output) DOUBLE PRECISION
               The smaller singular value.

       SSMAX   (output) DOUBLE PRECISION
               The larger singular value.

FURTHER DETAILS
       Barring over/underflow, all output quantities are correct to  within  a
       few  units  in  the  last  place (ulps), even in the absence of a guard
       digit in addition/subtraction.

       In IEEE arithmetic, the code works correctly if one matrix  element  is
```

infinite.

Overflow will not occur unless the largest singular value itself overflows, or is within a few ulps of overflow. (On machines with partial overflow, like the Cray, overflow may occur if the largest singular value is within a factor of 2 of overflow.)

Underflow is harmless if underflow is gradual. Otherwise, results may correspond to a matrix modified by perturbations of size near the underflow threshold.

⟨*LAPACK dlas2*⟩≡

```
(let* ((zero 0.0) (one 1.0) (two 2.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two))
  (defun dlas2 (f g h ssmin ssmax)
    (declare (type (double-float) ssmax ssmin h g f))
    (prog ((as 0.0) (at 0.0) (au 0.0) (c 0.0) (fa 0.0) (fhmn 0.0) (fhmx 0.0)
           (ga 0.0) (ha 0.0))
      (declare (type (double-float) as at au c fa fhmn fhmx ga ha))
      (setf fa (abs f))
      (setf ga (abs g))
      (setf ha (abs h))
      (setf fhmn (min fa ha))
      (setf fhmx (max fa ha))
      (cond
        ((= fhmn zero)
         (setf ssmin zero)
         (cond
           ((= fhmx zero)
            (setf ssmax ga))
           (t
            (setf ssmax
                    (* (max fhmx ga)
                       (f2cl-lib:fsqrt
                        (+ one (expt (/ (min fhmx ga) (max fhmx ga)) 2))))))))
        (t
         (cond
           ((< ga fhmx)
            (setf as (+ one (/ fhmn fhmx)))
            (setf at (/ (- fhmx fhmn) fhmx))
            (setf au (expt (/ ga fhmx) 2))
            (setf c
                    (/ two
                       (+ (f2cl-lib:fsqrt (+ (* as as) au))
                          (f2cl-lib:fsqrt (+ (* at at) au)))))
            (setf ssmin (* fhmn c))
            (setf ssmax (/ fhmx c)))
           (t
            (setf au (/ fhmx ga))
            (cond
              ((= au zero)
               (setf ssmin (/ (* fhmn fhmx) ga))
               (setf ssmax ga))
              (t
               (setf as (+ one (/ fhmn fhmx)))
```

```
                (setf at (/ (- fhmx fhmn) fhmx))
                (setf c
                        (/ one
                           (+ (f2cl-lib:fsqrt (+ one (expt (* as au) 2)))
                              (f2cl-lib:fsqrt (+ one (expt (* at au) 2))))))
                (setf ssmin (* fhmn c au))
                (setf ssmin (+ ssmin ssmin))
                (setf ssmax (/ ga (+ c c)))))))))
        (return (values nil nil nil ssmin ssmax)))))
```

7.53 dlascl LAPACK

⟨*dlascl.input*⟩≡

```
)set break resume
)sys rm -f dlascl.output
)spool dlascl.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlascl.help*⟩≡

```
  ========================================================================
  dlascl examples
  ========================================================================

  ========================================================================
  Man Page Details
  ========================================================================

  NAME
         DLASCL - the M by N real matrix A by the real scalar CTO/CFROM

  SYNOPSIS
         SUBROUTINE DLASCL( TYPE, KL, KU, CFROM, CTO, M, N, A, LDA, INFO )

             CHARACTER      TYPE

             INTEGER        INFO, KL, KU, LDA, M, N

             DOUBLE         PRECISION CFROM, CTO

             DOUBLE         PRECISION A( LDA, * )

  PURPOSE
         DLASCL  multiplies  the  M  by  N  real  matrix  A  by  the real scalar
         CTO/CFROM.  This is done without over/underflow as long as the final
         result  CTO*A(I,J)/CFROM does not over/underflow. TYPE specifies that A
         may be full, upper triangular, lower triangular, upper  Hessenberg,  or
         banded.


  ARGUMENTS
         TYPE    (input) CHARACTER*1
                 TYPE  indices  the storage type of the input matrix.  = 'G':  A
                 is a full matrix.
                 = 'L':  A is a lower triangular matrix.
                 = 'U':  A is an upper triangular matrix.
                 = 'H':  A is an upper Hessenberg matrix.
                 = 'B':  A is a symmetric band matrix with  lower  bandwidth  KL
                 and upper bandwidth KU and with the only the lower half stored.
                 = 'Q':  A is a symmetric band matrix with  lower  bandwidth  KL
                 and upper bandwidth KU and with the only the upper half stored.
                 = 'Z':  A is a band matrix with lower bandwidth  KL  and  upper
                 bandwidth KU.

         KL      (input) INTEGER
```

The lower bandwidth of A. Referenced only if TYPE = 'B', 'Q' or 'Z'.

KU (input) INTEGER
The upper bandwidth of A. Referenced only if TYPE = 'B', 'Q' or 'Z'.

CFROM (input) DOUBLE PRECISION
CTO (input) DOUBLE PRECISION The matrix A is multiplied by CTO/CFROM. A(I,J) is computed without over/underflow if the final result CTO*A(I,J)/CFROM can be represented without over/underflow. CFROM must be nonzero.

M (input) INTEGER
The number of rows of the matrix A. M >= 0.

N (input) INTEGER
The number of columns of the matrix A. N >= 0.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
The matrix to be multiplied by CTO/CFROM. See TYPE for the storage type.

LDA (input) INTEGER
The leading dimension of the array A. LDA >= max(1,M).

INFO (output) INTEGER
0 - successful exit <0 - if INFO = -i, the i-th argument had an illegal value.

⟨*LAPACK dlascl*⟩≡

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dlascl (type kl ku cfrom cto m n a lda info)
    (declare (type (simple-array double-float (*)) a)
             (type (double-float) cto cfrom)
             (type fixnum info lda n m ku kl)
             (type character type))
    (f2cl-lib:with-multi-array-data
        ((type double-float type-%data% type-%offset%)
         (a double-float a-%data% a-%offset%))
      (prog ((bignum 0.0) (cfrom1 0.0) (cfromc 0.0) (cto1 0.0) (ctoc 0.0)
             (mul 0.0) (smlnum 0.0) (i 0) (itype 0) (j 0) (k1 0) (k2 0) (k3 0)
             (k4 0) (done nil))
        (declare (type (double-float) bignum cfrom1 cfromc cto1 ctoc mul
                                      smlnum)
                 (type fixnum i itype j k1 k2 k3 k4)
                 (type (member t nil) done))
        (setf info 0)
        (cond
          ((char-equal type #\G)
           (setf itype 0))
          ((char-equal type #\L)
           (setf itype 1))
          ((char-equal type #\U)
           (setf itype 2))
          ((char-equal type #\H)
           (setf itype 3))
          ((char-equal type #\B)
           (setf itype 4))
          ((char-equal type #\Q)
           (setf itype 5))
          ((char-equal type #\Z)
           (setf itype 6))
          (t
           (setf itype -1)))
        (cond
          ((= itype (f2cl-lib:int-sub 1))
           (setf info -1))
          ((= cfrom zero)
           (setf info -4))
          ((< m 0)
           (setf info -6))
          ((or (< n 0) (and (= itype 4) (/= n m)) (and (= itype 5) (/= n m)))
           (setf info -7))
```

```
            ((and (<= itype 3)
                  (< lda
                     (max (the fixnum 1) (the fixnum m))))
             (setf info -9))
            ((>= itype 4)
             (cond
               ((or (< kl 0)
                    (> kl
                       (max
                        (the fixnum
                             (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
                        (the fixnum 0))))
                (setf info -2))
               ((or (< ku 0)
                    (> ku
                       (max
                        (the fixnum
                             (f2cl-lib:int-add n (f2cl-lib:int-sub 1)))
                        (the fixnum 0)))
                    (and (or (= itype 4) (= itype 5)) (/= kl ku)))
                (setf info -3))
               ((or (and (= itype 4) (< lda (f2cl-lib:int-add kl 1)))
                    (and (= itype 5) (< lda (f2cl-lib:int-add ku 1)))
                    (and (= itype 6)
                         (< lda (f2cl-lib:int-add (f2cl-lib:int-mul 2 kl) ku 1))))
                (setf info -9)))))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DLASCL" (f2cl-lib:int-sub info))
           (go end_label)))
        (if (or (= n 0) (= m 0)) (go end_label))
        (setf smlnum (dlamch "S"))
        (setf bignum (/ one smlnum))
        (setf cfromc cfrom)
        (setf ctoc cto)
  label10
        (setf cfrom1 (* cfromc smlnum))
        (setf cto1 (/ ctoc bignum))
        (cond
          ((and (> (abs cfrom1) (abs ctoc)) (/= ctoc zero))
           (setf mul smlnum)
           (setf done nil)
           (setf cfromc cfrom1))
          ((> (abs cto1) (abs cfromc))
```

```
   (setf mul bignum)
   (setf done nil)
   (setf ctoc cto1))
  (t
   (setf mul (/ ctoc cfromc))
   (setf done t)))
(cond
  ((= itype 0)
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                 ((> j n) nil)
     (tagbody
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i m) nil)
         (tagbody
           (setf (f2cl-lib:fref a-%data%
                                (i j)
                                ((1 lda) (1 *))
                                a-%offset%)
                 (*
                  (f2cl-lib:fref a-%data%
                                 (i j)
                                 ((1 lda) (1 *))
                                 a-%offset%)
                  mul)))))))
  ((= itype 1)
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                 ((> j n) nil)
     (tagbody
       (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                     ((> i m) nil)
         (tagbody
           (setf (f2cl-lib:fref a-%data%
                                (i j)
                                ((1 lda) (1 *))
                                a-%offset%)
                 (*
                  (f2cl-lib:fref a-%data%
                                 (i j)
                                 ((1 lda) (1 *))
                                 a-%offset%)
                  mul)))))))
  ((= itype 2)
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                 ((> j n) nil)
     (tagbody
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
```

```
                      ((> i
                          (min (the fixnum j)
                               (the fixnum m)))
                       nil)
             (tagbody
               (setf (f2cl-lib:fref a-%data%
                                    (i j)
                                    ((1 lda) (1 *))
                                    a-%offset%)
                     (*
                      (f2cl-lib:fref a-%data%
                                     (i j)
                                     ((1 lda) (1 *))
                                     a-%offset%)
                      mul)))))))
     ((= itype 3)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i
                            (min
                             (the fixnum
                                  (f2cl-lib:int-add j 1))
                             (the fixnum m)))
                         nil)
            (tagbody
              (setf (f2cl-lib:fref a-%data%
                                   (i j)
                                   ((1 lda) (1 *))
                                   a-%offset%)
                    (*
                     (f2cl-lib:fref a-%data%
                                    (i j)
                                    ((1 lda) (1 *))
                                    a-%offset%)
                     mul)))))))
     ((= itype 4)
      (setf k3 (f2cl-lib:int-add kl 1))
      (setf k4 (f2cl-lib:int-add n 1))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i
                            (min (the fixnum k3)
```

```
                                    (the fixnum
                                         (f2cl-lib:int-add k4
                                                           (f2cl-lib:int-sub
                                                            j)))))
                             nil)
                       (tagbody
                         (setf (f2cl-lib:fref a-%data%
                                              (i j)
                                              ((1 lda) (1 *))
                                              a-%offset%)
                               (*
                                (f2cl-lib:fref a-%data%
                                               (i j)
                                               ((1 lda) (1 *))
                                               a-%offset%)
                                mul)))))))
                  ((= itype 5)
                   (setf k1 (f2cl-lib:int-add ku 2))
                   (setf k3 (f2cl-lib:int-add ku 1))
                   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                 ((> j n) nil)
                     (tagbody
                       (f2cl-lib:fdo (i
                                      (max
                                       (the fixnum
                                            (f2cl-lib:int-add k1 (f2cl-lib:int-sub j)))
                                       (the fixnum 1))
                                      (f2cl-lib:int-add i 1))
                                     ((> i k3) nil)
                         (tagbody
                           (setf (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%)
                                 (*
                                  (f2cl-lib:fref a-%data%
                                                 (i j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%)
                                  mul)))))))
                  ((= itype 6)
                   (setf k1 (f2cl-lib:int-add kl ku 2))
                   (setf k2 (f2cl-lib:int-add kl 1))
                   (setf k3 (f2cl-lib:int-add (f2cl-lib:int-mul 2 kl) ku 1))
                   (setf k4 (f2cl-lib:int-add kl ku 1 m))
                   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
```

```
                        ((> j n) nil)
                (tagbody
                  (f2cl-lib:fdo (i
                                 (max
                                  (the fixnum
                                       (f2cl-lib:int-add k1 (f2cl-lib:int-sub j)))
                                  (the fixnum k2))
                                 (f2cl-lib:int-add i 1))
                                ((> i
                                    (min (the fixnum k3)
                                         (the fixnum
                                              (f2cl-lib:int-add k4
                                                                (f2cl-lib:int-sub
                                                                 j)))))
                                 nil)
                    (tagbody
                      (setf (f2cl-lib:fref a-%data%
                                           (i j)
                                           ((1 lda) (1 *))
                                           a-%offset%)
                            (*
                             (f2cl-lib:fref a-%data%
                                            (i j)
                                            ((1 lda) (1 *))
                                            a-%offset%)
                             mul))))))))
          (if (not done) (go label10))
   end_label
          (return (values nil nil nil nil nil nil nil nil nil info))))))
```

7.54 dlasd0 LAPACK

⟨*dlasd0.input*⟩≡

```
  )set break resume
  )sys rm -f dlasd0.output
  )spool dlasd0.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dlasd0.help*⟩≡

```
======================================================================
dlasd0 examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DLASD0 - divide and conquer approach, DLASD0 computes the singular
       value decomposition (SVD) of a real upper bidiagonal N-by-M matrix B
       with diagonal D and offdiagonal E, where M = N + SQRE

SYNOPSIS
       SUBROUTINE DLASD0( N, SQRE, D, E, U, LDU, VT, LDVT, SMLSIZ, IWORK,
                          WORK, INFO )

           INTEGER         INFO, LDU, LDVT, N, SMLSIZ, SQRE

           INTEGER         IWORK( * )

           DOUBLE          PRECISION D( * ), E( * ), U( LDU, * ), VT( LDVT, *
                           ), WORK( * )

PURPOSE
       Using a divide and conquer approach, DLASD0 computes the singular value
       decomposition (SVD) of a real upper bidiagonal N-by-M matrix B with
       diagonal D and offdiagonal E, where M = N + SQRE. The algorithm com-
       putes orthogonal matrices U and VT such that B = U * S * VT. The singu-
       lar values S are overwritten on D.

       A related subroutine, DLASDA, computes only the singular values, and
       optionally, the singular vectors in compact form.


ARGUMENTS
       N      (input) INTEGER
              On entry, the row dimension of the upper bidiagonal matrix.
              This is also the dimension of the main diagonal array D.

       SQRE   (input) INTEGER
              Specifies the column dimension of the bidiagonal matrix. = 0:
              The bidiagonal matrix has column dimension M = N;
              = 1: The bidiagonal matrix has column dimension M = N+1;
```

```
D       (input/output) DOUBLE PRECISION array, dimension (N)
        On entry D contains the main diagonal of the bidiagonal  matrix.
        On exit D, if INFO = 0, contains its singular values.

E       (input) DOUBLE PRECISION array, dimension (M-1)
        Contains  the  subdiagonal entries of the bidiagonal matrix.  On
        exit, E has been destroyed.

U       (output) DOUBLE PRECISION array, dimension at least (LDQ, N)
        On exit, U contains the left singular vectors.

LDU     (input) INTEGER
        On entry, leading dimension of U.

VT      (output) DOUBLE PRECISION array, dimension at least (LDVT, M)
        On exit, VT' contains the right singular vectors.

LDVT    (input) INTEGER
        On entry, leading dimension of VT.

        SMLSIZ (input) INTEGER On entry, maximum size of the subproblems
        at the bottom of the computation tree.

IWORK   (workspace) INTEGER work array.
        Dimension must be at least (8 * N)

WORK    (workspace) DOUBLE PRECISION work array.
        Dimension must be at least (3 * M**2 + 2 * M)

INFO    (output) INTEGER
        = 0:  successful exit.
        < 0:  if INFO = -i, the i-th argument had an illegal value.
        > 0:  if INFO = 1, an singular value did not converge
```

⟨*LAPACK dlasd0*⟩≡

```
(defun dlasd0 (n sqre d e u ldu vt ldvt smlsiz iwork work info)
  (declare (type (simple-array fixnum (*)) iwork)
           (type (simple-array double-float (*)) work vt u e d)
           (type fixnum info smlsiz ldvt ldu sqre n))
  (f2cl-lib:with-multi-array-data
      ((d double-float d-%data% d-%offset%)
       (e double-float e-%data% e-%offset%)
       (u double-float u-%data% u-%offset%)
       (vt double-float vt-%data% vt-%offset%)
       (work double-float work-%data% work-%offset%)
       (iwork fixnum iwork-%data% iwork-%offset%))
    (prog ((alpha 0.0) (beta 0.0) (i 0) (i1 0) (ic 0) (idxq 0) (idxqc 0)
           (im1 0) (inode 0) (itemp 0) (iwk 0) (j 0) (lf 0) (ll 0) (lvl 0)
           (m 0) (ncc 0) (nd 0) (ndb1 0) (ndiml 0) (ndimr 0) (nl 0) (nlf 0)
           (nlp1 0) (nlvl 0) (nr 0) (nrf 0) (nrp1 0) (sqrei 0))
      (declare (type fixnum sqrei nrp1 nrf nr nlvl nlp1 nlf nl
                                       ndimr ndiml ndb1 nd ncc m lvl ll lf j
                                       iwk itemp inode im1 idxqc idxq ic i1
                                       i)
               (type (double-float) beta alpha))
      (setf info 0)
      (cond
        ((< n 0)
         (setf info -1))
        ((or (< sqre 0) (> sqre 1))
         (setf info -2)))
      (setf m (f2cl-lib:int-add n sqre))
      (cond
        ((< ldu n)
         (setf info -6))
        ((< ldvt m)
         (setf info -8))
        ((< smlsiz 3)
         (setf info -9)))
      (cond
        ((/= info 0)
         (error
          " ** On entry to ~a parameter number ~a had an illegal value~%"
          "DLASD0" (f2cl-lib:int-sub info))
         (go end_label)))
      (cond
        ((<= n smlsiz)
         (multiple-value-bind
               (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
                var-10 var-11 var-12 var-13 var-14 var-15)
```

```
        (dlasdq "U" sqre n m n 0 d e vt ldvt u ldu u ldu work info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                       var-8 var-9 var-10 var-11 var-12 var-13 var-14))
      (setf info var-15))
    (go end_label)))
(setf inode 1)
(setf ndiml (f2cl-lib:int-add inode n))
(setf ndimr (f2cl-lib:int-add ndiml n))
(setf idxq (f2cl-lib:int-add ndimr n))
(setf iwk (f2cl-lib:int-add idxq n))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
    (dlasdt n nlvl nd
     (f2cl-lib:array-slice iwork fixnum (inode) ((1 *)))
     (f2cl-lib:array-slice iwork fixnum (ndiml) ((1 *)))
     (f2cl-lib:array-slice iwork fixnum (ndimr) ((1 *)))
     smlsiz)
  (declare (ignore var-0 var-3 var-4 var-5 var-6))
  (setf nlvl var-1)
  (setf nd var-2))
(setf ndb1 (the fixnum (truncate (+ nd 1) 2)))
(setf ncc 0)
(f2cl-lib:fdo (i ndb1 (f2cl-lib:int-add i 1))
              ((> i nd) nil)
  (tagbody
    (setf i1 (f2cl-lib:int-sub i 1))
    (setf ic
            (f2cl-lib:fref iwork-%data%
                           ((f2cl-lib:int-add inode i1))
                           ((1 *))
                           iwork-%offset%))
    (setf nl
            (f2cl-lib:fref iwork-%data%
                           ((f2cl-lib:int-add ndiml i1))
                           ((1 *))
                           iwork-%offset%))
    (setf nlp1 (f2cl-lib:int-add nl 1))
    (setf nr
            (f2cl-lib:fref iwork-%data%
                           ((f2cl-lib:int-add ndimr i1))
                           ((1 *))
                           iwork-%offset%))
    (setf nrp1 (f2cl-lib:int-add nr 1))
    (setf nlf (f2cl-lib:int-sub ic nl))
    (setf nrf (f2cl-lib:int-add ic 1))
    (setf sqrei 1)
    (multiple-value-bind
```

```
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
       var-10 var-11 var-12 var-13 var-14 var-15)
    (dlasdq "U" sqrei nl nlp1 nl ncc
     (f2cl-lib:array-slice d double-float (nlf) ((1 *)))
     (f2cl-lib:array-slice e double-float (nlf) ((1 *)))
     (f2cl-lib:array-slice vt
                           double-float
                           (nlf nlf)
                           ((1 ldvt) (1 *)))
     ldvt
     (f2cl-lib:array-slice u double-float (nlf nlf) ((1 ldu) (1 *)))
     ldu
     (f2cl-lib:array-slice u double-float (nlf nlf) ((1 ldu) (1 *)))
     ldu work info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                   var-8 var-9 var-10 var-11 var-12 var-13 var-14))
  (setf info var-15))
(cond
  ((/= info 0)
   (go end_label)))
(setf itemp (f2cl-lib:int-sub (f2cl-lib:int-add idxq nlf) 2))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j nl) nil)
  (tagbody
    (setf (f2cl-lib:fref iwork-%data%
                         ((f2cl-lib:int-add itemp j))
                         ((1 *))
                         iwork-%offset%)
            j)))
(cond
  ((= i nd)
   (setf sqrei sqre))
  (t
   (setf sqrei 1)))
(setf nrp1 (f2cl-lib:int-add nr sqrei))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
       var-10 var-11 var-12 var-13 var-14 var-15)
    (dlasdq "U" sqrei nr nrp1 nr ncc
     (f2cl-lib:array-slice d double-float (nrf) ((1 *)))
     (f2cl-lib:array-slice e double-float (nrf) ((1 *)))
     (f2cl-lib:array-slice vt
                           double-float
                           (nrf nrf)
                           ((1 ldvt) (1 *)))
     ldvt
```

```
           (f2cl-lib:array-slice u double-float (nrf nrf) ((1 ldu) (1 *)))
           ldu
           (f2cl-lib:array-slice u double-float (nrf nrf) ((1 ldu) (1 *)))
           ldu work info)
         (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                          var-8 var-9 var-10 var-11 var-12 var-13 var-14))
         (setf info var-15))
       (cond
         ((/= info 0)
          (go end_label)))
       (setf itemp (f2cl-lib:int-add idxq ic))
       (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                     ((> j nr) nil)
         (tagbody
           (setf (f2cl-lib:fref iwork-%data%
                                ((f2cl-lib:int-sub
                                  (f2cl-lib:int-add itemp j)
                                  1))
                                ((1 *))
                                iwork-%offset%)
                 j)))))
  (f2cl-lib:fdo (lvl nlvl (f2cl-lib:int-add lvl (f2cl-lib:int-sub 1)))
                ((> lvl 1) nil)
    (tagbody
      (cond
        ((= lvl 1)
         (setf lf 1)
         (setf ll 1))
        (t
         (setf lf (expt 2 (f2cl-lib:int-sub lvl 1)))
         (setf ll (f2cl-lib:int-sub (f2cl-lib:int-mul 2 lf) 1))))
      (f2cl-lib:fdo (i lf (f2cl-lib:int-add i 1))
                    ((> i ll) nil)
        (tagbody
          (setf im1 (f2cl-lib:int-sub i 1))
          (setf ic
                  (f2cl-lib:fref iwork-%data%
                                 ((f2cl-lib:int-add inode im1))
                                 ((1 *))
                                 iwork-%offset%))
          (setf nl
                  (f2cl-lib:fref iwork-%data%
                                 ((f2cl-lib:int-add ndiml im1))
                                 ((1 *))
                                 iwork-%offset%))
          (setf nr
```

```
                (f2cl-lib:fref iwork-%data%
                               ((f2cl-lib:int-add ndimr im1))
                               ((1 *))
                               iwork-%offset%))
              (setf nlf (f2cl-lib:int-sub ic nl))
              (cond
                ((and (= sqre 0) (= i ll))
                 (setf sqrei sqre))
                (t
                 (setf sqrei 1)))
              (setf idxqc (f2cl-lib:int-sub (f2cl-lib:int-add idxq nlf) 1))
              (setf alpha (f2cl-lib:fref d-%data% (ic) ((1 *)) d-%offset%))
              (setf beta (f2cl-lib:fref e-%data% (ic) ((1 *)) e-%offset%))
              (multiple-value-bind
                    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                     var-9 var-10 var-11 var-12 var-13)
                  (dlasd1 nl nr sqrei
                   (f2cl-lib:array-slice d double-float (nlf) ((1 *))) alpha
                   beta
                   (f2cl-lib:array-slice u
                                         double-float
                                         (nlf nlf)
                                         ((1 ldu) (1 *)))
                   ldu
                   (f2cl-lib:array-slice vt
                                         double-float
                                         (nlf nlf)
                                         ((1 ldvt) (1 *)))
                   ldvt
                   (f2cl-lib:array-slice iwork
                                         fixnum
                                         (idxqc)
                                         ((1 *)))
                   (f2cl-lib:array-slice iwork fixnum (iwk) ((1 *)))
                   work info)
                (declare (ignore var-0 var-1 var-2 var-3 var-6 var-7 var-8
                                 var-9 var-10 var-11 var-12))
                (setf alpha var-4)
                (setf beta var-5)
                (setf info var-13))
              (cond
                ((/= info 0)
                 (go end_label)))))))
end_label
     (return (values nil nil nil nil nil nil nil nil nil nil nil info)))))
```

7.55 dlasd1 LAPACK

⟨*dlasd1.input*⟩≡

```
)set break resume
)sys rm -f dlasd1.output
)spool dlasd1.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasd1.help*⟩≡

```
======================================================================
dlasd1 examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DLASD1 - the SVD of an upper bidiagonal N-by-M matrix B,

SYNOPSIS
       SUBROUTINE DLASD1( NL,  NR,  SQRE,  D,  ALPHA,  BETA, U, LDU, VT, LDVT,
                          IDXQ, IWORK, WORK, INFO )

           INTEGER        INFO, LDU, LDVT, NL, NR, SQRE

           DOUBLE         PRECISION ALPHA, BETA

           INTEGER        IDXQ( * ), IWORK( * )

           DOUBLE         PRECISION D( * ), U( LDU, * ), VT( LDVT, * ),  WORK(
                          * )

PURPOSE
       DLASD1 computes the SVD of an upper bidiagonal N-by-M matrix B, where N
       = NL + NR + 1 and M = N + SQRE. DLASD1 is called from DLASD0.

       A related subroutine DLASD7 handles the case in which the singular val-
       ues (and the singular vectors in factored form) are desired.

       DLASD1 computes the SVD as follows:

                    ( D1(in)  0    0     0 )
         B = U(in) * (   Z1'   a   Z2'    b ) * VT(in)
                    (   0     0   D2(in) 0 )

           = U(out) * ( D(out) 0) * VT(out)

       where  Z'  =  (Z1'  a Z2' b) = u' VT', and u is a vector of dimension M
       with ALPHA and BETA in the NL+1 and NL+2 th  entries  and  zeros  else-
       where; and the entry b is empty if SQRE = 0.

       The  left  singular vectors of the original matrix are stored in U, and
       the transpose of the right singular vectors are stored in VT,  and  the
```

```
        singular values are in D.  The algorithm consists of three stages:

           The first stage consists of deflating the size of the problem
           when there are multiple singular values or when there are zeros in
           the Z vector.  For each such occurence the dimension of the
           secular equation problem is reduced by one.  This stage is
           performed by the routine DLASD2.

           The second stage consists of calculating the updated
           singular values. This is done by finding the square roots of the
           roots of the secular equation via the routine DLASD4 (as called
           by DLASD3). This routine also calculates the singular vectors of
           the current problem.

           The final stage consists of computing the updated singular vectors
           directly using the updated singular values.  The singular vectors
           for the current problem are multiplied with the singular vectors
           from the overall problem.


ARGUMENTS
       NL     (input) INTEGER
              The row dimension of the upper block.  NL >= 1.

       NR     (input) INTEGER
              The row dimension of the lower block.  NR >= 1.

       SQRE   (input) INTEGER
              = 0: the lower block is an NR-by-NR square matrix.
              = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

              The  bidiagonal  matrix  has  row dimension N = NL + NR + 1, and
              column dimension M = N + SQRE.

       D      (input/output) DOUBLE PRECISION array,
              dimension (N = NL+NR+1).  On  entry  D(1:NL,1:NL)  contains  the
              singular values of the
              upper block; and D(NL+2:N) contains the singular values of
              the  lower block. On exit D(1:N) contains the singular values of
              the modified matrix.

       ALPHA  (input/output) DOUBLE PRECISION
              Contains the diagonal element associated with the added row.

       BETA   (input/output) DOUBLE PRECISION
              Contains the off-diagonal element associated with the added row.
```

```
U      (input/output) DOUBLE PRECISION array, dimension(LDU,N)
       On entry U(1:NL, 1:NL) contains the left singular vectors of
       the  upper  block;  U(NL+2:N, NL+2:N) contains the left singular
       vectors of the lower block. On exit U contains the left singular
       vectors of the bidiagonal matrix.

LDU    (input) INTEGER
       The leading dimension of the array U.  LDU >= max( 1, N ).

VT     (input/output) DOUBLE PRECISION array, dimension(LDVT,M)
       where  M  = N + SQRE.  On entry VT(1:NL+1, 1:NL+1)' contains the
       right singular
       vectors of the upper block;  VT(NL+2:M,  NL+2:M)'  contains  the
       right  singular vectors of the lower block. On exit VT' contains
       the right singular vectors of the bidiagonal matrix.

LDVT   (input) INTEGER
       The leading dimension of the array VT.  LDVT >= max( 1, M ).

IDXQ  (output) INTEGER array, dimension(N)
      This contains the permutation which will reintegrate the subprob-
      lem just solved back into sorted order, i.e.  D( IDXQ( I = 1, N )
      ) will be in ascending order.

IWORK  (workspace) INTEGER array, dimension( 4 * N )

WORK   (workspace) DOUBLE PRECISION array, dimension( 3*M**2 + 2*M )

INFO   (output) INTEGER
       = 0:  successful exit.
       < 0:  if INFO = -i, the i-th argument had an illegal value.
       > 0:  if INFO = 1, an singular value did not converge
```

⟨*LAPACK dlasd1*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dlasd1 (nl nr sqre d alpha beta u ldu vt ldvt idxq iwork work info)
    (declare (type (simple-array fixnum (*)) iwork idxq)
             (type (double-float) beta alpha)
             (type (simple-array double-float (*)) work vt u d)
             (type fixnum info ldvt ldu sqre nr nl))
    (f2cl-lib:with-multi-array-data
        ((d double-float d-%data% d-%offset%)
         (u double-float u-%data% u-%offset%)
         (vt double-float vt-%data% vt-%offset%)
         (work double-float work-%data% work-%offset%)
         (idxq fixnum idxq-%data% idxq-%offset%)
         (iwork fixnum iwork-%data% iwork-%offset%))
      (prog ((orgnrm 0.0) (coltyp 0) (i 0) (idx 0) (idxc 0) (idxp 0) (iq 0)
             (isigma 0) (iu2 0) (ivt2 0) (iz 0) (k 0) (ldq 0) (ldu2 0)
             (ldvt2 0) (m 0) (n 0) (n1 0) (n2 0))
        (declare (type (double-float) orgnrm)
                 (type fixnum coltyp i idx idxc idxp iq isigma iu2
                                          ivt2 iz k ldq ldu2 ldvt2 m n n1 n2))
        (setf info 0)
        (cond
          ((< nl 1)
           (setf info -1))
          ((< nr 1)
           (setf info -2))
          ((or (< sqre 0) (> sqre 1))
           (setf info -3)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DLASD1" (f2cl-lib:int-sub info))
           (go end_label)))
        (setf n (f2cl-lib:int-add nl nr 1))
        (setf m (f2cl-lib:int-add n sqre))
        (setf ldu2 n)
        (setf ldvt2 m)
        (setf iz 1)
        (setf isigma (f2cl-lib:int-add iz m))
        (setf iu2 (f2cl-lib:int-add isigma n))
        (setf ivt2 (f2cl-lib:int-add iu2 (f2cl-lib:int-mul ldu2 n)))
        (setf iq (f2cl-lib:int-add ivt2 (f2cl-lib:int-mul ldvt2 m)))
        (setf idx 1)
```

```
(setf idxc (f2cl-lib:int-add idx n))
(setf coltyp (f2cl-lib:int-add idxc n))
(setf idxp (f2cl-lib:int-add coltyp n))
(setf orgnrm (max (abs alpha) (abs beta)))
(setf (f2cl-lib:fref d-%data%
                     ((f2cl-lib:int-add nl 1))
                     ((1 *))
                     d-%offset%)
       zero)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i n) nil)
  (tagbody
    (cond
      ((> (abs (f2cl-lib:fref d (i) ((1 *)))) orgnrm)
       (setf orgnrm
             (abs (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)))))))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
    (dlascl "G" 0 0 orgnrm one n 1 d n info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                   var-8))
  (setf info var-9))
(setf alpha (/ alpha orgnrm))
(setf beta (/ beta orgnrm))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
       var-10 var-11 var-12 var-13 var-14 var-15 var-16 var-17 var-18
       var-19 var-20 var-21 var-22)
    (dlasd2 nl nr sqre k d
     (f2cl-lib:array-slice work double-float (iz) ((1 *))) alpha beta u
     ldu vt ldvt
     (f2cl-lib:array-slice work double-float (isigma) ((1 *)))
     (f2cl-lib:array-slice work double-float (iu2) ((1 *))) ldu2
     (f2cl-lib:array-slice work double-float (ivt2) ((1 *))) ldvt2
     (f2cl-lib:array-slice iwork fixnum (idxp) ((1 *)))
     (f2cl-lib:array-slice iwork fixnum (idx) ((1 *)))
     (f2cl-lib:array-slice iwork fixnum (idxc) ((1 *))) idxq
     (f2cl-lib:array-slice iwork fixnum (coltyp) ((1 *)))
     info)
  (declare (ignore var-0 var-1 var-2 var-4 var-5 var-6 var-7 var-8
                   var-9 var-10 var-11 var-12 var-13 var-14 var-15
                   var-16 var-17 var-18 var-19 var-20 var-21))
  (setf k var-3)
  (setf info var-22))
(setf ldq k)
(multiple-value-bind
```

```
              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
               var-10 var-11 var-12 var-13 var-14 var-15 var-16 var-17 var-18
               var-19)
            (dlasd3 nl nr sqre k d
             (f2cl-lib:array-slice work double-float (iq) ((1 *))) ldq
             (f2cl-lib:array-slice work double-float (isigma) ((1 *))) u ldu
             (f2cl-lib:array-slice work double-float (iu2) ((1 *))) ldu2 vt
             ldvt (f2cl-lib:array-slice work double-float (ivt2) ((1 *))) ldvt2
             (f2cl-lib:array-slice iwork fixnum (idxc) ((1 *)))
             (f2cl-lib:array-slice iwork fixnum (coltyp) ((1 *)))
             (f2cl-lib:array-slice work double-float (iz) ((1 *))) info)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                           var-8 var-9 var-10 var-11 var-12 var-13 var-14
                           var-15 var-16 var-17 var-18))
          (setf info var-19))
        (cond
          ((/= info 0)
           (go end_label)))
        (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
            (dlascl "G" 0 0 one orgnrm n 1 d n info)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                           var-8))
          (setf info var-9))
        (setf n1 k)
        (setf n2 (f2cl-lib:int-sub n k))
        (dlamrg n1 n2 d 1 -1 idxq)
end_label
        (return
         (values nil
                 nil
                 nil
                 nil
                 alpha
                 beta
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 info))))))
```

7.56 dlasd2 LAPACK

⟨*dlasd2.input*⟩≡

```
)set break resume
)sys rm -f dlasd2.output
)spool dlasd2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasd2.help*⟩≡

```
  ======================================================================
  dlasd2 examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLASD2  - the two sets of singular values together into a single sorted
         set

  SYNOPSIS
         SUBROUTINE DLASD2( NL, NR, SQRE, K, D, Z,  ALPHA,  BETA,  U,  LDU,  VT,
                            LDVT, DSIGMA, U2, LDU2, VT2, LDVT2, IDXP, IDX, IDXC,
                            IDXQ, COLTYP, INFO )

             INTEGER        INFO, K, LDU, LDU2, LDVT, LDVT2, NL, NR, SQRE

             DOUBLE         PRECISION ALPHA, BETA

             INTEGER        COLTYP( * ), IDX( * ), IDXC( * ), IDXP( * ), IDXQ( *
                            )

             DOUBLE         PRECISION  D(  *  ),  DSIGMA(  * ), U( LDU, * ), U2(
                            LDU2, * ), VT( LDVT, * ), VT2( LDVT2, * ), Z( * )

  PURPOSE
         DLASD2 merges the two sets of singular values together  into  a  single
         sorted  set.   Then it tries to deflate the size of the problem.  There
         are two ways in which deflation can occur:  when two or  more  singular
         values  are close together or if there is a tiny entry in the Z vector.
         For each such occurrence the order  of  the  related  secular  equation
         problem is reduced by one.

         DLASD2 is called from DLASD1.


  ARGUMENTS
         NL     (input) INTEGER
                The row dimension of the upper block.  NL >= 1.

         NR     (input) INTEGER
                The row dimension of the lower block.  NR >= 1.
```

```
SQRE   (input) INTEGER
       = 0: the lower block is an NR-by-NR square matrix.
       = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

       The  bidiagonal matrix has N = NL + NR + 1 rows and M = N + SQRE
       >= N columns.

K      (output) INTEGER
       Contains the dimension of the non-deflated matrix, This  is  the
       order of the related secular equation. 1 <= K <=N.

D      (input/output) DOUBLE PRECISION array, dimension(N)
       On  entry  D contains the singular values of the two submatrices
       to be combined.  On exit D contains the trailing  (N-K)  updated
       singular values (those which were deflated) sorted into increas-
       ing order.

Z      (output) DOUBLE PRECISION array, dimension(N)
       On exit Z contains the updating row vector in the secular  equa-
       tion.

ALPHA  (input) DOUBLE PRECISION
       Contains the diagonal element associated with the added row.

BETA   (input) DOUBLE PRECISION
       Contains the off-diagonal element associated with the added row.

U      (input/output) DOUBLE PRECISION array, dimension(LDU,N)
       On entry U contains the left singular vectors of two submatrices
       in  the  two  square blocks with corners at (1,1), (NL, NL), and
       (NL+2, NL+2), (N,N).  On exit  U  contains  the  trailing  (N-K)
       updated left singular vectors (those which were deflated) in its
       last N-K columns.

LDU    (input) INTEGER
       The leading dimension of the array U.  LDU >= N.

VT     (input/output) DOUBLE PRECISION array, dimension(LDVT,M)
       On entry VT' contains the right singular vectors of  two  subma-
       trices  in  the  two square blocks with corners at (1,1), (NL+1,
       NL+1), and (NL+2, NL+2), (M,M).  On exit VT' contains the trail-
       ing  (N-K)  updated  right  singular  vectors  (those which were
       deflated) in its last N-K columns.  In case SQRE  =1,  the  last
       row of VT spans the right null space.

LDVT   (input) INTEGER
```

The leading dimension of the array VT. LDVT >= M.

DSIGMA (output) DOUBLE PRECISION array, dimension (N) Contains a copy of the diagonal elements (K-1 singular values and one zero) in the secular equation.

U2 (output) DOUBLE PRECISION array, dimension(LDU2,N)
Contains a copy of the first K-1 left singular vectors which will be used by DLASD3 in a matrix multiply (DGEMM) to solve for the new left singular vectors. U2 is arranged into four blocks. The first block contains a column with 1 at NL+1 and zero everywhere else; the second block contains non-zero entries only at and above NL; the third contains non-zero entries only below NL+1; and the fourth is dense.

LDU2 (input) INTEGER
The leading dimension of the array U2. LDU2 >= N.

VT2 (output) DOUBLE PRECISION array, dimension(LDVT2,N)
VT2' contains a copy of the first K right singular vectors which will be used by DLASD3 in a matrix multiply (DGEMM) to solve for the new right singular vectors. VT2 is arranged into three blocks. The first block contains a row that corresponds to the special 0 diagonal element in SIGMA; the second block contains non-zeros only at and before NL +1; the third block contains non-zeros only at and after NL +2.

LDVT2 (input) INTEGER
The leading dimension of the array VT2. LDVT2 >= M.

IDXP (workspace) INTEGER array dimension(N)
This will contain the permutation used to place deflated values of D at the end of the array. On output IDXP(2:K)
points to the nondeflated D-values and IDXP(K+1:N) points to the deflated singular values.

IDX (workspace) INTEGER array dimension(N)
This will contain the permutation used to sort the contents of D into ascending order.

IDXC (output) INTEGER array dimension(N)
This will contain the permutation used to arrange the columns of the deflated U matrix into three groups: the first group contains non-zero entries only at and above NL, the second contains non-zero entries only below NL+2, and the third is dense.

```
IDXQ   (input/output) INTEGER array dimension(N)
       This contains the permutation which separately sorts the two
       sub-problems in D into ascending order.  Note that entries in
       the first hlaf of this permutation must first be moved one posi-
       tion backward; and entries in the second half must first have
       NL+1 added to their values.

       COLTYP (workspace/output) INTEGER array dimension(N) As
       workspace, this will contain a label which will indicate which
       of the following types a column in the U2 matrix or a row in the
       VT2 matrix is:
       1 : non-zero in the upper half only
       2 : non-zero in the lower half only
       3 : dense
       4 : deflated

       On exit, it is an array of dimension 4, with COLTYP(I) being the
       dimension of the I-th type columns.

INFO   (output) INTEGER
       = 0:  successful exit.
       < 0:  if INFO = -i, the i-th argument had an illegal value.
```

⟨*LAPACK dlasd2*⟩≡
```
(let* ((zero 0.0) (one 1.0) (two 2.0) (eight 8.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two)
           (type (double-float 8.0 8.0) eight))
  (defun dlasd2
         (nl nr sqre k d z alpha beta u ldu vt ldvt dsigma u2 ldu2 vt2 ldvt2
          idxp idx idxc idxq coltyp info)
    (declare (type (simple-array fixnum (*)) coltyp idxq idxc idx idxp)
             (type (double-float) beta alpha)
             (type (simple-array double-float (*)) vt2 u2 dsigma vt u z d)
             (type fixnum info ldvt2 ldu2 ldvt ldu k sqre nr nl))
    (f2cl-lib:with-multi-array-data
        ((d double-float d-%data% d-%offset%)
         (z double-float z-%data% z-%offset%)
         (u double-float u-%data% u-%offset%)
         (vt double-float vt-%data% vt-%offset%)
         (dsigma double-float dsigma-%data% dsigma-%offset%)
         (u2 double-float u2-%data% u2-%offset%)
         (vt2 double-float vt2-%data% vt2-%offset%)
         (idxp fixnum idxp-%data% idxp-%offset%)
         (idx fixnum idx-%data% idx-%offset%)
         (idxc fixnum idxc-%data% idxc-%offset%)
         (idxq fixnum idxq-%data% idxq-%offset%)
         (coltyp fixnum coltyp-%data% coltyp-%offset%))
      (prog ((c 0.0) (eps 0.0) (hlftol 0.0) (s 0.0) (tau 0.0) (tol 0.0)
             (z1 0.0) (ct 0) (i 0) (idxi 0) (idxj 0) (idxjp 0) (j 0) (jp 0)
             (jprev 0) (k2 0) (m 0) (n 0) (nlp1 0) (nlp2 0)
             (ctot (make-array 4 :element-type 'fixnum))
             (psm (make-array 4 :element-type 'fixnum)))
        (declare (type (double-float) c eps hlftol s tau tol z1)
                 (type fixnum ct i idxi idxj idxjp j jp jprev k2 m
                                    n nlp1 nlp2)
                 (type (simple-array fixnum (4)) ctot psm))
        (setf info 0)
        (cond
          ((< nl 1)
           (setf info -1))
          ((< nr 1)
           (setf info -2))
          ((and (/= sqre 1) (/= sqre 0))
           (setf info -3)))
        (setf n (f2cl-lib:int-add nl nr 1))
        (setf m (f2cl-lib:int-add n sqre))
        (cond
```

```
  ((< ldu n)
   (setf info -10))
  ((< ldvt m)
   (setf info -12))
  ((< ldu2 n)
   (setf info -15))
  ((< ldvt2 m)
   (setf info -17)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DLASD2" (f2cl-lib:int-sub info))
   (go end_label)))
(setf nlp1 (f2cl-lib:int-add nl 1))
(setf nlp2 (f2cl-lib:int-add nl 2))
(setf z1
        (* alpha
           (f2cl-lib:fref vt-%data%
                          (nlp1 nlp1)
                          ((1 ldvt) (1 *))
                          vt-%offset%)))
(setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) z1)
(f2cl-lib:fdo (i nl (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
              ((> i 1) nil)
  (tagbody
    (setf (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-add i 1))
                         ((1 *))
                         z-%offset%)
            (* alpha
               (f2cl-lib:fref vt-%data%
                              (i nlp1)
                              ((1 ldvt) (1 *))
                              vt-%offset%)))
    (setf (f2cl-lib:fref d-%data%
                         ((f2cl-lib:int-add i 1))
                         ((1 *))
                         d-%offset%)
            (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
    (setf (f2cl-lib:fref idxq-%data%
                         ((f2cl-lib:int-add i 1))
                         ((1 *))
                         idxq-%offset%)
            (f2cl-lib:int-add
             (f2cl-lib:fref idxq-%data% (i) ((1 *)) idxq-%offset%)
```

```
             1))))
(f2cl-lib:fdo (i nlp2 (f2cl-lib:int-add i 1))
              ((> i m) nil)
  (tagbody
    (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
            (* beta
               (f2cl-lib:fref vt-%data%
                              (i nlp2)
                              ((1 ldvt) (1 *))
                              vt-%offset%)))))
(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
              ((> i nlp1) nil)
  (tagbody
   (setf (f2cl-lib:fref coltyp-%data% (i) ((1 *)) coltyp-%offset%) 1)))
(f2cl-lib:fdo (i nlp2 (f2cl-lib:int-add i 1))
              ((> i n) nil)
  (tagbody
   (setf (f2cl-lib:fref coltyp-%data% (i) ((1 *)) coltyp-%offset%) 2)))
(f2cl-lib:fdo (i nlp2 (f2cl-lib:int-add i 1))
              ((> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref idxq-%data% (i) ((1 *)) idxq-%offset%)
            (f2cl-lib:int-add
             (f2cl-lib:fref idxq-%data% (i) ((1 *)) idxq-%offset%)
             nlp1))))
(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
              ((> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref dsigma-%data% (i) ((1 *)) dsigma-%offset%)
            (f2cl-lib:fref d-%data%
                           ((f2cl-lib:fref idxq (i) ((1 *))))
                           ((1 *))
                           d-%offset%))
    (setf (f2cl-lib:fref u2-%data% (i 1) ((1 ldu2) (1 *)) u2-%offset%)
            (f2cl-lib:fref z-%data%
                           ((f2cl-lib:fref idxq (i) ((1 *))))
                           ((1 *))
                           z-%offset%))
    (setf (f2cl-lib:fref idxc-%data% (i) ((1 *)) idxc-%offset%)
            (f2cl-lib:fref coltyp-%data%
                           ((f2cl-lib:fref idxq (i) ((1 *))))
                           ((1 *))
                           coltyp-%offset%))))
(dlamrg nl nr (f2cl-lib:array-slice dsigma double-float (2) ((1 *))) 1
 1 (f2cl-lib:array-slice idx fixnum (2) ((1 *))))
(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
```

```
                    ((> i n) nil)
          (tagbody
            (setf idxi
                    (f2cl-lib:int-add 1
                                      (f2cl-lib:fref idx-%data%
                                                     (i)
                                                     ((1 *))
                                                     idx-%offset%)))
            (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                    (f2cl-lib:fref dsigma-%data%
                                   (idxi)
                                   ((1 *))
                                   dsigma-%offset%))
            (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
                    (f2cl-lib:fref u2-%data%
                                   (idxi 1)
                                   ((1 ldu2) (1 *))
                                   u2-%offset%))
            (setf (f2cl-lib:fref coltyp-%data% (i) ((1 *)) coltyp-%offset%)
                    (f2cl-lib:fref idxc-%data% (idxi) ((1 *)) idxc-%offset%))))
        (setf eps (dlamch "Epsilon"))
        (setf tol (max (abs alpha) (abs beta)))
        (setf tol
                (* eight
                   eps
                   (max (abs (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%))
                        tol)))
        (setf k 1)
        (setf k2 (f2cl-lib:int-add n 1))
        (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                      ((> j n) nil)
          (tagbody
            (cond
              ((<= (abs (f2cl-lib:fref z (j) ((1 *)))) tol)
               (setf k2 (f2cl-lib:int-sub k2 1))
               (setf (f2cl-lib:fref idxp-%data% (k2) ((1 *)) idxp-%offset%) j)
               (setf (f2cl-lib:fref coltyp-%data% (j) ((1 *)) coltyp-%offset%)
                     4)
               (if (= j n) (go label120)))
              (t
               (setf jprev j)
               (go label90)))))
label90
        (setf j jprev)
label100
        (setf j (f2cl-lib:int-add j 1))
```

```
(if (> j n) (go label110))
(cond
  ((<= (abs (f2cl-lib:fref z (j) ((1 *)))) tol)
   (setf k2 (f2cl-lib:int-sub k2 1))
   (setf (f2cl-lib:fref idxp-%data% (k2) ((1 *)) idxp-%offset%) j)
   (setf (f2cl-lib:fref coltyp-%data% (j) ((1 *)) coltyp-%offset%) 4))
  (t
   (cond
     ((<=
       (abs
        (+ (f2cl-lib:fref d (j) ((1 *)))
           (- (f2cl-lib:fref d (jprev) ((1 *))))))
       tol)
      (setf s (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%))
      (setf c (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%))
      (setf tau (dlapy2 c s))
      (setf c (/ c tau))
      (setf s (/ (- s) tau))
      (setf (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%) tau)
      (setf (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%) zero)
      (setf idxjp
            (f2cl-lib:fref idxq-%data%
                           ((f2cl-lib:int-add
                             (f2cl-lib:fref idx (jprev) ((1 *)))
                             1))
                           ((1 *))
                           idxq-%offset%))
      (setf idxj
            (f2cl-lib:fref idxq-%data%
                           ((f2cl-lib:int-add
                             (f2cl-lib:fref idx (j) ((1 *)))
                             1))
                           ((1 *))
                           idxq-%offset%))
      (cond
        ((<= idxjp nlp1)
         (setf idxjp (f2cl-lib:int-sub idxjp 1))))
      (cond
        ((<= idxj nlp1)
         (setf idxj (f2cl-lib:int-sub idxj 1))))
      (drot n
       (f2cl-lib:array-slice u double-float (1 idxjp) ((1 ldu) (1 *)))
       1 (f2cl-lib:array-slice u double-float (1 idxj) ((1 ldu) (1 *)))
       1 c s)
      (drot m
       (f2cl-lib:array-slice vt
```

```
                                    double-float
                                    (idxjp 1)
                                    ((1 ldvt) (1 *)))
                 ldvt
                 (f2cl-lib:array-slice vt double-float (idxj 1) ((1 ldvt) (1 *)))
                 ldvt c s)
                (cond
                  ((/= (f2cl-lib:fref coltyp (j) ((1 *)))
                       (f2cl-lib:fref coltyp (jprev) ((1 *))))
                   (setf (f2cl-lib:fref coltyp-%data%
                                        (j)
                                        ((1 *))
                                        coltyp-%offset%)
                           3)))
                (setf (f2cl-lib:fref coltyp-%data%
                                     (jprev)
                                     ((1 *))
                                     coltyp-%offset%)
                        4)
                (setf k2 (f2cl-lib:int-sub k2 1))
                (setf (f2cl-lib:fref idxp-%data% (k2) ((1 *)) idxp-%offset%)
                        jprev)
                (setf jprev j))
               (t
                (setf k (f2cl-lib:int-add k 1))
                (setf (f2cl-lib:fref u2-%data%
                                     (k 1)
                                     ((1 ldu2) (1 *))
                                     u2-%offset%)
                        (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%))
                (setf (f2cl-lib:fref dsigma-%data% (k) ((1 *)) dsigma-%offset%)
                        (f2cl-lib:fref d-%data% (jprev) ((1 *)) d-%offset%))
                (setf (f2cl-lib:fref idxp-%data% (k) ((1 *)) idxp-%offset%) jprev)
                (setf jprev j)))))
       (go label100)
label110
       (setf k (f2cl-lib:int-add k 1))
       (setf (f2cl-lib:fref u2-%data% (k 1) ((1 ldu2) (1 *)) u2-%offset%)
               (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%))
       (setf (f2cl-lib:fref dsigma-%data% (k) ((1 *)) dsigma-%offset%)
               (f2cl-lib:fref d-%data% (jprev) ((1 *)) d-%offset%))
       (setf (f2cl-lib:fref idxp-%data% (k) ((1 *)) idxp-%offset%) jprev)
label120
       (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                     ((> j 4) nil)
         (tagbody (setf (f2cl-lib:fref ctot (j) ((1 4))) 0)))
```

```
(f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
              ((> j n) nil)
  (tagbody
    (setf ct (f2cl-lib:fref coltyp-%data% (j) ((1 *)) coltyp-%offset%))
    (setf (f2cl-lib:fref ctot (ct) ((1 4)))
            (f2cl-lib:int-add (f2cl-lib:fref ctot (ct) ((1 4))) 1))))
(setf (f2cl-lib:fref psm (1) ((1 4))) 2)
(setf (f2cl-lib:fref psm (2) ((1 4)))
        (f2cl-lib:int-add 2 (f2cl-lib:fref ctot (1) ((1 4)))))
(setf (f2cl-lib:fref psm (3) ((1 4)))
        (f2cl-lib:int-add (f2cl-lib:fref psm (2) ((1 4)))
                          (f2cl-lib:fref ctot (2) ((1 4)))))
(setf (f2cl-lib:fref psm (4) ((1 4)))
        (f2cl-lib:int-add (f2cl-lib:fref psm (3) ((1 4)))
                          (f2cl-lib:fref ctot (3) ((1 4)))))
(f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
              ((> j n) nil)
  (tagbody
    (setf jp (f2cl-lib:fref idxp-%data% (j) ((1 *)) idxp-%offset%))
    (setf ct
            (f2cl-lib:fref coltyp-%data% (jp) ((1 *)) coltyp-%offset%))
    (setf (f2cl-lib:fref idxc-%data%
                         ((f2cl-lib:fref psm (ct) ((1 4))))
                         ((1 *))
                         idxc-%offset%)
            j)
    (setf (f2cl-lib:fref psm (ct) ((1 4)))
            (f2cl-lib:int-add (f2cl-lib:fref psm (ct) ((1 4))) 1))))
(f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
              ((> j n) nil)
  (tagbody
    (setf jp (f2cl-lib:fref idxp-%data% (j) ((1 *)) idxp-%offset%))
    (setf (f2cl-lib:fref dsigma-%data% (j) ((1 *)) dsigma-%offset%)
            (f2cl-lib:fref d-%data% (jp) ((1 *)) d-%offset%))
    (setf idxj
            (f2cl-lib:fref idxq-%data%
             ((f2cl-lib:int-add
               (f2cl-lib:fref idx
                ((f2cl-lib:fref idxp
                  ((f2cl-lib:fref
                    idxc
                    (j)
                    ((1 *))))
                  ((1 *))))
                ((1 *)))
               1))
```

```
                ((1 *))
                idxq-%offset%))
      (cond
        ((<= idxj nlp1)
         (setf idxj (f2cl-lib:int-sub idxj 1))))
      (dcopy n
       (f2cl-lib:array-slice u double-float (1 idxj) ((1 ldu) (1 *))) 1
       (f2cl-lib:array-slice u2 double-float (1 j) ((1 ldu2) (1 *))) 1)
      (dcopy m
       (f2cl-lib:array-slice vt double-float (idxj 1) ((1 ldvt) (1 *)))
       ldvt
       (f2cl-lib:array-slice vt2 double-float (j 1) ((1 ldvt2) (1 *)))
       ldvt2)))
  (setf (f2cl-lib:fref dsigma-%data% (1) ((1 *)) dsigma-%offset%) zero)
  (setf hlftol (/ tol two))
  (if
   (<= (abs (f2cl-lib:fref dsigma-%data% (2) ((1 *)) dsigma-%offset%))
       hlftol)
   (setf (f2cl-lib:fref dsigma-%data% (2) ((1 *)) dsigma-%offset%)
           hlftol))
  (cond
    ((> m n)
     (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
             (dlapy2 z1 (f2cl-lib:fref z-%data% (m) ((1 *)) z-%offset%)))
     (cond
       ((<= (f2cl-lib:fref z (1) ((1 *))) tol)
        (setf c one)
        (setf s zero)
        (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) tol))
       (t
        (setf c (/ z1 (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)))
        (setf s
                (/ (f2cl-lib:fref z-%data% (m) ((1 *)) z-%offset%)
                   (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%))))))
    (t
     (cond
       ((<= (abs z1) tol)
        (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) tol))
       (t
        (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) z1)))))
  (dcopy (f2cl-lib:int-sub k 1)
   (f2cl-lib:array-slice u2 double-float (2 1) ((1 ldu2) (1 *))) 1
   (f2cl-lib:array-slice z double-float (2) ((1 *))) 1)
  (dlaset "A" n 1 zero zero u2 ldu2)
  (setf (f2cl-lib:fref u2-%data% (nlp1 1) ((1 ldu2) (1 *)) u2-%offset%)
          one)
```

```
(cond
  ((> m n)
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                 ((> i nlp1) nil)
     (tagbody
       (setf (f2cl-lib:fref vt-%data%
                            (m i)
                            ((1 ldvt) (1 *))
                            vt-%offset%)
             (* (- s)
                (f2cl-lib:fref vt-%data%
                               (nlp1 i)
                               ((1 ldvt) (1 *))
                               vt-%offset%)))
       (setf (f2cl-lib:fref vt2-%data%
                            (1 i)
                            ((1 ldvt2) (1 *))
                            vt2-%offset%)
             (* c
                (f2cl-lib:fref vt-%data%
                               (nlp1 i)
                               ((1 ldvt) (1 *))
                               vt-%offset%)))))
   (f2cl-lib:fdo (i nlp2 (f2cl-lib:int-add i 1))
                 ((> i m) nil)
     (tagbody
       (setf (f2cl-lib:fref vt2-%data%
                            (1 i)
                            ((1 ldvt2) (1 *))
                            vt2-%offset%)
             (* s
                (f2cl-lib:fref vt-%data%
                               (m i)
                               ((1 ldvt) (1 *))
                               vt-%offset%)))
       (setf (f2cl-lib:fref vt-%data%
                            (m i)
                            ((1 ldvt) (1 *))
                            vt-%offset%)
             (* c
                (f2cl-lib:fref vt-%data%
                               (m i)
                               ((1 ldvt) (1 *))
                               vt-%offset%))))))
  (t
   (dcopy m
```

```
            (f2cl-lib:array-slice vt double-float (nlp1 1) ((1 ldvt) (1 *)))
            ldvt
            (f2cl-lib:array-slice vt2 double-float (1 1) ((1 ldvt2) (1 *)))
            ldvt2)))
          (cond
            ((> m n)
             (dcopy m
              (f2cl-lib:array-slice vt double-float (m 1) ((1 ldvt) (1 *))) ldvt
              (f2cl-lib:array-slice vt2 double-float (m 1) ((1 ldvt2) (1 *)))
              ldvt2)))
          (cond
            ((> n k)
             (dcopy (f2cl-lib:int-sub n k)
              (f2cl-lib:array-slice dsigma double-float ((+ k 1)) ((1 *))) 1
              (f2cl-lib:array-slice d double-float ((+ k 1)) ((1 *))) 1)
             (dlacpy "A" n (f2cl-lib:int-sub n k)
              (f2cl-lib:array-slice u2
                                    double-float
                                    (1 (f2cl-lib:int-add k 1))
                                    ((1 ldu2) (1 *)))
              ldu2
              (f2cl-lib:array-slice u
                                    double-float
                                    (1 (f2cl-lib:int-add k 1))
                                    ((1 ldu) (1 *)))
              ldu)
             (dlacpy "A" (f2cl-lib:int-sub n k) m
              (f2cl-lib:array-slice vt2
                                    double-float
                                    ((+ k 1) 1)
                                    ((1 ldvt2) (1 *)))
              ldvt2
              (f2cl-lib:array-slice vt double-float ((+ k 1) 1) ((1 ldvt) (1 *)))
              ldvt)))
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                        ((> j 4) nil)
            (tagbody
              (setf (f2cl-lib:fref coltyp-%data% (j) ((1 *)) coltyp-%offset%)
                      (f2cl-lib:fref ctot (j) ((1 4))))))
  end_label
          (return
           (values nil
                   nil
                   nil
                   k
                   nil
```

```
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  info))))))
```

7.57 dlasd3 LAPACK

⟨*dlasd3.input*⟩≡

```
)set break resume
)sys rm -f dlasd3.output
)spool dlasd3.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasd3.help*⟩≡

```
==========================================================================
dlasd3 examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

NAME
       DLASD3  - all the square roots of the roots of the secular equation, as
       defined by the values in D and Z

SYNOPSIS
       SUBROUTINE DLASD3( NL, NR, SQRE, K, D, Q,  LDQ,  DSIGMA,  U,  LDU,  U2,
                          LDU2, VT, LDVT, VT2, LDVT2, IDXC, CTOT, Z, INFO )

           INTEGER        INFO, K, LDQ, LDU, LDU2, LDVT, LDVT2, NL, NR, SQRE

           INTEGER        CTOT( * ), IDXC( * )

           DOUBLE         PRECISION  D( * ), DSIGMA( * ), Q( LDQ, * ), U( LDU,
                          * ), U2( LDU2, * ), VT( LDVT, * ), VT2( LDVT2, *  ),
                          Z( * )

PURPOSE
       DLASD3 finds all the square roots of the roots of the secular equation,
       as defined by the values in D and Z.  It makes the appropriate calls to
       DLASD4  and then updates the singular vectors by matrix multiplication.

       This code makes very mild assumptions about floating point  arithmetic.
       It  will  work  on  machines  with a guard digit in add/subtract, or on
       those binary machines without guard digits which subtract like the Cray
       XMP, Cray YMP, Cray C 90, or Cray 2.  It could conceivably fail on hex-
       adecimal or decimal machines without guard digits, but we know of none.

       DLASD3 is called from DLASD1.


ARGUMENTS
       NL     (input) INTEGER
              The row dimension of the upper block.  NL >= 1.

       NR     (input) INTEGER
              The row dimension of the lower block.  NR >= 1.
```

```
SQRE   (input) INTEGER
       = 0: the lower block is an NR-by-NR square matrix.
       = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

       The  bidiagonal matrix has N = NL + NR + 1 rows and M = N + SQRE
       >= N columns.

K      (input) INTEGER
       The size of the secular equation, 1 =< K = < N.

D      (output) DOUBLE PRECISION array, dimension(K)
       On exit the square roots of the roots of the  secular  equation,
       in ascending order.

Q      (workspace) DOUBLE PRECISION array,
       dimension at least (LDQ,K).

LDQ    (input) INTEGER
       The leading dimension of the array Q.  LDQ >= K.

       DSIGMA  (input) DOUBLE PRECISION array, dimension(K) The first K
       elements of this array contain the old  roots  of  the  deflated
       updating  problem.  These are the poles of the secular equation.

U      (output) DOUBLE PRECISION array, dimension (LDU, N)
       The last N - K columns of this matrix contain the deflated  left
       singular vectors.

LDU    (input) INTEGER
       The leading dimension of the array U.  LDU >= N.

U2     (input/output) DOUBLE PRECISION array, dimension (LDU2, N)
       The first K columns of this matrix contain the non-deflated left
       singular vectors for the split problem.

LDU2   (input) INTEGER
       The leading dimension of the array U2.  LDU2 >= N.

VT     (output) DOUBLE PRECISION array, dimension (LDVT, M)
       The last M - K columns of VT' contain the deflated right  singu-
       lar vectors.

LDVT   (input) INTEGER
       The leading dimension of the array VT.  LDVT >= N.

VT2    (input/output) DOUBLE PRECISION array, dimension (LDVT2, N)
```

```
          The  first K columns of VT2' contain the non-deflated right sin-
          gular vectors for the split problem.

LDVT2     (input) INTEGER
          The leading dimension of the array VT2.  LDVT2 >= N.

IDXC      (input) INTEGER array, dimension ( N )
          The permutation used to arrange the columns of U  (and  rows  of
          VT)  into  three  groups:  the  first  group  contains non-zero
          entries only at and above (or before) NL +1; the second contains
          non-zero  entries  only  at  and  below (or after) NL+2; and the
          third is dense. The first column of U and  the  row  of  VT  are
          treated separately, however.

          The  rows  of the singular vectors found by DLASD4 must be like-
          wise permuted before the matrix multiplies can take place.

CTOT      (input) INTEGER array, dimension ( 4 )
          A count of the total number of the various types of columns in U
          (or rows in VT), as described in IDXC. The fourth column type is
          any column which has been deflated.

Z         (input) DOUBLE PRECISION array, dimension (K)
          The first K elements of this array contain the components of the
          deflation-adjusted updating row vector.

INFO      (output) INTEGER
          = 0:  successful exit.
          < 0:  if INFO = -i, the i-th argument had an illegal value.
          > 0:  if INFO = 1, an singular value did not converge
```

⟨*LAPACK dlasd3*⟩≡
```
(let* ((one 1.0) (zero 0.0) (negone (- 1.0)))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero)
           (type (double-float) negone))
  (defun dlasd3
         (nl nr sqre k d q ldq dsigma u ldu u2 ldu2 vt ldvt vt2 ldvt2 idxc ctot
          z info)
    (declare (type (simple-array fixnum (*)) ctot idxc)
             (type (simple-array double-float (*)) z vt2 vt u2 u dsigma q d)
             (type fixnum info ldvt2 ldvt ldu2 ldu ldq k sqre nr
                                  nl))
    (f2cl-lib:with-multi-array-data
        ((d double-float d-%data% d-%offset%)
         (q double-float q-%data% q-%offset%)
         (dsigma double-float dsigma-%data% dsigma-%offset%)
         (u double-float u-%data% u-%offset%)
         (u2 double-float u2-%data% u2-%offset%)
         (vt double-float vt-%data% vt-%offset%)
         (vt2 double-float vt2-%data% vt2-%offset%)
         (z double-float z-%data% z-%offset%)
         (idxc fixnum idxc-%data% idxc-%offset%)
         (ctot fixnum ctot-%data% ctot-%offset%))
      (prog ((rho 0.0) (temp 0.0) (ctemp 0) (i 0) (j 0) (jc 0) (ktemp 0) (m 0)
             (n 0) (nlp1 0) (nlp2 0) (nrp1 0))
        (declare (type (double-float) rho temp)
                 (type fixnum ctemp i j jc ktemp m n nlp1 nlp2
                                      nrp1))
        (setf info 0)
        (cond
          ((< nl 1)
           (setf info -1))
          ((< nr 1)
           (setf info -2))
          ((and (/= sqre 1) (/= sqre 0))
           (setf info -3)))
        (setf n (f2cl-lib:int-add nl nr 1))
        (setf m (f2cl-lib:int-add n sqre))
        (setf nlp1 (f2cl-lib:int-add nl 1))
        (setf nlp2 (f2cl-lib:int-add nl 2))
        (cond
          ((or (< k 1) (> k n))
           (setf info -4))
          ((< ldq k)
           (setf info -7))
          ((< ldu n)
```

```
   (setf info -10))
  ((< ldu2 n)
   (setf info -12))
  ((< ldvt m)
   (setf info -14))
  ((< ldvt2 m)
   (setf info -16)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DLASD3" (f2cl-lib:int-sub info))
   (go end_label)))
(cond
  ((= k 1)
   (setf (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)
           (abs (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)))
   (dcopy m
    (f2cl-lib:array-slice vt2 double-float (1 1) ((1 ldvt2) (1 *)))
    ldvt2 (f2cl-lib:array-slice vt double-float (1 1) ((1 ldvt) (1 *)))
    ldvt)
   (cond
     ((> (f2cl-lib:fref z (1) ((1 *))) zero)
      (dcopy n
       (f2cl-lib:array-slice u2 double-float (1 1) ((1 ldu2) (1 *))) 1
       (f2cl-lib:array-slice u double-float (1 1) ((1 ldu) (1 *))) 1))
     (t
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i n) nil)
        (tagbody
          (setf (f2cl-lib:fref u-%data%
                               (i 1)
                               ((1 ldu) (1 *))
                               u-%offset%)
                  (-
                   (f2cl-lib:fref u2-%data%
                                  (i 1)
                                  ((1 ldu2) (1 *))
                                  u2-%offset%)))))))
   (go end_label)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i k) nil)
  (tagbody
    (setf (f2cl-lib:fref dsigma-%data% (i) ((1 *)) dsigma-%offset%)
            (-
             (multiple-value-bind (ret-val var-0 var-1)
```

```
                    (dlamc3
                     (f2cl-lib:fref dsigma-%data%
                                    (i)
                                    ((1 *))
                                    dsigma-%offset%)
                     (f2cl-lib:fref dsigma-%data%
                                    (i)
                                    ((1 *))
                                    dsigma-%offset%))
                    (declare (ignore))
                    (setf (f2cl-lib:fref dsigma-%data%
                                         (i)
                                         ((1 *))
                                         dsigma-%offset%)
                          var-0)
                    (setf (f2cl-lib:fref dsigma-%data%
                                         (i)
                                         ((1 *))
                                         dsigma-%offset%)
                          var-1)
                    ret-val)
                  (f2cl-lib:fref dsigma-%data%
                                 (i)
                                 ((1 *))
                                 dsigma-%offset%)))))
(dcopy k z 1 q 1)
(setf rho (dnrm2 k z 1))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
    (dlascl "G" 0 0 rho one k 1 z k info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                   var-8))
  (setf info var-9))
(setf rho (* rho rho))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j k) nil)
  (tagbody
    (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
        (dlasd4 k j dsigma z
         (f2cl-lib:array-slice u double-float (1 j) ((1 ldu) (1 *)))
         rho (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
         (f2cl-lib:array-slice vt double-float (1 j) ((1 ldvt) (1 *)))
         info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-7))
      (setf (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%) var-6)
```

```
      (setf info var-8))
    (cond
      ((/= info 0)
       (go end_label)))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i k) nil)
  (tagbody
    (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
            (*
             (f2cl-lib:fref u-%data% (i k) ((1 ldu) (1 *)) u-%offset%)
             (f2cl-lib:fref vt-%data%
                            (i k)
                            ((1 ldvt) (1 *))
                            vt-%offset%)))
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  ((> j (f2cl-lib:int-add i (f2cl-lib:int-sub 1))) nil)
      (tagbody
        (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
                (* (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
                   (/
                    (/
                     (*
                      (f2cl-lib:fref u-%data%
                                     (i j)
                                     ((1 ldu) (1 *))
                                     u-%offset%)
                      (f2cl-lib:fref vt-%data%
                                     (i j)
                                     ((1 ldvt) (1 *))
                                     vt-%offset%))
                     (-
                      (f2cl-lib:fref dsigma-%data%
                                     (i)
                                     ((1 *))
                                     dsigma-%offset%)
                      (f2cl-lib:fref dsigma-%data%
                                     (j)
                                     ((1 *))
                                     dsigma-%offset%)))
                    (+
                     (f2cl-lib:fref dsigma-%data%
                                    (i)
                                    ((1 *))
                                    dsigma-%offset%)
                     (f2cl-lib:fref dsigma-%data%
                                    (j)
```

```
                                        ((1 *))
                                        dsigma-%offset%)))))))
            (f2cl-lib:fdo (j i (f2cl-lib:int-add j 1))
                          ((> j (f2cl-lib:int-add k (f2cl-lib:int-sub 1))) nil)
              (tagbody
                (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
                        (* (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
                           (/
                            (/
                             (*
                              (f2cl-lib:fref u-%data%
                                             (i j)
                                             ((1 ldu) (1 *))
                                             u-%offset%)
                              (f2cl-lib:fref vt-%data%
                                             (i j)
                                             ((1 ldvt) (1 *))
                                             vt-%offset%))
                             (-
                              (f2cl-lib:fref dsigma-%data%
                                             (i)
                                             ((1 *))
                                             dsigma-%offset%)
                              (f2cl-lib:fref dsigma-%data%
                                             ((f2cl-lib:int-add j 1))
                                             ((1 *))
                                             dsigma-%offset%)))
                            (+
                             (f2cl-lib:fref dsigma-%data%
                                            (i)
                                            ((1 *))
                                            dsigma-%offset%)
                             (f2cl-lib:fref dsigma-%data%
                                            ((f2cl-lib:int-add j 1))
                                            ((1 *))
                                            dsigma-%offset%)))))))
            (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
                    (f2cl-lib:sign
                     (f2cl-lib:fsqrt
                      (abs (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)))
                     (f2cl-lib:fref q-%data%
                                    (i 1)
                                    ((1 ldq) (1 *))
                                    q-%offset%)))))
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i k) nil)
```

```
(tagbody
  (setf (f2cl-lib:fref vt-%data% (1 i) ((1 ldvt) (1 *)) vt-%offset%)
        (/
         (/ (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
            (f2cl-lib:fref u-%data%
                           (1 i)
                           ((1 ldu) (1 *))
                           u-%offset%))
         (f2cl-lib:fref vt-%data%
                        (1 i)
                        ((1 ldvt) (1 *))
                        vt-%offset%)))
  (setf (f2cl-lib:fref u-%data% (1 i) ((1 ldu) (1 *)) u-%offset%)
        negone)
  (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                ((> j k) nil)
    (tagbody
      (setf (f2cl-lib:fref vt-%data%
                           (j i)
                           ((1 ldvt) (1 *))
                           vt-%offset%)
            (/
             (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
                (f2cl-lib:fref u-%data%
                               (j i)
                               ((1 ldu) (1 *))
                               u-%offset%))
             (f2cl-lib:fref vt-%data%
                            (j i)
                            ((1 ldvt) (1 *))
                            vt-%offset%)))
      (setf (f2cl-lib:fref u-%data% (j i) ((1 ldu) (1 *)) u-%offset%)
            (*
             (f2cl-lib:fref dsigma-%data%
                            (j)
                            ((1 *))
                            dsigma-%offset%)
             (f2cl-lib:fref vt-%data%
                            (j i)
                            ((1 ldvt) (1 *))
                            vt-%offset%)))))
  (setf temp
        (dnrm2 k
         (f2cl-lib:array-slice u
                               double-float
                               (1 i)
```

```
                                        ((1 ldu) (1 *)))
                1))
        (setf (f2cl-lib:fref q-%data% (1 i) ((1 ldq) (1 *)) q-%offset%)
              (/
               (f2cl-lib:fref u-%data% (1 i) ((1 ldu) (1 *)) u-%offset%)
               temp))
        (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                      ((> j k) nil)
          (tagbody
            (setf jc (f2cl-lib:fref idxc-%data% (j) ((1 *)) idxc-%offset%))
            (setf (f2cl-lib:fref q-%data% (j i) ((1 ldq) (1 *)) q-%offset%)
                  (/
                   (f2cl-lib:fref u-%data%
                                  (jc i)
                                  ((1 ldu) (1 *))
                                  u-%offset%)
                   temp))))))
  (cond
    ((= k 2)
     (dgemm "N" "N" n k k one u2 ldu2 q ldq zero u ldu)
     (go label100)))
  (cond
    ((> (f2cl-lib:fref ctot (1) ((1 *))) 0)
     (dgemm "N" "N" nl k
      (f2cl-lib:fref ctot-%data% (1) ((1 *)) ctot-%offset%) one
      (f2cl-lib:array-slice u2 double-float (1 2) ((1 ldu2) (1 *))) ldu2
      (f2cl-lib:array-slice q double-float (2 1) ((1 ldq) (1 *))) ldq
      zero (f2cl-lib:array-slice u double-float (1 1) ((1 ldu) (1 *)))
      ldu)
     (cond
       ((> (f2cl-lib:fref ctot (3) ((1 *))) 0)
        (setf ktemp
              (f2cl-lib:int-add 2
                                (f2cl-lib:fref ctot-%data%
                                               (1)
                                               ((1 *))
                                               ctot-%offset%)
                                (f2cl-lib:fref ctot-%data%
                                               (2)
                                               ((1 *))
                                               ctot-%offset%)))
        (dgemm "N" "N" nl k
         (f2cl-lib:fref ctot-%data% (3) ((1 *)) ctot-%offset%) one
         (f2cl-lib:array-slice u2
                               double-float
                               (1 ktemp)
```

```
                                    ((1 ldu2) (1 *)))
                ldu2
                (f2cl-lib:array-slice q double-float (ktemp 1) ((1 ldq) (1 *)))
                ldq one
                (f2cl-lib:array-slice u double-float (1 1) ((1 ldu) (1 *)))
                ldu))))
            ((> (f2cl-lib:fref ctot (3) ((1 *))) 0)
             (setf ktemp
                   (f2cl-lib:int-add 2
                                     (f2cl-lib:fref ctot-%data%
                                                    (1)
                                                    ((1 *))
                                                    ctot-%offset%)
                                     (f2cl-lib:fref ctot-%data%
                                                    (2)
                                                    ((1 *))
                                                    ctot-%offset%)))
             (dgemm "N" "N" nl k
              (f2cl-lib:fref ctot-%data% (3) ((1 *)) ctot-%offset%) one
              (f2cl-lib:array-slice u2 double-float (1 ktemp) ((1 ldu2) (1 *)))
              ldu2
              (f2cl-lib:array-slice q double-float (ktemp 1) ((1 ldq) (1 *))) ldq
              zero (f2cl-lib:array-slice u double-float (1 1) ((1 ldu) (1 *)))
              ldu))
            (t
             (dlacpy "F" nl k u2 ldu2 u ldu)))
           (dcopy k (f2cl-lib:array-slice q double-float (1 1) ((1 ldq) (1 *)))
            ldq (f2cl-lib:array-slice u double-float (nlp1 1) ((1 ldu) (1 *)))
            ldu)
           (setf ktemp
                   (f2cl-lib:int-add 2
                                     (f2cl-lib:fref ctot-%data%
                                                    (1)
                                                    ((1 *))
                                                    ctot-%offset%)))
           (setf ctemp
                   (f2cl-lib:int-add
                    (f2cl-lib:fref ctot-%data% (2) ((1 *)) ctot-%offset%)
                    (f2cl-lib:fref ctot-%data% (3) ((1 *)) ctot-%offset%)))
           (dgemm "N" "N" nr k ctemp one
            (f2cl-lib:array-slice u2 double-float (nlp2 ktemp) ((1 ldu2) (1 *)))
            ldu2 (f2cl-lib:array-slice q double-float (ktemp 1) ((1 ldq) (1 *)))
            ldq zero
            (f2cl-lib:array-slice u double-float (nlp2 1) ((1 ldu) (1 *))) ldu)
     label100
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
```

```
                ((> i k) nil)
          (tagbody
            (setf temp
                  (dnrm2 k
                   (f2cl-lib:array-slice vt
                                         double-float
                                         (1 i)
                                         ((1 ldvt) (1 *)))
                   1))
            (setf (f2cl-lib:fref q-%data% (i 1) ((1 ldq) (1 *)) q-%offset%)
                  (/
                   (f2cl-lib:fref vt-%data%
                                  (1 i)
                                  ((1 ldvt) (1 *))
                                  vt-%offset%)
                   temp))
            (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                          ((> j k) nil)
              (tagbody
                (setf jc (f2cl-lib:fref idxc-%data% (j) ((1 *)) idxc-%offset%))
                (setf (f2cl-lib:fref q-%data% (i j) ((1 ldq) (1 *)) q-%offset%)
                        (/
                         (f2cl-lib:fref vt-%data%
                                        (jc i)
                                        ((1 ldvt) (1 *))
                                        vt-%offset%)
                         temp))))))
        (cond
          ((= k 2)
           (dgemm "N" "N" k m k one q ldq vt2 ldvt2 zero vt ldvt)
           (go end_label)))
        (setf ktemp
                (f2cl-lib:int-add 1
                                  (f2cl-lib:fref ctot-%data%
                                                 (1)
                                                 ((1 *))
                                                 ctot-%offset%)))
        (dgemm "N" "N" k nlp1 ktemp one
         (f2cl-lib:array-slice q double-float (1 1) ((1 ldq) (1 *))) ldq
         (f2cl-lib:array-slice vt2 double-float (1 1) ((1 ldvt2) (1 *))) ldvt2
         zero (f2cl-lib:array-slice vt double-float (1 1) ((1 ldvt) (1 *)))
         ldvt)
        (setf ktemp
                (f2cl-lib:int-add 2
                                  (f2cl-lib:fref ctot-%data%
                                                 (1)
```

```
                                                ((1 *))
                                                ctot-%offset%)
                                    (f2cl-lib:fref ctot-%data%
                                                (2)
                                                ((1 *))
                                                ctot-%offset%)))
          (if (<= ktemp ldvt2)
              (dgemm "N" "N" k nlp1
               (f2cl-lib:fref ctot-%data% (3) ((1 *)) ctot-%offset%) one
               (f2cl-lib:array-slice q double-float (1 ktemp) ((1 ldq) (1 *)))
               ldq
               (f2cl-lib:array-slice vt2
                                     double-float
                                     (ktemp 1)
                                     ((1 ldvt2) (1 *)))
               ldvt2 one
               (f2cl-lib:array-slice vt double-float (1 1) ((1 ldvt) (1 *)))
               ldvt))
          (setf ktemp
                (f2cl-lib:int-add
                 (f2cl-lib:fref ctot-%data% (1) ((1 *)) ctot-%offset%)
                 1))
          (setf nrp1 (f2cl-lib:int-add nr sqre))
          (cond
            ((> ktemp 1)
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i k) nil)
               (tagbody
                 (setf (f2cl-lib:fref q-%data%
                                      (i ktemp)
                                      ((1 ldq) (1 *))
                                      q-%offset%)
                       (f2cl-lib:fref q-%data%
                                      (i 1)
                                      ((1 ldq) (1 *))
                                      q-%offset%))))
             (f2cl-lib:fdo (i nlp2 (f2cl-lib:int-add i 1))
                           ((> i m) nil)
               (tagbody
                 (setf (f2cl-lib:fref vt2-%data%
                                      (ktemp i)
                                      ((1 ldvt2) (1 *))
                                      vt2-%offset%)
                       (f2cl-lib:fref vt2-%data%
                                      (1 i)
                                      ((1 ldvt2) (1 *))
```

```
                                    vt2-%offset%))))))
        (setf ctemp
                (f2cl-lib:int-add 1
                                  (f2cl-lib:fref ctot-%data%
                                                 (2)
                                                 ((1 *))
                                                 ctot-%offset%)
                                  (f2cl-lib:fref ctot-%data%
                                                 (3)
                                                 ((1 *))
                                                 ctot-%offset%)))
        (dgemm "N" "N" k nrp1 ctemp one
         (f2cl-lib:array-slice q double-float (1 ktemp) ((1 ldq) (1 *))) ldq
         (f2cl-lib:array-slice vt2 double-float (ktemp nlp2) ((1 ldvt2) (1 *)))
         ldvt2 zero
         (f2cl-lib:array-slice vt double-float (1 nlp2) ((1 ldvt) (1 *))) ldvt)
  end_label
        (return
         (values nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 info))))))
```

7.58 dlasd4 LAPACK

⟨*dlasd4.input*⟩≡

```
)set break resume
)sys rm -f dlasd4.output
)spool dlasd4.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasd4.help*⟩≡

```
  ======================================================================
  dlasd4 examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLASD4  -  compute  the square root of the I-th updated eigenvalue of a
         positive symmetric rank-one modification to a positive diagonal  matrix
         whose  entries are given as the squares of the corresponding entries in
         the array d, and that   0 <= D(i) < D(j) for i < j  and that RHO > 0

  SYNOPSIS
         SUBROUTINE DLASD4( N, I, D, Z, DELTA, RHO, SIGMA, WORK, INFO )

             INTEGER        I, INFO, N

             DOUBLE         PRECISION RHO, SIGMA

             DOUBLE         PRECISION D( * ), DELTA( * ), WORK( * ), Z( * )

  PURPOSE
         This subroutine computes the square root of the I-th updated eigenvalue
         of  a  positive  symmetric rank-one modification to a positive diagonal
         matrix whose entries are given as  the  squares  of  the  corresponding
         entries  in  the array d, and that no loss in generality.  The rank-one
         modified system is thus

                diag( D ) * diag( D ) +  RHO *  Z * Z_transpose.

         where we assume the Euclidean norm of Z is 1.

         The method consists of approximating the rational functions in the sec-
         ular equation by simpler interpolating rational functions.


  ARGUMENTS
         N      (input) INTEGER
                The length of all arrays.

         I      (input) INTEGER
                The index of the eigenvalue to be computed.  1 <= I <= N.
```

```
        D      (input) DOUBLE PRECISION array, dimension ( N )
               The original eigenvalues.  It is assumed that they are in order,
               0 <= D(I) < D(J)  for I < J.

        Z      (input) DOUBLE PRECISION array, dimension ( N )
               The components of the updating vector.

        DELTA  (output) DOUBLE PRECISION array, dimension ( N )
               If N .ne. 1, DELTA contains (D(j) - sigma_I) in its  j-th compo-
               nent.   If  N = 1, then DELTA(1) = 1.  The vector DELTA contains
               the information necessary to construct the (singular)  eigenvec-
               tors.

        RHO    (input) DOUBLE PRECISION
               The scalar in the symmetric updating formula.

        SIGMA  (output) DOUBLE PRECISION
               The computed sigma_I, the I-th updated eigenvalue.

        WORK   (workspace) DOUBLE PRECISION array, dimension ( N )
               If  N .ne. 1, WORK contains (D(j) + sigma_I) in its  j-th compo-
               nent.  If N = 1, then WORK( 1 ) = 1.

        INFO   (output) INTEGER
               = 0:  successful exit
               > 0:  if INFO = 1, the updating process failed.

    PARAMETERS
           Logical variable  ORGATI  (origin-at-i?)  is  used  for  distinguishing
           whether D(i) or D(i+1) is treated as the origin.

           ORGATI = .true.    origin at i ORGATI = .false.   origin at i+1

           Logical  variable  SWTCH3 (switch-for-3-poles?) is for noting if we are
           working with THREE poles!

           MAXIT is the maximum number of iterations allowed for each  eigenvalue.
```

⟨*LAPACK dlasd4*⟩≡
```
(let* ((maxit 20)
       (zero 0.0)
       (one 1.0)
       (two 2.0)
       (three 3.0)
       (four 4.0)
       (eight 8.0)
       (ten 10.0))
  (declare (type (fixnum 20 20) maxit)
           (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two)
           (type (double-float 3.0 3.0) three)
           (type (double-float 4.0 4.0) four)
           (type (double-float 8.0 8.0) eight)
           (type (double-float 10.0 10.0) ten))
  (defun dlasd4 (n i d z delta rho sigma work info)
    (declare (type (double-float) sigma rho)
             (type (simple-array double-float (*)) work delta z d)
             (type fixnum info i n))
    (f2cl-lib:with-multi-array-data
        ((d double-float d-%data% d-%offset%)
         (z double-float z-%data% z-%offset%)
         (delta double-float delta-%data% delta-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((dd (make-array 3 :element-type 'double-float))
             (zz (make-array 3 :element-type 'double-float)) (a 0.0) (b 0.0)
             (c 0.0) (delsq 0.0) (delsq2 0.0) (dphi 0.0) (dpsi 0.0) (dtiim 0.0)
             (dtiip 0.0) (dtipsq 0.0) (dtisq 0.0) (dtnsq 0.0) (dtnsq1 0.0)
             (dw 0.0) (eps 0.0) (erretm 0.0) (eta 0.0) (phi 0.0) (prew 0.0)
             (psi 0.0) (rhoinv 0.0) (sg2lb 0.0) (sg2ub 0.0) (tau 0.0)
             (temp 0.0) (temp1 0.0) (temp2 0.0) (w 0.0) (ii 0) (iim1 0)
             (iip1 0) (ip1 0) (iter 0) (j 0) (niter 0) (orgati nil) (swtch nil)
             (swtch3 nil))
        (declare (type (simple-array double-float (3)) dd zz)
                 (type (double-float) a b c delsq delsq2 dphi dpsi dtiim dtiip
                                      dtipsq dtisq dtnsq dtnsq1 dw eps erretm
                                      eta phi prew psi rhoinv sg2lb sg2ub tau
                                      temp temp1 temp2 w)
                 (type fixnum ii iim1 iip1 ip1 iter j niter)
                 (type (member t nil) orgati swtch swtch3))
        (setf info 0)
        (cond
          ((= n 1)
           (setf sigma
```

```
                (f2cl-lib:fsqrt
                 (+
                  (* (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)
                     (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))
                  (* rho
                     (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
                     (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)))))
     (setf (f2cl-lib:fref delta-%data% (1) ((1 *)) delta-%offset%) one)
     (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) one)
     (go end_label)))
  (cond
    ((= n 2)
     (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
         (dlasd5 i d z delta rho sigma work)
       (declare (ignore var-0 var-1 var-2 var-3 var-4 var-6))
       (setf sigma var-5))
     (go end_label)))
  (setf eps (dlamch "Epsilon"))
  (setf rhoinv (/ one rho))
  (cond
    ((= i n)
     (setf ii (f2cl-lib:int-sub n 1))
     (setf niter 1)
     (setf temp (/ rho two))
     (setf temp1
             (/ temp
                (+ (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
                   (f2cl-lib:fsqrt
                    (+
                     (* (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
                        (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%))
                     temp)))))
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   ((> j n) nil)
       (tagbody
         (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
                 (+ (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
                    (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
                    temp1))
         (setf (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
                 (- (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
                    (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
                    temp1))))
     (setf psi zero)
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   ((> j (f2cl-lib:int-add n (f2cl-lib:int-sub 2))) nil)
```

```
      (tagbody
        (setf psi
                (+ psi
                   (/
                    (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
                       (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%))
                    (*
                     (f2cl-lib:fref delta-%data%
                                    (j)
                                    ((1 *))
                                    delta-%offset%)
                     (f2cl-lib:fref work-%data%
                                    (j)
                                    ((1 *))
                                    work-%offset%)))))))
    (setf c (+ rhoinv psi))
    (setf w
            (+ c
               (/
                (* (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)
                   (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%))
                (*
                 (f2cl-lib:fref delta-%data%
                                (ii)
                                ((1 *))
                                delta-%offset%)
                 (f2cl-lib:fref work-%data%
                                (ii)
                                ((1 *))
                                work-%offset%)))
               (/
                (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
                   (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%))
                (*
                 (f2cl-lib:fref delta-%data% (n) ((1 *)) delta-%offset%)
                 (f2cl-lib:fref work-%data%
                                (n)
                                ((1 *))
                                work-%offset%)))))
    (cond
      ((<= w zero)
       (setf temp1
               (f2cl-lib:fsqrt
                (+
                 (* (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
                    (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%))
```

```
          rho)))
(setf temp
       (+
        (/
         (*
          (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub n 1))
                         ((1 *))
                         z-%offset%)
          (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub n 1))
                         ((1 *))
                         z-%offset%))
         (*
          (+
           (f2cl-lib:fref d-%data%
                          ((f2cl-lib:int-sub n 1))
                          ((1 *))
                          d-%offset%)
           temp1)
          (+
           (- (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
              (f2cl-lib:fref d-%data%
                             ((f2cl-lib:int-sub n 1))
                             ((1 *))
                             d-%offset%))
           (/ rho
              (+ (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
                 temp1)))))
        (/
         (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
            (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%))
         rho)))
(cond
  ((<= c temp)
   (setf tau rho))
  (t
   (setf delsq
           (*
            (- (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
               (f2cl-lib:fref d-%data%
                              ((f2cl-lib:int-sub n 1))
                              ((1 *))
                              d-%offset%))
            (+ (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
               (f2cl-lib:fref d-%data%
```

```
                                    ((f2cl-lib:int-sub n 1))
                                    ((1 *))
                                    d-%offset%))))
            (setf a
                    (+ (* (- c) delsq)
                       (*
                        (f2cl-lib:fref z-%data%
                                       ((f2cl-lib:int-sub n 1))
                                       ((1 *))
                                       z-%offset%)
                        (f2cl-lib:fref z-%data%
                                       ((f2cl-lib:int-sub n 1))
                                       ((1 *))
                                       z-%offset%))
                       (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
                          (f2cl-lib:fref z-%data%
                                         (n)
                                         ((1 *))
                                         z-%offset%))))
            (setf b
                    (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
                       (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
                       delsq))
            (cond
              ((< a zero)
               (setf tau
                       (/ (* two b)
                          (- (f2cl-lib:fsqrt (+ (* a a) (* four b c)))
                             a))))
              (t
               (setf tau
                       (/ (+ a (f2cl-lib:fsqrt (+ (* a a) (* four b c))))
                          (* two c))))))))
          (t
           (setf delsq
                   (*
                    (- (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
                       (f2cl-lib:fref d-%data%
                                      ((f2cl-lib:int-sub n 1))
                                      ((1 *))
                                      d-%offset%))
                    (+ (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
                       (f2cl-lib:fref d-%data%
                                      ((f2cl-lib:int-sub n 1))
                                      ((1 *))
                                      d-%offset%))))
```

```
   (setf a
           (+ (* (- c) delsq)
              (*
               (f2cl-lib:fref z-%data%
                              ((f2cl-lib:int-sub n 1))
                              ((1 *))
                              z-%offset%)
               (f2cl-lib:fref z-%data%
                              ((f2cl-lib:int-sub n 1))
                              ((1 *))
                              z-%offset%))
              (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
                 (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%))))
   (setf b
           (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
              (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
              delsq))
   (cond
     ((< a zero)
      (setf tau
              (/ (* two b)
                 (- (f2cl-lib:fsqrt (+ (* a a) (* four b c))) a))))
     (t
      (setf tau
              (/ (+ a (f2cl-lib:fsqrt (+ (* a a) (* four b c))))
                 (* two c)))))))
(setf eta
        (/ tau
           (+ (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
              (f2cl-lib:fsqrt
               (+
                (* (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
                   (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%))
                tau)))))
(setf sigma (+ (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%) eta))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j n) nil)
  (tagbody
    (setf (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
            (- (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
               (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
               eta))
    (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
            (+ (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
               (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
               eta))))
```

```
(setf dpsi zero)
(setf psi zero)
(setf erretm zero)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j ii) nil)
  (tagbody
    (setf temp
            (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
               (*
                (f2cl-lib:fref delta-%data%
                               (j)
                               ((1 *))
                               delta-%offset%)
                (f2cl-lib:fref work-%data%
                               (j)
                               ((1 *))
                               work-%offset%))))
    (setf psi
            (+ psi
               (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
                  temp)))
    (setf dpsi (+ dpsi (* temp temp)))
    (setf erretm (+ erretm psi))))
(setf erretm (abs erretm))
(setf temp
        (/ (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
           (*
            (f2cl-lib:fref delta-%data% (n) ((1 *)) delta-%offset%)
            (f2cl-lib:fref work-%data% (n) ((1 *)) work-%offset%))))
(setf phi (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%) temp))
(setf dphi (* temp temp))
(setf erretm
        (+ (- (+ (* eight (- (- psi) phi)) erretm) phi)
           rhoinv
           (* (abs tau) (+ dpsi dphi))))
(setf w (+ rhoinv phi psi))
(cond
  ((<= (abs w) (* eps erretm))
   (go end_label)))
(setf niter (f2cl-lib:int-add niter 1))
(setf dtnsq1
        (*
         (f2cl-lib:fref work-%data%
                        ((f2cl-lib:int-sub n 1))
                        ((1 *))
                        work-%offset%)
```

```
         (f2cl-lib:fref delta-%data%
                        ((f2cl-lib:int-sub n 1))
                        ((1 *))
                        delta-%offset%)))
(setf dtnsq
        (* (f2cl-lib:fref work-%data% (n) ((1 *)) work-%offset%)
           (f2cl-lib:fref delta-%data% (n) ((1 *)) delta-%offset%)))
(setf c (- w (* dtnsq1 dpsi) (* dtnsq dphi)))
(setf a
        (+ (* (+ dtnsq dtnsq1) w)
           (* (- dtnsq) dtnsq1 (+ dpsi dphi))))
(setf b (* dtnsq dtnsq1 w))
(if (< c zero) (setf c (abs c)))
(cond
  ((= c zero)
   (setf eta (- rho (* sigma sigma))))
  ((>= a zero)
   (setf eta
           (/
            (+ a
               (f2cl-lib:fsqrt (abs (+ (* a a) (* (- four) b c)))))
            (* two c))))
  (t
   (setf eta
           (/ (* two b)
              (- a
                 (f2cl-lib:fsqrt
                  (abs (+ (* a a) (* (- four) b c)))))))))
(if (> (* w eta) zero) (setf eta (/ (- w) (+ dpsi dphi))))
(setf temp (- eta dtnsq))
(if (> temp rho) (setf eta (+ rho dtnsq)))
(setf tau (+ tau eta))
(setf eta (/ eta (+ sigma (f2cl-lib:fsqrt (+ eta (* sigma sigma))))))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j n) nil)
  (tagbody
    (setf (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
            (-
             (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
             eta))
    (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
            (+ (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
               eta))))
(setf sigma (+ sigma eta))
(setf dpsi zero)
(setf psi zero)
```

```
(setf erretm zero)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j ii) nil)
  (tagbody
    (setf temp
            (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
               (*
                (f2cl-lib:fref work-%data%
                               (j)
                               ((1 *))
                               work-%offset%)
                (f2cl-lib:fref delta-%data%
                               (j)
                               ((1 *))
                               delta-%offset%))))
    (setf psi
            (+ psi
               (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
                  temp)))
    (setf dpsi (+ dpsi (* temp temp)))
    (setf erretm (+ erretm psi))))
(setf erretm (abs erretm))
(setf temp
        (/ (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
           (* (f2cl-lib:fref work-%data% (n) ((1 *)) work-%offset%)
              (f2cl-lib:fref delta-%data%
                             (n)
                             ((1 *))
                             delta-%offset%))))
(setf phi (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%) temp))
(setf dphi (* temp temp))
(setf erretm
        (+ (- (+ (* eight (- (- psi) phi)) erretm) phi)
           rhoinv
           (* (abs tau) (+ dpsi dphi))))
(setf w (+ rhoinv phi psi))
(setf iter (f2cl-lib:int-add niter 1))
(f2cl-lib:fdo (niter iter (f2cl-lib:int-add niter 1))
              ((> niter maxit) nil)
  (tagbody
    (cond
      ((<= (abs w) (* eps erretm))
       (go end_label)))
    (setf dtnsq1
            (*
             (f2cl-lib:fref work-%data%
```

```
                          ((f2cl-lib:int-sub n 1))
                          ((1 *))
                          work-%offset%)
          (f2cl-lib:fref delta-%data%
                          ((f2cl-lib:int-sub n 1))
                          ((1 *))
                          delta-%offset%)))
(setf dtnsq
        (* (f2cl-lib:fref work-%data% (n) ((1 *)) work-%offset%)
           (f2cl-lib:fref delta-%data%
                          (n)
                          ((1 *))
                          delta-%offset%)))
(setf c (- w (* dtnsq1 dpsi) (* dtnsq dphi)))
(setf a
        (+ (* (+ dtnsq dtnsq1) w)
           (* (- dtnsq1) dtnsq (+ dpsi dphi))))
(setf b (* dtnsq1 dtnsq w))
(cond
  ((>= a zero)
   (setf eta
           (/
            (+ a
               (f2cl-lib:fsqrt
                (abs (+ (* a a) (* (- four) b c)))))
            (* two c))))
  (t
   (setf eta
           (/ (* two b)
              (- a
                 (f2cl-lib:fsqrt
                  (abs (+ (* a a) (* (- four) b c)))))))))
(if (> (* w eta) zero) (setf eta (/ (- w) (+ dpsi dphi))))
(setf temp (- eta dtnsq))
(if (<= temp zero) (setf eta (/ eta two)))
(setf tau (+ tau eta))
(setf eta
        (/ eta
           (+ sigma (f2cl-lib:fsqrt (+ eta (* sigma sigma))))))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j n) nil)
  (tagbody
    (setf (f2cl-lib:fref delta-%data%
                         (j)
                         ((1 *))
                         delta-%offset%)
```

```
                (-
                 (f2cl-lib:fref delta-%data%
                                (j)
                                ((1 *))
                                delta-%offset%)
                 eta))
            (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
                  (+
                   (f2cl-lib:fref work-%data%
                                  (j)
                                  ((1 *))
                                  work-%offset%)
                   eta))))
          (setf sigma (+ sigma eta))
          (setf dpsi zero)
          (setf psi zero)
          (setf erretm zero)
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                        ((> j ii) nil)
            (tagbody
              (setf temp
                    (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
                       (*
                        (f2cl-lib:fref work-%data%
                                       (j)
                                       ((1 *))
                                       work-%offset%)
                        (f2cl-lib:fref delta-%data%
                                       (j)
                                       ((1 *))
                                       delta-%offset%))))
              (setf psi
                    (+ psi
                       (*
                        (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
                        temp)))
              (setf dpsi (+ dpsi (* temp temp)))
              (setf erretm (+ erretm psi))))
          (setf erretm (abs erretm))
          (setf temp
                (/ (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
                   (*
                    (f2cl-lib:fref work-%data%
                                   (n)
                                   ((1 *))
                                   work-%offset%)
```

```
                    (f2cl-lib:fref delta-%data%
                                   (n)
                                   ((1 *))
                                   delta-%offset%))))
        (setf phi
                (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
                   temp))
        (setf dphi (* temp temp))
        (setf erretm
                (+ (- (+ (* eight (- (- psi) phi)) erretm) phi)
                   rhoinv
                   (* (abs tau) (+ dpsi dphi))))
        (setf w (+ rhoinv phi psi))))
     (setf info 1)
     (go end_label))
    (t
     (setf niter 1)
     (setf ip1 (f2cl-lib:int-add i 1))
     (setf delsq
             (*
              (- (f2cl-lib:fref d-%data% (ip1) ((1 *)) d-%offset%)
                 (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
              (+ (f2cl-lib:fref d-%data% (ip1) ((1 *)) d-%offset%)
                 (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))))
     (setf delsq2 (/ delsq two))
     (setf temp
             (/ delsq2
                (+ (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                   (f2cl-lib:fsqrt
                    (+
                     (* (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                        (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
                     delsq2)))))
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   ((> j n) nil)
       (tagbody
         (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
                 (+ (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
                    (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                    temp))
         (setf (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
                 (- (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
                    (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                    temp))))
     (setf psi zero)
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
```

```
                ((> j (f2cl-lib:int-add i (f2cl-lib:int-sub 1))) nil)
  (tagbody
    (setf psi
            (+ psi
               (/
                (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
                   (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%))
                (*
                 (f2cl-lib:fref work-%data%
                                (j)
                                ((1 *))
                                work-%offset%)
                 (f2cl-lib:fref delta-%data%
                                (j)
                                ((1 *))
                                delta-%offset%)))))))
(setf phi zero)
(f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
              ((> j (f2cl-lib:int-add i 2)) nil)
  (tagbody
    (setf phi
            (+ phi
               (/
                (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
                   (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%))
                (*
                 (f2cl-lib:fref work-%data%
                                (j)
                                ((1 *))
                                work-%offset%)
                 (f2cl-lib:fref delta-%data%
                                (j)
                                ((1 *))
                                delta-%offset%)))))))
(setf c (+ rhoinv psi phi))
(setf w
        (+ c
           (/
            (* (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
               (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%))
            (* (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
               (f2cl-lib:fref delta-%data%
                              (i)
                              ((1 *))
                              delta-%offset%)))
           (/
```

```
                (* (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%)
                   (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%))
                (*
                 (f2cl-lib:fref work-%data% (ip1) ((1 *)) work-%offset%)
                 (f2cl-lib:fref delta-%data%
                                (ip1)
                                ((1 *))
                                delta-%offset%)))))
          (cond
            ((> w zero)
             (setf orgati t)
             (setf sg2lb zero)
             (setf sg2ub delsq2)
             (setf a
                     (+ (* c delsq)
                        (* (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
                           (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%))
                        (* (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%)
                           (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%))))
             (setf b
                     (* (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
                        (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
                        delsq))
             (cond
               ((> a zero)
                (setf tau
                        (/ (* two b)
                           (+ a
                              (f2cl-lib:fsqrt
                               (abs (+ (* a a) (* (- four) b c))))))))
               (t
                (setf tau
                        (/
                         (- a
                            (f2cl-lib:fsqrt
                             (abs (+ (* a a) (* (- four) b c)))))
                         (* two c)))))
             (setf eta
                     (/ tau
                        (+ (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                           (f2cl-lib:fsqrt
                            (+
                             (*
                              (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                              (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
                             tau))))))
```

```
                (t
                 (setf orgati nil)
                 (setf sg2lb (- delsq2))
                 (setf sg2ub zero)
                 (setf a
                         (- (* c delsq)
                            (* (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
                               (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%))
                            (* (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%)
                               (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%))))
                 (setf b
                         (* (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%)
                            (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%)
                            delsq))
                 (cond
                   ((< a zero)
                    (setf tau
                            (/ (* two b)
                               (- a
                                  (f2cl-lib:fsqrt
                                   (abs (+ (* a a) (* four b c))))))))
                   (t
                    (setf tau
                            (/
                             (-
                              (+ a
                                 (f2cl-lib:fsqrt (abs (+ (* a a) (* four b c))))))
                             (* two c)))))
                 (setf eta
                         (/ tau
                            (+ (f2cl-lib:fref d-%data% (ip1) ((1 *)) d-%offset%)
                               (f2cl-lib:fsqrt
                                (abs
                                 (+
                                  (*
                                   (f2cl-lib:fref d-%data%
                                                  (ip1)
                                                  ((1 *))
                                                  d-%offset%)
                                   (f2cl-lib:fref d-%data%
                                                  (ip1)
                                                  ((1 *))
                                                  d-%offset%))
                                   tau))))))))
               (cond
                 (orgati
```

```
   (setf ii i)
   (setf sigma
           (+ (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) eta))
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                 ((> j n) nil)
     (tagbody
       (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
               (+ (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
                  (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                  eta))
       (setf (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
               (- (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
                  (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                  eta)))))
  (t
   (setf ii (f2cl-lib:int-add i 1))
   (setf sigma
           (+ (f2cl-lib:fref d-%data% (ip1) ((1 *)) d-%offset%) eta))
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                 ((> j n) nil)
     (tagbody
       (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
               (+ (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
                  (f2cl-lib:fref d-%data% (ip1) ((1 *)) d-%offset%)
                  eta))
       (setf (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
               (- (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
                  (f2cl-lib:fref d-%data% (ip1) ((1 *)) d-%offset%)
                  eta))))))
(setf iim1 (f2cl-lib:int-sub ii 1))
(setf iip1 (f2cl-lib:int-add ii 1))
(setf dpsi zero)
(setf psi zero)
(setf erretm zero)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j iim1) nil)
  (tagbody
    (setf temp
            (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
               (*
                (f2cl-lib:fref work-%data%
                               (j)
                               ((1 *))
                               work-%offset%)
                (f2cl-lib:fref delta-%data%
                               (j)
```

```
                                ((1 *))
                                delta-%offset%))))
    (setf psi
            (+ psi
               (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
                  temp)))
    (setf dpsi (+ dpsi (* temp temp)))
    (setf erretm (+ erretm psi))))
(setf erretm (abs erretm))
(setf dphi zero)
(setf phi zero)
(f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
              ((> j iip1) nil)
  (tagbody
    (setf temp
            (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
               (*
                (f2cl-lib:fref work-%data%
                               (j)
                               ((1 *))
                               work-%offset%)
                (f2cl-lib:fref delta-%data%
                               (j)
                               ((1 *))
                               delta-%offset%))))
    (setf phi
            (+ phi
               (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
                  temp)))
    (setf dphi (+ dphi (* temp temp)))
    (setf erretm (+ erretm phi))))
(setf w (+ rhoinv phi psi))
(setf swtch3 nil)
(cond
  (orgati
   (if (< w zero) (setf swtch3 t)))
  (t
   (if (> w zero) (setf swtch3 t))))
(if (or (= ii 1) (= ii n)) (setf swtch3 nil))
(setf temp
        (/ (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)
           (* (f2cl-lib:fref work-%data% (ii) ((1 *)) work-%offset%)
              (f2cl-lib:fref delta-%data%
                             (ii)
                             ((1 *))
                             delta-%offset%))))
```

```
(setf dw (+ dpsi dphi (* temp temp)))
(setf temp (* (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%) temp))
(setf w (+ w temp))
(setf erretm
        (+ (* eight (- phi psi))
           erretm
           (* two rhoinv)
           (* three (abs temp))
           (* (abs tau) dw)))
(cond
  ((<= (abs w) (* eps erretm))
   (go end_label)))
(cond
  ((<= w zero)
   (setf sg2lb (max sg2lb tau)))
  (t
   (setf sg2ub (min sg2ub tau))))
(setf niter (f2cl-lib:int-add niter 1))
(cond
  ((not swtch3)
   (setf dtipsq
           (*
            (f2cl-lib:fref work-%data% (ip1) ((1 *)) work-%offset%)
            (f2cl-lib:fref delta-%data%
                           (ip1)
                           ((1 *))
                           delta-%offset%)))
   (setf dtisq
           (* (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
              (f2cl-lib:fref delta-%data%
                             (i)
                             ((1 *))
                             delta-%offset%)))
   (cond
     (orgati
      (setf c
              (+ (- w (* dtipsq dw))
                 (* delsq
                    (expt
                     (/
                      (f2cl-lib:fref z-%data%
                                     (i)
                                     ((1 *))
                                     z-%offset%)
                      dtisq)
                     2)))))
```

```
                  (t
                   (setf c
                           (- w
                              (* dtisq dw)
                              (* delsq
                                 (expt
                                  (/
                                   (f2cl-lib:fref z-%data%
                                                  (ip1)
                                                  ((1 *))
                                                  z-%offset%)
                                   dtipsq)
                                  2))))))
                (setf a (+ (* (+ dtipsq dtisq) w) (* (- dtipsq) dtisq dw)))
                (setf b (* dtipsq dtisq w))
                (cond
                  ((= c zero)
                   (cond
                     ((= a zero)
                      (cond
                        (orgati
                         (setf a
                                 (+
                                  (*
                                   (f2cl-lib:fref z-%data%
                                                  (i)
                                                  ((1 *))
                                                  z-%offset%)
                                   (f2cl-lib:fref z-%data%
                                                  (i)
                                                  ((1 *))
                                                  z-%offset%))
                                  (* dtipsq dtipsq (+ dpsi dphi)))))
                        (t
                         (setf a
                                 (+
                                  (*
                                   (f2cl-lib:fref z-%data%
                                                  (ip1)
                                                  ((1 *))
                                                  z-%offset%)
                                   (f2cl-lib:fref z-%data%
                                                  (ip1)
                                                  ((1 *))
                                                  z-%offset%))
                                  (* dtisq dtisq (+ dpsi dphi))))))))
```

```
    (setf eta (/ b a)))
   ((<= a zero)
    (setf eta
            (/
             (- a
                (f2cl-lib:fsqrt
                 (abs (+ (* a a) (* (- four) b c)))))
             (* two c))))
   (t
    (setf eta
            (/ (* two b)
               (+ a
                  (f2cl-lib:fsqrt
                   (abs (+ (* a a) (* (- four) b c))))))))))
(t
 (setf dtiim
        (*
         (f2cl-lib:fref work-%data% (iim1) ((1 *)) work-%offset%)
         (f2cl-lib:fref delta-%data%
                        (iim1)
                        ((1 *))
                        delta-%offset%)))
 (setf dtiip
        (*
         (f2cl-lib:fref work-%data% (iip1) ((1 *)) work-%offset%)
         (f2cl-lib:fref delta-%data%
                        (iip1)
                        ((1 *))
                        delta-%offset%)))
 (setf temp (+ rhoinv psi phi))
 (cond
   (orgati
    (setf temp1
            (/ (f2cl-lib:fref z-%data% (iim1) ((1 *)) z-%offset%)
               dtiim))
    (setf temp1 (* temp1 temp1))
    (setf c
            (+ (- temp (* dtiip (+ dpsi dphi)))
               (*
                (-
                 (-
                  (f2cl-lib:fref d-%data%
                                 (iim1)
                                 ((1 *))
                                 d-%offset%)
                  (f2cl-lib:fref d-%data%
```

```
                                    (iip1)
                                    ((1 *))
                                    d-%offset%)))
                     (+
                      (f2cl-lib:fref d-%data%
                                     (iim1)
                                     ((1 *))
                                     d-%offset%)
                      (f2cl-lib:fref d-%data%
                                     (iip1)
                                     ((1 *))
                                     d-%offset%))
                      temp1)))
               (setf (f2cl-lib:fref zz (1) ((1 3)))
                     (* (f2cl-lib:fref z-%data% (iim1) ((1 *)) z-%offset%)
                        (f2cl-lib:fref z-%data% (iim1) ((1 *)) z-%offset%)))
               (cond
                 ((< dpsi temp1)
                  (setf (f2cl-lib:fref zz (3) ((1 3))) (* dtiip dtiip dphi)))
                 (t
                  (setf (f2cl-lib:fref zz (3) ((1 3)))
                        (* dtiip dtiip (+ (- dpsi temp1) dphi))))))
              (t
               (setf temp1
                     (/ (f2cl-lib:fref z-%data% (iip1) ((1 *)) z-%offset%)
                        dtiip))
               (setf temp1 (* temp1 temp1))
               (setf c
                     (+ (- temp (* dtiim (+ dpsi dphi)))
                        (*
                         (-
                          (-
                           (f2cl-lib:fref d-%data%
                                          (iip1)
                                          ((1 *))
                                          d-%offset%)
                           (f2cl-lib:fref d-%data%
                                          (iim1)
                                          ((1 *))
                                          d-%offset%)))
                          (+
                           (f2cl-lib:fref d-%data%
                                          (iim1)
                                          ((1 *))
                                          d-%offset%)
                           (f2cl-lib:fref d-%data%
```

```
                                        (iip1)
                                        ((1 *))
                                        d-%offset%))
                        temp1)))
            (cond
              ((< dphi temp1)
               (setf (f2cl-lib:fref zz (1) ((1 3))) (* dtiim dtiim dpsi)))
              (t
               (setf (f2cl-lib:fref zz (1) ((1 3)))
                       (* dtiim dtiim (+ dpsi (- dphi temp1))))))
            (setf (f2cl-lib:fref zz (3) ((1 3)))
                    (* (f2cl-lib:fref z-%data% (iip1) ((1 *)) z-%offset%)
                       (f2cl-lib:fref z-%data%
                                      (iip1)
                                      ((1 *))
                                      z-%offset%)))))
          (setf (f2cl-lib:fref zz (2) ((1 3)))
                  (* (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)
                     (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)))
          (setf (f2cl-lib:fref dd (1) ((1 3))) dtiim)
          (setf (f2cl-lib:fref dd (2) ((1 3)))
                  (*
                   (f2cl-lib:fref delta-%data% (ii) ((1 *)) delta-%offset%)
                   (f2cl-lib:fref work-%data% (ii) ((1 *)) work-%offset%)))
          (setf (f2cl-lib:fref dd (3) ((1 3))) dtiip)
          (multiple-value-bind
                (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
              (dlaed6 niter orgati c dd zz w eta info)
            (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5))
            (setf eta var-6)
            (setf info var-7))
          (if (/= info 0) (go end_label))))
      (if (>= (* w eta) zero) (setf eta (/ (- w) dw)))
      (cond
        (orgati
         (setf temp1
                 (* (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
                    (f2cl-lib:fref delta-%data%
                                   (i)
                                   ((1 *))
                                   delta-%offset%)))
         (setf temp (- eta temp1)))
        (t
         (setf temp1
                 (*
                  (f2cl-lib:fref work-%data% (ip1) ((1 *)) work-%offset%)
```

```
            (f2cl-lib:fref delta-%data%
                           (ip1)
                           ((1 *))
                           delta-%offset%)))
   (setf temp (- eta temp1))))
(cond
  ((or (> temp sg2ub) (< temp sg2lb))
   (cond
     ((< w zero)
      (setf eta (/ (- sg2ub tau) two)))
     (t
      (setf eta (/ (- sg2lb tau) two))))))
(setf tau (+ tau eta))
(setf eta (/ eta (+ sigma (f2cl-lib:fsqrt (+ (* sigma sigma) eta)))))
(setf prew w)
(setf sigma (+ sigma eta))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j n) nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
            (+ (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
               eta))
    (setf (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
            (-
             (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
             eta))))
(setf dpsi zero)
(setf psi zero)
(setf erretm zero)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j iim1) nil)
  (tagbody
    (setf temp
            (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
               (*
                (f2cl-lib:fref work-%data%
                               (j)
                               ((1 *))
                               work-%offset%)
                (f2cl-lib:fref delta-%data%
                               (j)
                               ((1 *))
                               delta-%offset%))))
    (setf psi
            (+ psi
               (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
```

```
                  temp)))
    (setf dpsi (+ dpsi (* temp temp)))
    (setf erretm (+ erretm psi))))
(setf erretm (abs erretm))
(setf dphi zero)
(setf phi zero)
(f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
              ((> j iip1) nil)
  (tagbody
    (setf temp
            (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
               (*
                (f2cl-lib:fref work-%data%
                               (j)
                               ((1 *))
                               work-%offset%)
                (f2cl-lib:fref delta-%data%
                               (j)
                               ((1 *))
                               delta-%offset%))))
    (setf phi
            (+ phi
               (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
                  temp)))
    (setf dphi (+ dphi (* temp temp)))
    (setf erretm (+ erretm phi))))
(setf temp
        (/ (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)
           (* (f2cl-lib:fref work-%data% (ii) ((1 *)) work-%offset%)
              (f2cl-lib:fref delta-%data%
                             (ii)
                             ((1 *))
                             delta-%offset%))))
(setf dw (+ dpsi dphi (* temp temp)))
(setf temp (* (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%) temp))
(setf w (+ rhoinv phi psi temp))
(setf erretm
        (+ (* eight (- phi psi))
           erretm
           (* two rhoinv)
           (* three (abs temp))
           (* (abs tau) dw)))
(cond
  ((<= w zero)
   (setf sg2lb (max sg2lb tau)))
  (t
```

```
   (setf sg2ub (min sg2ub tau))))
(setf swtch nil)
(cond
  (orgati
   (if (> (- w) (/ (abs prew) ten)) (setf swtch t)))
  (t
   (if (> w (/ (abs prew) ten)) (setf swtch t))))
(setf iter (f2cl-lib:int-add niter 1))
(f2cl-lib:fdo (niter iter (f2cl-lib:int-add niter 1))
              ((> niter maxit) nil)
  (tagbody
    (cond
      ((<= (abs w) (* eps erretm))
       (go end_label)))
    (cond
      ((not swtch3)
       (setf dtipsq
               (*
                (f2cl-lib:fref work-%data%
                               (ip1)
                               ((1 *))
                               work-%offset%)
                (f2cl-lib:fref delta-%data%
                               (ip1)
                               ((1 *))
                               delta-%offset%)))
       (setf dtisq
               (*
                (f2cl-lib:fref work-%data%
                               (i)
                               ((1 *))
                               work-%offset%)
                (f2cl-lib:fref delta-%data%
                               (i)
                               ((1 *))
                               delta-%offset%)))
       (cond
         ((not swtch)
          (cond
            (orgati
             (setf c
                     (+ (- w (* dtipsq dw))
                        (* delsq
                           (expt
                            (/
                             (f2cl-lib:fref z-%data%
```

```
                                          (i)
                                          ((1 *))
                                          z-%offset%)
                         dtisq)
                        2)))))
      (t
       (setf c
             (- w
                (* dtisq dw)
                (* delsq
                   (expt
                    (/
                     (f2cl-lib:fref z-%data%
                                    (ip1)
                                    ((1 *))
                                    z-%offset%)
                     dtipsq)
                    2)))))))
  (t
   (setf temp
         (/
          (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)
          (*
           (f2cl-lib:fref work-%data%
                          (ii)
                          ((1 *))
                          work-%offset%)
           (f2cl-lib:fref delta-%data%
                          (ii)
                          ((1 *))
                          delta-%offset%))))
   (cond
     (orgati
      (setf dpsi (+ dpsi (* temp temp))))
     (t
      (setf dphi (+ dphi (* temp temp)))))
   (setf c (- w (* dtisq dpsi) (* dtipsq dphi)))))
(setf a (+ (* (+ dtipsq dtisq) w) (* (- dtipsq) dtisq dw)))
(setf b (* dtipsq dtisq w))
(cond
  ((= c zero)
   (cond
     ((= a zero)
      (cond
        ((not swtch)
         (cond
```

```
                    (orgati
                     (setf a
                             (+
                              (*
                               (f2cl-lib:fref z-%data%
                                              (i)
                                              ((1 *))
                                              z-%offset%)
                               (f2cl-lib:fref z-%data%
                                              (i)
                                              ((1 *))
                                              z-%offset%))
                              (* dtipsq dtipsq (+ dpsi dphi)))))
                    (t
                     (setf a
                             (+
                              (*
                               (f2cl-lib:fref z-%data%
                                              (ip1)
                                              ((1 *))
                                              z-%offset%)
                               (f2cl-lib:fref z-%data%
                                              (ip1)
                                              ((1 *))
                                              z-%offset%))
                              (* dtisq dtisq (+ dpsi dphi)))))))
               (t
                (setf a
                        (+ (* dtisq dtisq dpsi)
                           (* dtipsq dtipsq dphi)))))
           (setf eta (/ b a)))
          ((<= a zero)
           (setf eta
                   (/
                    (- a
                       (f2cl-lib:fsqrt
                        (abs (+ (* a a) (* (- four) b c)))))
                    (* two c))))
          (t
           (setf eta
                   (/ (* two b)
                      (+ a
                         (f2cl-lib:fsqrt
                          (abs (+ (* a a) (* (- four) b c))))))))))
      (t
       (setf dtiim
```

```
        (*
         (f2cl-lib:fref work-%data%
                        (iim1)
                        ((1 *))
                        work-%offset%)
         (f2cl-lib:fref delta-%data%
                        (iim1)
                        ((1 *))
                        delta-%offset%)))
(setf dtiip
        (*
         (f2cl-lib:fref work-%data%
                        (iip1)
                        ((1 *))
                        work-%offset%)
         (f2cl-lib:fref delta-%data%
                        (iip1)
                        ((1 *))
                        delta-%offset%)))
(setf temp (+ rhoinv psi phi))
(cond
  (swtch
   (setf c (- temp (* dtiim dpsi) (* dtiip dphi)))
   (setf (f2cl-lib:fref zz (1) ((1 3))) (* dtiim dtiim dpsi))
   (setf (f2cl-lib:fref zz (3) ((1 3))) (* dtiip dtiip dphi)))
  (t
   (cond
     (orgati
      (setf temp1
              (/
               (f2cl-lib:fref z-%data%
                              (iim1)
                              ((1 *))
                              z-%offset%)
               dtiim))
      (setf temp1 (* temp1 temp1))
      (setf temp2
              (*
               (-
                (f2cl-lib:fref d-%data%
                               (iim1)
                               ((1 *))
                               d-%offset%)
                (f2cl-lib:fref d-%data%
                               (iip1)
                               ((1 *))
```

```
                                d-%offset%))
                         (+
                          (f2cl-lib:fref d-%data%
                                         (iim1)
                                         ((1 *))
                                         d-%offset%)
                          (f2cl-lib:fref d-%data%
                                         (iip1)
                                         ((1 *))
                                         d-%offset%))
                         temp1))
                 (setf c (- temp (* dtiip (+ dpsi dphi)) temp2))
                 (setf (f2cl-lib:fref zz (1) ((1 3)))
                       (*
                        (f2cl-lib:fref z-%data%
                                       (iim1)
                                       ((1 *))
                                       z-%offset%)
                        (f2cl-lib:fref z-%data%
                                       (iim1)
                                       ((1 *))
                                       z-%offset%)))
                 (cond
                   ((< dpsi temp1)
                    (setf (f2cl-lib:fref zz (3) ((1 3)))
                          (* dtiip dtiip dphi)))
                   (t
                    (setf (f2cl-lib:fref zz (3) ((1 3)))
                          (* dtiip dtiip (+ (- dpsi temp1) dphi))))))
                (t
                 (setf temp1
                       (/
                        (f2cl-lib:fref z-%data%
                                       (iip1)
                                       ((1 *))
                                       z-%offset%)
                        dtiip))
                 (setf temp1 (* temp1 temp1))
                 (setf temp2
                       (*
                        (-
                         (f2cl-lib:fref d-%data%
                                        (iip1)
                                        ((1 *))
                                        d-%offset%)
                         (f2cl-lib:fref d-%data%
```

```
                                    (iim1)
                                    ((1 *))
                                    d-%offset%))
                            (+
                             (f2cl-lib:fref d-%data%
                                    (iim1)
                                    ((1 *))
                                    d-%offset%)
                             (f2cl-lib:fref d-%data%
                                    (iip1)
                                    ((1 *))
                                    d-%offset%))
                            temp1))
                      (setf c (- temp (* dtiim (+ dpsi dphi)) temp2))
                      (cond
                        ((< dphi temp1)
                         (setf (f2cl-lib:fref zz (1) ((1 3)))
                                 (* dtiim dtiim dpsi)))
                        (t
                         (setf (f2cl-lib:fref zz (1) ((1 3)))
                                 (* dtiim dtiim (+ dpsi (- dphi temp1))))))
                      (setf (f2cl-lib:fref zz (3) ((1 3)))
                              (*
                               (f2cl-lib:fref z-%data%
                                      (iip1)
                                      ((1 *))
                                      z-%offset%)
                               (f2cl-lib:fref z-%data%
                                      (iip1)
                                      ((1 *))
                                      z-%offset%)))))))
                (setf (f2cl-lib:fref dd (1) ((1 3))) dtiim)
                (setf (f2cl-lib:fref dd (2) ((1 3)))
                        (*
                         (f2cl-lib:fref delta-%data%
                                        (ii)
                                        ((1 *))
                                        delta-%offset%)
                         (f2cl-lib:fref work-%data%
                                        (ii)
                                        ((1 *))
                                        work-%offset%)))
                (setf (f2cl-lib:fref dd (3) ((1 3))) dtiip)
                (multiple-value-bind
                      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
                    (dlaed6 niter orgati c dd zz w eta info)
```

```
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5))
     (setf eta var-6)
     (setf info var-7))
   (if (/= info 0) (go end_label))))
(if (>= (* w eta) zero) (setf eta (/ (- w) dw)))
(cond
  (orgati
   (setf temp1
         (*
          (f2cl-lib:fref work-%data%
                         (i)
                         ((1 *))
                         work-%offset%)
          (f2cl-lib:fref delta-%data%
                         (i)
                         ((1 *))
                         delta-%offset%)))
   (setf temp (- eta temp1)))
  (t
   (setf temp1
         (*
          (f2cl-lib:fref work-%data%
                         (ip1)
                         ((1 *))
                         work-%offset%)
          (f2cl-lib:fref delta-%data%
                         (ip1)
                         ((1 *))
                         delta-%offset%)))
   (setf temp (- eta temp1))))
(cond
  ((or (> temp sg2ub) (< temp sg2lb))
   (cond
     ((< w zero)
      (setf eta (/ (- sg2ub tau) two)))
     (t
      (setf eta (/ (- sg2lb tau) two))))))
(setf tau (+ tau eta))
(setf eta
       (/ eta
          (+ sigma (f2cl-lib:fsqrt (+ (* sigma sigma) eta)))))
(setf sigma (+ sigma eta))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j n) nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
```

```
                (+
                 (f2cl-lib:fref work-%data%
                                (j)
                                ((1 *))
                                work-%offset%)
                 eta))
            (setf (f2cl-lib:fref delta-%data%
                                 (j)
                                 ((1 *))
                                 delta-%offset%)
                  (-
                   (f2cl-lib:fref delta-%data%
                                  (j)
                                  ((1 *))
                                  delta-%offset%)
                   eta))))
        (setf prew w)
        (setf dpsi zero)
        (setf psi zero)
        (setf erretm zero)
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                      ((> j iim1) nil)
          (tagbody
            (setf temp
                  (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
                     (*
                      (f2cl-lib:fref work-%data%
                                     (j)
                                     ((1 *))
                                     work-%offset%)
                      (f2cl-lib:fref delta-%data%
                                     (j)
                                     ((1 *))
                                     delta-%offset%))))
            (setf psi
                  (+ psi
                     (*
                      (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
                      temp)))
            (setf dpsi (+ dpsi (* temp temp)))
            (setf erretm (+ erretm psi))))
        (setf erretm (abs erretm))
        (setf dphi zero)
        (setf phi zero)
        (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                      ((> j iip1) nil)
```

```
  (tagbody
    (setf temp
            (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
               (*
                (f2cl-lib:fref work-%data%
                               (j)
                               ((1 *))
                               work-%offset%)
                (f2cl-lib:fref delta-%data%
                               (j)
                               ((1 *))
                               delta-%offset%))))
    (setf phi
            (+ phi
               (*
                (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
                temp)))
    (setf dphi (+ dphi (* temp temp)))
    (setf erretm (+ erretm phi))))
(setf temp
        (/ (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)
           (*
            (f2cl-lib:fref work-%data%
                           (ii)
                           ((1 *))
                           work-%offset%)
            (f2cl-lib:fref delta-%data%
                           (ii)
                           ((1 *))
                           delta-%offset%))))
(setf dw (+ dpsi dphi (* temp temp)))
(setf temp
        (* (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)
           temp))
(setf w (+ rhoinv phi psi temp))
(setf erretm
        (+ (* eight (- phi psi))
           erretm
           (* two rhoinv)
           (* three (abs temp))
           (* (abs tau) dw)))
(if (and (> (* w prew) zero) (> (abs w) (/ (abs prew) ten)))
    (setf swtch (not swtch)))
(cond
  ((<= w zero)
   (setf sg2lb (max sg2lb tau)))
```

```
                (t
                 (setf sg2ub (min sg2ub tau))))))
            (setf info 1)))
   end_label
         (return (values nil nil nil nil nil nil sigma nil info))))))
```

7.59 dlasd5 LAPACK

⟨*dlasd5.input*⟩≡

```
  )set break resume
  )sys rm -f dlasd5.output
  )spool dlasd5.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dlasd5.help*⟩≡

```
  ======================================================================
  dlasd5 examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLASD5  -  compute the square root of the I-th eigenvalue of a positive
         symmetric rank-one modification of a 2-by-2 diagonal matrix   diag( D )
         *  diag(  D ) + RHO  The diagonal entries in the array D are assumed to
         satisfy   0 <= D(i) < D(j) for i < j

  SYNOPSIS
         SUBROUTINE DLASD5( I, D, Z, DELTA, RHO, DSIGMA, WORK )

             INTEGER        I

             DOUBLE         PRECISION DSIGMA, RHO

             DOUBLE         PRECISION D( 2 ), DELTA( 2 ), WORK( 2 ), Z( 2 )

  PURPOSE
         This subroutine computes the square root of the I-th  eigenvalue  of  a
         positive symmetric rank-one modification of a 2-by-2 diagonal matrix

         We  also  assume RHO > 0 and that the Euclidean norm of the vector Z is
         one.


  ARGUMENTS
         I      (input) INTEGER
                The index of the eigenvalue to be computed.  I = 1 or I = 2.

         D      (input) DOUBLE PRECISION array, dimension ( 2 )
                The original eigenvalues.  We assume 0 <= D(1) < D(2).

         Z      (input) DOUBLE PRECISION array, dimension ( 2 )
                The components of the updating vector.

         DELTA  (output) DOUBLE PRECISION array, dimension ( 2 )
                Contains (D(j) - sigma_I) in its  j-th  component.   The  vector
                DELTA contains the information necessary to construct the eigen-
                vectors.
```

```
RHO    (input) DOUBLE PRECISION
       The scalar in the symmetric updating formula.

       DSIGMA (output) DOUBLE PRECISION The computed sigma_I, the  I-th
       updated eigenvalue.

WORK   (workspace) DOUBLE PRECISION array, dimension ( 2 )
       WORK contains (D(j) + sigma_I) in its  j-th component.
```

⟨*LAPACK dlasd5*⟩≡

```
(let* ((zero 0.0) (one 1.0) (two 2.0) (three 3.0) (four 4.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two)
           (type (double-float 3.0 3.0) three)
           (type (double-float 4.0 4.0) four))
  (defun dlasd5 (i d z delta rho dsigma work)
    (declare (type (double-float) dsigma rho)
             (type (simple-array double-float (*)) work delta z d)
             (type fixnum i))
    (f2cl-lib:with-multi-array-data
        ((d double-float d-%data% d-%offset%)
         (z double-float z-%data% z-%offset%)
         (delta double-float delta-%data% delta-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((b 0.0) (c 0.0) (del 0.0) (delsq 0.0) (tau 0.0) (w 0.0))
        (declare (type (double-float) b c del delsq tau w))
        (setf del
                (- (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)
                   (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)))
        (setf delsq
                (* del
                   (+ (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)
                      (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%))))
        (cond
          ((= i 1)
           (setf w
                   (+ one
                      (/
                       (* four
                          rho
                          (+
                           (/
                            (* (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
                               (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%))
                            (+ (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)
                               (* three
                                  (f2cl-lib:fref d-%data%
                                                 (2)
                                                 ((1 2))
                                                 d-%offset%))))
                           (/
                            (*
                             (-
                              (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%))
```

```
                        (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%))
                       (+
                        (* three
                           (f2cl-lib:fref d-%data%
                                          (1)
                                          ((1 2))
                                          d-%offset%))
                        (f2cl-lib:fref d-%data%
                                       (2)
                                       ((1 2))
                                       d-%offset%)))))
                del)))
          (cond
            ((> w zero)
             (setf b
                   (+ delsq
                      (* rho
                         (+
                          (* (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%)
                             (f2cl-lib:fref z-%data%
                                            (1)
                                            ((1 2))
                                            z-%offset%))
                          (* (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
                             (f2cl-lib:fref z-%data%
                                            (2)
                                            ((1 2))
                                            z-%offset%))))))
             (setf c
                   (* rho
                      (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%)
                      (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%)
                      delsq))
             (setf tau
                   (/ (* two c)
                      (+ b (f2cl-lib:fsqrt (abs (- (* b b) (* four c)))))))
             (setf tau
                   (/ tau
                      (+ (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)
                         (f2cl-lib:fsqrt
                          (+
                           (*
                            (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)
                            (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%))
                           tau)))))
             (setf dsigma
```

```
         (+ (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%) tau))
 (setf (f2cl-lib:fref delta-%data% (1) ((1 2)) delta-%offset%)
         (- tau))
 (setf (f2cl-lib:fref delta-%data% (2) ((1 2)) delta-%offset%)
         (- del tau))
 (setf (f2cl-lib:fref work-%data% (1) ((1 2)) work-%offset%)
         (+
          (* two (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%))
          tau))
 (setf (f2cl-lib:fref work-%data% (2) ((1 2)) work-%offset%)
         (+ (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)
            tau
            (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%))))
(t
 (setf b
         (-
          (* rho
             (+
              (* (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%)
                 (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%))
              (* (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
                 (f2cl-lib:fref z-%data%
                                (2)
                                ((1 2))
                                z-%offset%))))
          delsq))
 (setf c
         (* rho
            (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
            (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
            delsq))
 (cond
   ((> b zero)
    (setf tau
            (/ (* (- two) c)
               (+ b (f2cl-lib:fsqrt (+ (* b b) (* four c)))))))
   (t
    (setf tau
            (/ (- b (f2cl-lib:fsqrt (+ (* b b) (* four c))))
               two))))
 (setf tau
         (/ tau
            (+ (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)
               (f2cl-lib:fsqrt
                (abs
                 (+
```

```
                        (*
                         (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)
                         (f2cl-lib:fref d-%data%
                                        (2)
                                        ((1 2))
                                        d-%offset%))
                        tau))))))
    (setf dsigma
            (+ (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%) tau))
    (setf (f2cl-lib:fref delta-%data% (1) ((1 2)) delta-%offset%)
            (- (+ del tau)))
    (setf (f2cl-lib:fref delta-%data% (2) ((1 2)) delta-%offset%)
            (- tau))
    (setf (f2cl-lib:fref work-%data% (1) ((1 2)) work-%offset%)
            (+ (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)
               tau
               (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)))
    (setf (f2cl-lib:fref work-%data% (2) ((1 2)) work-%offset%)
            (+
             (* two (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%))
             tau)))))
(t
 (setf b
         (-
          (* rho
             (+
              (* (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%)
                 (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%))
              (* (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
                 (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%))))
          delsq))
 (setf c
         (* rho
            (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
            (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
            delsq))
 (cond
   ((> b zero)
    (setf tau (/ (+ b (f2cl-lib:fsqrt (+ (* b b) (* four c)))) two)))
   (t
    (setf tau
            (/ (* two c)
               (- (f2cl-lib:fsqrt (+ (* b b) (* four c))) b)))))
 (setf tau
         (/ tau
            (+ (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)
```

```
                        (f2cl-lib:fsqrt
                         (+
                          (* (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)
                             (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%))
                          tau)))))
            (setf dsigma (+ (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%) tau))
            (setf (f2cl-lib:fref delta-%data% (1) ((1 2)) delta-%offset%)
                    (- (+ del tau)))
            (setf (f2cl-lib:fref delta-%data% (2) ((1 2)) delta-%offset%)
                    (- tau))
            (setf (f2cl-lib:fref work-%data% (1) ((1 2)) work-%offset%)
                    (+ (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)
                       tau
                       (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)))
            (setf (f2cl-lib:fref work-%data% (2) ((1 2)) work-%offset%)
                    (+ (* two (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%))
                       tau))))
          (return (values nil nil nil nil nil dsigma nil))))))
```

7.60 dlasd6 LAPACK

⟨*dlasd6.input*⟩≡

```
)set break resume
)sys rm -f dlasd6.output
)spool dlasd6.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasd6.help*⟩≡

```
======================================================================
dlasd6 examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DLASD6  -  the  SVD of an updated upper bidiagonal matrix B obtained by
       merging two smaller ones by appending a row

SYNOPSIS
       SUBROUTINE DLASD6( ICOMPQ, NL, NR, SQRE, D, VF, VL, ALPHA, BETA,  IDXQ,
                          PERM, GIVPTR, GIVCOL, LDGCOL, GIVNUM, LDGNUM, POLES,
                          DIFL, DIFR, Z, K, C, S, WORK, IWORK, INFO )

           INTEGER        GIVPTR, ICOMPQ, INFO, K,  LDGCOL,  LDGNUM,  NL,  NR,
                          SQRE

           DOUBLE         PRECISION ALPHA, BETA, C, S

           INTEGER        GIVCOL(  LDGCOL, * ), IDXQ( * ), IWORK( * ), PERM( *
                          )

           DOUBLE         PRECISION D( * ), DIFL( *  ),  DIFR(  *  ),  GIVNUM(
                          LDGNUM,  *  ), POLES( LDGNUM, * ), VF( * ), VL( * ),
                          WORK( * ), Z( * )

PURPOSE
       DLASD6 computes the  SVD  of  an  updated  upper  bidiagonal  matrix  B
       obtained  by  merging two smaller ones by appending a row. This routine
       is used only for the problem which requires  all  singular  values  and
       optionally  singular  vector matrices in factored form.  B is an N-by-M
       matrix with N = NL + NR + 1 and M = N + SQRE.   A  related  subroutine,
       DLASD1, handles the case in which all singular values and singular vec-
       tors of the bidiagonal matrix are desired.

       DLASD6 computes the SVD as follows:

                    ( D1(in)  0    0     0 )
         B = U(in) * (   Z1'   a   Z2'    b ) * VT(in)
                    (   0     0   D2(in) 0 )

           = U(out) * ( D(out) 0) * VT(out)
```

where Z' = (Z1' a Z2' b) = u' VT', and u is a vector of dimension M with ALPHA and BETA in the NL+1 and NL+2 th entries and zeros elsewhere; and the entry b is empty if SQRE = 0.

The singular values of B can be computed using D1, D2, the first components of all the right singular vectors of the lower block, and the last components of all the right singular vectors of the upper block. These components are stored and updated in VF and VL, respectively, in DLASD6. Hence U and VT are not explicitly referenced.

The singular values are stored in D. The algorithm consists of two stages:

> The first stage consists of deflating the size of the problem when there are multiple singular values or if there is a zero in the Z vector. For each such occurence the dimension of the secular equation problem is reduced by one. This stage is performed by the routine DLASD7.
>
> The second stage consists of calculating the updated singular values. This is done by finding the roots of the secular equation via the routine DLASD4 (as called by DLASD8). This routine also updates VF and VL and computes the distances between the updated singular values and the old singular values.

DLASD6 is called from DLASDA.

ARGUMENTS

ICOMPQ (input) INTEGER Specifies whether singular vectors are to be computed in factored form:
= 0: Compute singular values only.
= 1: Compute singular vectors in factored form as well.

NL (input) INTEGER
The row dimension of the upper block. NL >= 1.

NR (input) INTEGER
The row dimension of the lower block. NR >= 1.

SQRE (input) INTEGER
= 0: the lower block is an NR-by-NR square matrix.
= 1: the lower block is an NR-by-(NR+1) rectangular matrix.

```
          The bidiagonal matrix has row dimension N = NL +  NR  +  1,  and
          column dimension M = N + SQRE.

D         (input/output) DOUBLE PRECISION array, dimension ( NL+NR+1 ).
          On entry D(1:NL,1:NL) contains the singular values of the
          upper block, and D(NL+2:N) contains the singular values
          of  the lower block. On exit D(1:N) contains the singular values
          of the modified matrix.

VF        (input/output) DOUBLE PRECISION array, dimension ( M )
          On entry, VF(1:NL+1) contains the first components of all
          right singular vectors of the upper block; and  VF(NL+2:M)  con-
          tains  the first components of all right singular vectors of the
          lower block. On exit, VF contains the first  components  of  all
          right singular vectors of the bidiagonal matrix.

VL        (input/output) DOUBLE PRECISION array, dimension ( M )
          On entry, VL(1:NL+1) contains the  last components of all
          right  singular  vectors of the upper block; and VL(NL+2:M) con-
          tains the last components of all right singular vectors  of  the
          lower  block.  On  exit,  VL contains the last components of all
          right singular vectors of the bidiagonal matrix.

ALPHA     (input/output) DOUBLE PRECISION
          Contains the diagonal element associated with the added row.

BETA      (input/output) DOUBLE PRECISION
          Contains the off-diagonal element associated with the added row.

IDXQ      (output) INTEGER array, dimension ( N )
          This  contains  the  permutation which will reintegrate the sub-
          problem just solved back into sorted order, i.e.  D( IDXQ(  I  =
          1, N ) ) will be in ascending order.

PERM      (output) INTEGER array, dimension ( N )
          The  permutations  (from deflation and sorting) to be applied to
          each block. Not referenced if ICOMPQ = 0.

          GIVPTR (output) INTEGER The number  of  Givens  rotations  which
          took place in this subproblem. Not referenced if ICOMPQ = 0.

          GIVCOL (output) INTEGER array, dimension ( LDGCOL, 2 ) Each pair
          of numbers indicates a pair of columns to take place in a Givens
          rotation. Not referenced if ICOMPQ = 0.

          LDGCOL  (input)  INTEGER leading dimension of GIVCOL, must be at
```

least N.

GIVNUM (output) DOUBLE PRECISION array, dimension (LDGNUM, 2)
Each number indicates the C or S value to be used in the corresponding Givens rotation. Not referenced if ICOMPQ = 0.

LDGNUM (input) INTEGER The leading dimension of GIVNUM and POLES, must be at least N.

POLES (output) DOUBLE PRECISION array, dimension (LDGNUM, 2)
On exit, POLES(1,*) is an array containing the new singular values obtained from solving the secular equation, and POLES(2,*) is an array containing the poles in the secular equation. Not referenced if ICOMPQ = 0.

DIFL (output) DOUBLE PRECISION array, dimension (N)
On exit, DIFL(I) is the distance between I-th updated (undeflated) singular value and the I-th (undeflated) old singular value.

DIFR (output) DOUBLE PRECISION array,
dimension (LDGNUM, 2) if ICOMPQ = 1 and dimension (N) if ICOMPQ = 0. On exit, DIFR(I, 1) is the distance between I-th updated (undeflated) singular value and the I+1-th (undeflated) old singular value.

If ICOMPQ = 1, DIFR(1:K,2) is an array containing the normalizing factors for the right singular vector matrix.

See DLASD8 for details on DIFL and DIFR.

Z (output) DOUBLE PRECISION array, dimension (M)
The first elements of this array contain the components of the deflation-adjusted updating row vector.

K (output) INTEGER
Contains the dimension of the non-deflated matrix, This is the order of the related secular equation. 1 <= K <=N.

C (output) DOUBLE PRECISION
C contains garbage if SQRE =0 and the C-value of a Givens rotation related to the right null space if SQRE = 1.

S (output) DOUBLE PRECISION
S contains garbage if SQRE =0 and the S-value of a Givens rotation related to the right null space if SQRE = 1.

```
WORK   (workspace) DOUBLE PRECISION array, dimension ( 4 * M )

IWORK  (workspace) INTEGER array, dimension ( 3 * N )

INFO   (output) INTEGER
       = 0:  successful exit.
       < 0:  if INFO = -i, the i-th argument had an illegal value.
       > 0:  if INFO = 1, an singular value did not converge
```

⟨*LAPACK dlasd6*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dlasd6
         (icompq nl nr sqre d vf vl alpha beta idxq perm givptr givcol ldgcol
          givnum ldgnum poles difl difr z k c s work iwork info)
    (declare (type (simple-array fixnum (*)) iwork givcol perm idxq)
             (type (double-float) s c beta alpha)
             (type (simple-array double-float (*)) work z difr difl poles givnum vl vf
                                                   d)
             (type fixnum info k ldgnum ldgcol givptr sqre nr nl
                                              icompq))
    (f2cl-lib:with-multi-array-data
        ((d double-float d-%data% d-%offset%)
         (vf double-float vf-%data% vf-%offset%)
         (vl double-float vl-%data% vl-%offset%)
         (givnum double-float givnum-%data% givnum-%offset%)
         (poles double-float poles-%data% poles-%offset%)
         (difl double-float difl-%data% difl-%offset%)
         (difr double-float difr-%data% difr-%offset%)
         (z double-float z-%data% z-%offset%)
         (work double-float work-%data% work-%offset%)
         (idxq fixnum idxq-%data% idxq-%offset%)
         (perm fixnum perm-%data% perm-%offset%)
         (givcol fixnum givcol-%data% givcol-%offset%)
         (iwork fixnum iwork-%data% iwork-%offset%))
      (prog ((orgnrm 0.0) (i 0) (idx 0) (idxc 0) (idxp 0) (isigma 0) (ivfw 0)
             (ivlw 0) (iw 0) (m 0) (n 0) (n1 0) (n2 0))
        (declare (type (double-float) orgnrm)
                 (type fixnum i idx idxc idxp isigma ivfw ivlw iw
                                          m n n1 n2))
        (setf info 0)
        (setf n (f2cl-lib:int-add nl nr 1))
        (setf m (f2cl-lib:int-add n sqre))
        (cond
          ((or (< icompq 0) (> icompq 1))
           (setf info -1))
          ((< nl 1)
           (setf info -2))
          ((< nr 1)
           (setf info -3))
          ((or (< sqre 0) (> sqre 1))
           (setf info -4))
          ((< ldgcol n)
           (setf info -14))
```

```
  ((< ldgnum n)
   (setf info -16)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DLASD6" (f2cl-lib:int-sub info))
   (go end_label)))
(setf isigma 1)
(setf iw (f2cl-lib:int-add isigma n))
(setf ivfw (f2cl-lib:int-add iw m))
(setf ivlw (f2cl-lib:int-add ivfw m))
(setf idx 1)
(setf idxc (f2cl-lib:int-add idx n))
(setf idxp (f2cl-lib:int-add idxc n))
(setf orgnrm (max (abs alpha) (abs beta)))
(setf (f2cl-lib:fref d-%data%
                     ((f2cl-lib:int-add nl 1))
                     ((1 *))
                     d-%offset%)
        zero)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i n) nil)
  (tagbody
    (cond
      ((> (abs (f2cl-lib:fref d (i) ((1 *)))) orgnrm)
       (setf orgnrm
           (abs (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)))))))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
    (dlascl "G" 0 0 orgnrm one n 1 d n info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                   var-8))
  (setf info var-9))
(setf alpha (/ alpha orgnrm))
(setf beta (/ beta orgnrm))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
       var-10 var-11 var-12 var-13 var-14 var-15 var-16 var-17 var-18
       var-19 var-20 var-21 var-22 var-23 var-24 var-25 var-26)
    (dlasd7 icompq nl nr sqre k d z
     (f2cl-lib:array-slice work double-float (iw) ((1 *))) vf
     (f2cl-lib:array-slice work double-float (ivfw) ((1 *))) vl
     (f2cl-lib:array-slice work double-float (ivlw) ((1 *))) alpha beta
     (f2cl-lib:array-slice work double-float (isigma) ((1 *)))
     (f2cl-lib:array-slice iwork fixnum (idx) ((1 *)))
```

```
             (f2cl-lib:array-slice iwork fixnum (idxp) ((1 *))) idxq
             perm givptr givcol ldgcol givnum ldgnum c s info)
           (declare (ignore var-0 var-1 var-2 var-3 var-5 var-6 var-7 var-8
                            var-9 var-10 var-11 var-12 var-13 var-14 var-15
                            var-16 var-17 var-18 var-20 var-21 var-22 var-23))
           (setf k var-4)
           (setf givptr var-19)
           (setf c var-24)
           (setf s var-25)
           (setf info var-26))
         (multiple-value-bind
               (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
                var-10 var-11)
             (dlasd8 icompq k d z vf vl difl difr ldgnum
              (f2cl-lib:array-slice work double-float (isigma) ((1 *)))
              (f2cl-lib:array-slice work double-float (iw) ((1 *))) info)
           (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                            var-8 var-9 var-10))
           (setf info var-11))
         (cond
           ((= icompq 1)
            (dcopy k d 1
             (f2cl-lib:array-slice poles double-float (1 1) ((1 ldgnum) (1 *)))
             1)
            (dcopy k (f2cl-lib:array-slice work double-float (isigma) ((1 *))) 1
             (f2cl-lib:array-slice poles double-float (1 2) ((1 ldgnum) (1 *)))
             1)))
         (multiple-value-bind
               (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
             (dlascl "G" 0 0 one orgnrm n 1 d n info)
           (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                            var-8))
           (setf info var-9))
         (setf n1 k)
         (setf n2 (f2cl-lib:int-sub n k))
         (dlamrg n1 n2 d 1 -1 idxq)
   end_label
         (return
          (values nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  alpha
```

```
beta
nil
nil
givptr
nil
nil
nil
nil
nil
nil
nil
nil
k
c
s
nil
nil
info))))))
```

7.61 dlasd7 LAPACK

⟨*dlasd7.input*⟩≡

```
)set break resume
)sys rm -f dlasd7.output
)spool dlasd7.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasd7.help*⟩≡

```
  ======================================================================
  dlasd7 examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLASD7  - the two sets of singular values together into a single sorted
         set

  SYNOPSIS
         SUBROUTINE DLASD7( ICOMPQ, NL, NR, SQRE, K, D, Z, ZW, VF, VFW, VL, VLW,
                            ALPHA,  BETA, DSIGMA, IDX, IDXP, IDXQ, PERM, GIVPTR,
                            GIVCOL, LDGCOL, GIVNUM, LDGNUM, C, S, INFO )

             INTEGER        GIVPTR, ICOMPQ, INFO, K,  LDGCOL,  LDGNUM,  NL,  NR,
                            SQRE

             DOUBLE         PRECISION ALPHA, BETA, C, S

             INTEGER        GIVCOL( LDGCOL, * ), IDX( * ), IDXP( * ), IDXQ( * ),
                            PERM( * )

             DOUBLE         PRECISION D( * ), DSIGMA( * ), GIVNUM( LDGNUM, *  ),
                            VF(  * ), VFW( * ), VL( * ), VLW( * ), Z( * ), ZW( *
                            )

  PURPOSE
         DLASD7 merges the two sets of singular values together  into  a  single
         sorted set. Then it tries to deflate the size of the problem. There are
         two ways in which deflation can occur:  when two or more singular  val-
         ues are close together or if there is a tiny entry in the Z vector. For
         each such occurrence the order of the related secular equation  problem
         is reduced by one.

         DLASD7 is called from DLASD6.


  ARGUMENTS
         ICOMPQ  (input) INTEGER
                 Specifies  whether  singular vectors are to be computed in com-
                 pact form, as follows:
                 = 0: Compute singular values only.
```

```
          = 1: Compute singular vectors of  upper  bidiagonal  matrix  in
          compact form.

NL        (input) INTEGER
          The row dimension of the upper block. NL >= 1.

NR        (input) INTEGER
          The row dimension of the lower block. NR >= 1.

SQRE      (input) INTEGER
          = 0: the lower block is an NR-by-NR square matrix.
          = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

          The  bidiagonal matrix has N = NL + NR + 1 rows and M = N + SQRE
          >= N columns.

K         (output) INTEGER
          Contains the dimension of the non-deflated matrix, this  is  the
          order of the related secular equation. 1 <= K <=N.

D         (input/output) DOUBLE PRECISION array, dimension ( N )
          On  entry  D contains the singular values of the two submatrices
          to be combined. On exit D contains the  trailing  (N-K)  updated
          singular values (those which were deflated) sorted into increas-
          ing order.

Z         (output) DOUBLE PRECISION array, dimension ( M )
          On exit Z contains the updating row vector in the secular  equa-
          tion.

ZW        (workspace) DOUBLE PRECISION array, dimension ( M )
          Workspace for Z.

VF        (input/output) DOUBLE PRECISION array, dimension ( M )
          On entry, VF(1:NL+1) contains the first components of all
          right  singular  vectors of the upper block; and VF(NL+2:M) con-
          tains the first components of all right singular vectors of  the
          lower  block.  On  exit, VF contains the first components of all
          right singular vectors of the bidiagonal matrix.

VFW       (workspace) DOUBLE PRECISION array, dimension ( M )
          Workspace for VF.

VL        (input/output) DOUBLE PRECISION array, dimension ( M )
          On entry, VL(1:NL+1) contains the  last components of all
          right singular vectors of the upper block; and  VL(NL+2:M)  con-
```

tains the last components of all right singular vectors of the lower block. On exit, VL contains the last components of all right singular vectors of the bidiagonal matrix.

VLW (workspace) DOUBLE PRECISION array, dimension (M)
Workspace for VL.

ALPHA (input) DOUBLE PRECISION
Contains the diagonal element associated with the added row.

BETA (input) DOUBLE PRECISION
Contains the off-diagonal element associated with the added row.

DSIGMA (output) DOUBLE PRECISION array, dimension (N) Contains a copy of the diagonal elements (K-1 singular values and one zero) in the secular equation.

IDX (workspace) INTEGER array, dimension (N)
This will contain the permutation used to sort the contents of D into ascending order.

IDXP (workspace) INTEGER array, dimension (N)
This will contain the permutation used to place deflated values of D at the end of the array. On output IDXP(2:K)
points to the nondeflated D-values and IDXP(K+1:N) points to the deflated singular values.

IDXQ (input) INTEGER array, dimension (N)
This contains the permutation which separately sorts the two sub-problems in D into ascending order. Note that entries in the first half of this permutation must first be moved one position backward; and entries in the second half must first have NL+1 added to their values.

PERM (output) INTEGER array, dimension (N)
The permutations (from deflation and sorting) to be applied to each singular block. Not referenced if ICOMPQ = 0.

GIVPTR (output) INTEGER The number of Givens rotations which took place in this subproblem. Not referenced if ICOMPQ = 0.

GIVCOL (output) INTEGER array, dimension (LDGCOL, 2) Each pair of numbers indicates a pair of columns to take place in a Givens rotation. Not referenced if ICOMPQ = 0.

LDGCOL (input) INTEGER The leading dimension of GIVCOL, must be

```
              at least N.

              GIVNUM (output) DOUBLE PRECISION array, dimension ( LDGNUM, 2  )
              Each  number indicates the C or S value to be used in the corre-
              sponding Givens rotation. Not referenced if ICOMPQ = 0.

              LDGNUM (input) INTEGER The leading dimension of GIVNUM, must  be
              at least N.

       C      (output) DOUBLE PRECISION
              C  contains garbage if SQRE =0 and the C-value of a Givens rota-
              tion related to the right null space if SQRE = 1.

       S      (output) DOUBLE PRECISION
              S contains garbage if SQRE =0 and the S-value of a Givens  rota-
              tion related to the right null space if SQRE = 1.

       INFO   (output) INTEGER
              = 0:  successful exit.
              < 0:  if INFO = -i, the i-th argument had an illegal value.
```

⟨*LAPACK dlasd7*⟩≡

```
(let* ((zero 0.0) (one 1.0) (two 2.0) (eight 8.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two)
           (type (double-float 8.0 8.0) eight))
  (defun dlasd7
         (icompq nl nr sqre k d z zw vf vfw vl vlw alpha beta dsigma idx idxp
          idxq perm givptr givcol ldgcol givnum ldgnum c s info)
    (declare (type (simple-array fixnum (*)) givcol perm idxq idxp idx)
             (type (double-float) s c beta alpha)
             (type (simple-array double-float (*)) givnum dsigma vlw vl vfw vf zw z d)
             (type fixnum info ldgnum ldgcol givptr k sqre nr nl
                                    icompq))
    (f2cl-lib:with-multi-array-data
        ((d double-float d-%data% d-%offset%)
         (z double-float z-%data% z-%offset%)
         (zw double-float zw-%data% zw-%offset%)
         (vf double-float vf-%data% vf-%offset%)
         (vfw double-float vfw-%data% vfw-%offset%)
         (vl double-float vl-%data% vl-%offset%)
         (vlw double-float vlw-%data% vlw-%offset%)
         (dsigma double-float dsigma-%data% dsigma-%offset%)
         (givnum double-float givnum-%data% givnum-%offset%)
         (idx fixnum idx-%data% idx-%offset%)
         (idxp fixnum idxp-%data% idxp-%offset%)
         (idxq fixnum idxq-%data% idxq-%offset%)
         (perm fixnum perm-%data% perm-%offset%)
         (givcol fixnum givcol-%data% givcol-%offset%))
      (prog ((eps 0.0) (hlftol 0.0) (tau 0.0) (tol 0.0) (z1 0.0) (i 0) (idxi 0)
             (idxj 0) (idxjp 0) (j 0) (jp 0) (jprev 0) (k2 0) (m 0) (n 0)
             (nlp1 0) (nlp2 0))
        (declare (type (double-float) eps hlftol tau tol z1)
                 (type fixnum i idxi idxj idxjp j jp jprev k2 m n
                                        nlp1 nlp2))
        (setf info 0)
        (setf n (f2cl-lib:int-add nl nr 1))
        (setf m (f2cl-lib:int-add n sqre))
        (cond
          ((or (< icompq 0) (> icompq 1))
           (setf info -1))
          ((< nl 1)
           (setf info -2))
          ((< nr 1)
           (setf info -3))
          ((or (< sqre 0) (> sqre 1))
```

```
   (setf info -4))
  ((< ldgcol n)
   (setf info -22))
  ((< ldgnum n)
   (setf info -24)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DLASD7" (f2cl-lib:int-sub info))
   (go end_label)))
(setf nlp1 (f2cl-lib:int-add nl 1))
(setf nlp2 (f2cl-lib:int-add nl 2))
(cond
  ((= icompq 1)
   (setf givptr 0)))
(setf z1
        (* alpha (f2cl-lib:fref vl-%data% (nlp1) ((1 *)) vl-%offset%)))
(setf (f2cl-lib:fref vl-%data% (nlp1) ((1 *)) vl-%offset%) zero)
(setf tau (f2cl-lib:fref vf-%data% (nlp1) ((1 *)) vf-%offset%))
(f2cl-lib:fdo (i nl (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
              ((> i 1) nil)
  (tagbody
    (setf (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-add i 1))
                         ((1 *))
                         z-%offset%)
            (* alpha
               (f2cl-lib:fref vl-%data% (i) ((1 *)) vl-%offset%)))
    (setf (f2cl-lib:fref vl-%data% (i) ((1 *)) vl-%offset%) zero)
    (setf (f2cl-lib:fref vf-%data%
                         ((f2cl-lib:int-add i 1))
                         ((1 *))
                         vf-%offset%)
            (f2cl-lib:fref vf-%data% (i) ((1 *)) vf-%offset%))
    (setf (f2cl-lib:fref d-%data%
                         ((f2cl-lib:int-add i 1))
                         ((1 *))
                         d-%offset%)
            (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
    (setf (f2cl-lib:fref idxq-%data%
                         ((f2cl-lib:int-add i 1))
                         ((1 *))
                         idxq-%offset%)
            (f2cl-lib:int-add
             (f2cl-lib:fref idxq-%data% (i) ((1 *)) idxq-%offset%)
```

```
             1))))
(setf (f2cl-lib:fref vf-%data% (1) ((1 *)) vf-%offset%) tau)
(f2cl-lib:fdo (i nlp2 (f2cl-lib:int-add i 1))
              ((> i m) nil)
  (tagbody
    (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
            (* beta (f2cl-lib:fref vf-%data% (i) ((1 *)) vf-%offset%)))
    (setf (f2cl-lib:fref vf-%data% (i) ((1 *)) vf-%offset%) zero)))
(f2cl-lib:fdo (i nlp2 (f2cl-lib:int-add i 1))
              ((> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref idxq-%data% (i) ((1 *)) idxq-%offset%)
            (f2cl-lib:int-add
             (f2cl-lib:fref idxq-%data% (i) ((1 *)) idxq-%offset%)
             nlp1))))
(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
              ((> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref dsigma-%data% (i) ((1 *)) dsigma-%offset%)
            (f2cl-lib:fref d-%data%
                           ((f2cl-lib:fref idxq (i) ((1 *))))
                           ((1 *))
                           d-%offset%))
    (setf (f2cl-lib:fref zw-%data% (i) ((1 *)) zw-%offset%)
            (f2cl-lib:fref z-%data%
                           ((f2cl-lib:fref idxq (i) ((1 *))))
                           ((1 *))
                           z-%offset%))
    (setf (f2cl-lib:fref vfw-%data% (i) ((1 *)) vfw-%offset%)
            (f2cl-lib:fref vf-%data%
                           ((f2cl-lib:fref idxq (i) ((1 *))))
                           ((1 *))
                           vf-%offset%))
    (setf (f2cl-lib:fref vlw-%data% (i) ((1 *)) vlw-%offset%)
            (f2cl-lib:fref vl-%data%
                           ((f2cl-lib:fref idxq (i) ((1 *))))
                           ((1 *))
                           vl-%offset%))))
(dlamrg nl nr (f2cl-lib:array-slice dsigma double-float (2) ((1 *))) 1
 1 (f2cl-lib:array-slice idx fixnum (2) ((1 *))))
(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
              ((> i n) nil)
  (tagbody
    (setf idxi
            (f2cl-lib:int-add 1
                              (f2cl-lib:fref idx-%data%
```

```
                                                    (i)
                                                    ((1 *))
                                                    idx-%offset%)))
            (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                  (f2cl-lib:fref dsigma-%data%
                                 (idxi)
                                 ((1 *))
                                 dsigma-%offset%))
            (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
                  (f2cl-lib:fref zw-%data% (idxi) ((1 *)) zw-%offset%))
            (setf (f2cl-lib:fref vf-%data% (i) ((1 *)) vf-%offset%)
                  (f2cl-lib:fref vfw-%data% (idxi) ((1 *)) vfw-%offset%))
            (setf (f2cl-lib:fref vl-%data% (i) ((1 *)) vl-%offset%)
                  (f2cl-lib:fref vlw-%data% (idxi) ((1 *)) vlw-%offset%))))
          (setf eps (dlamch "Epsilon"))
          (setf tol (max (abs alpha) (abs beta)))
          (setf tol
                (* eight
                   eight
                   eps
                   (max (abs (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%))
                        tol)))
          (setf k 1)
          (setf k2 (f2cl-lib:int-add n 1))
          (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                        ((> j n) nil)
            (tagbody
              (cond
                ((<= (abs (f2cl-lib:fref z (j) ((1 *)))) tol)
                 (setf k2 (f2cl-lib:int-sub k2 1))
                 (setf (f2cl-lib:fref idxp-%data% (k2) ((1 *)) idxp-%offset%) j)
                 (if (= j n) (go label100)))
                (t
                 (setf jprev j)
                 (go label70)))))
label70
          (setf j jprev)
label80
          (setf j (f2cl-lib:int-add j 1))
          (if (> j n) (go label90))
          (cond
            ((<= (abs (f2cl-lib:fref z (j) ((1 *)))) tol)
             (setf k2 (f2cl-lib:int-sub k2 1))
             (setf (f2cl-lib:fref idxp-%data% (k2) ((1 *)) idxp-%offset%) j))
            (t
             (cond
```

```
((<=
  (abs
   (+ (f2cl-lib:fref d (j) ((1 *)))
      (- (f2cl-lib:fref d (jprev) ((1 *))))))
  tol)
 (setf s (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%))
 (setf c (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%))
 (setf tau (dlapy2 c s))
 (setf (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%) tau)
 (setf (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%) zero)
 (setf c (/ c tau))
 (setf s (/ (- s) tau))
 (cond
   ((= icompq 1)
    (setf givptr (f2cl-lib:int-add givptr 1))
    (setf idxjp
            (f2cl-lib:fref idxq-%data%
                           ((f2cl-lib:int-add
                             (f2cl-lib:fref idx (jprev) ((1 *)))
                             1))
                           ((1 *))
                           idxq-%offset%))
    (setf idxj
            (f2cl-lib:fref idxq-%data%
                           ((f2cl-lib:int-add
                             (f2cl-lib:fref idx (j) ((1 *)))
                             1))
                           ((1 *))
                           idxq-%offset%))
    (cond
      ((<= idxjp nlp1)
       (setf idxjp (f2cl-lib:int-sub idxjp 1))))
    (cond
      ((<= idxj nlp1)
       (setf idxj (f2cl-lib:int-sub idxj 1))))
    (setf (f2cl-lib:fref givcol-%data%
                         (givptr 2)
                         ((1 ldgcol) (1 *))
                         givcol-%offset%)
            idxjp)
    (setf (f2cl-lib:fref givcol-%data%
                         (givptr 1)
                         ((1 ldgcol) (1 *))
                         givcol-%offset%)
            idxj)
    (setf (f2cl-lib:fref givnum-%data%
```

```
                                    (givptr 2)
                                    ((1 ldgnum) (1 *))
                                    givnum-%offset%)
                            c)
                       (setf (f2cl-lib:fref givnum-%data%
                                    (givptr 1)
                                    ((1 ldgnum) (1 *))
                                    givnum-%offset%)
                            s)))
                     (drot 1 (f2cl-lib:array-slice vf double-float (jprev) ((1 *))) 1
                      (f2cl-lib:array-slice vf double-float (j) ((1 *))) 1 c s)
                     (drot 1 (f2cl-lib:array-slice vl double-float (jprev) ((1 *))) 1
                      (f2cl-lib:array-slice vl double-float (j) ((1 *))) 1 c s)
                     (setf k2 (f2cl-lib:int-sub k2 1))
                     (setf (f2cl-lib:fref idxp-%data% (k2) ((1 *)) idxp-%offset%)
                             jprev)
                     (setf jprev j))
                    (t
                     (setf k (f2cl-lib:int-add k 1))
                     (setf (f2cl-lib:fref zw-%data% (k) ((1 *)) zw-%offset%)
                             (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%))
                     (setf (f2cl-lib:fref dsigma-%data% (k) ((1 *)) dsigma-%offset%)
                             (f2cl-lib:fref d-%data% (jprev) ((1 *)) d-%offset%))
                     (setf (f2cl-lib:fref idxp-%data% (k) ((1 *)) idxp-%offset%) jprev)
                     (setf jprev j)))))
       (go label80)
label90
       (setf k (f2cl-lib:int-add k 1))
       (setf (f2cl-lib:fref zw-%data% (k) ((1 *)) zw-%offset%)
               (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%))
       (setf (f2cl-lib:fref dsigma-%data% (k) ((1 *)) dsigma-%offset%)
               (f2cl-lib:fref d-%data% (jprev) ((1 *)) d-%offset%))
       (setf (f2cl-lib:fref idxp-%data% (k) ((1 *)) idxp-%offset%) jprev)
label100
       (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                     ((> j n) nil)
         (tagbody
           (setf jp (f2cl-lib:fref idxp-%data% (j) ((1 *)) idxp-%offset%))
           (setf (f2cl-lib:fref dsigma-%data% (j) ((1 *)) dsigma-%offset%)
                   (f2cl-lib:fref d-%data% (jp) ((1 *)) d-%offset%))
           (setf (f2cl-lib:fref vfw-%data% (j) ((1 *)) vfw-%offset%)
                   (f2cl-lib:fref vf-%data% (jp) ((1 *)) vf-%offset%))
           (setf (f2cl-lib:fref vlw-%data% (j) ((1 *)) vlw-%offset%)
                   (f2cl-lib:fref vl-%data% (jp) ((1 *)) vl-%offset%))))
       (cond
         ((= icompq 1)
```

```
    (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                  ((> j n) nil)
      (tagbody
        (setf jp (f2cl-lib:fref idxp-%data% (j) ((1 *)) idxp-%offset%))
        (setf (f2cl-lib:fref perm-%data% (j) ((1 *)) perm-%offset%)
                (f2cl-lib:fref idxq-%data%
                               ((f2cl-lib:int-add
                                 (f2cl-lib:fref idx (jp) ((1 *)))
                                 1))
                               ((1 *))
                               idxq-%offset%))
        (cond
          ((<= (f2cl-lib:fref perm (j) ((1 *))) nlp1)
           (setf (f2cl-lib:fref perm-%data% (j) ((1 *)) perm-%offset%)
                   (f2cl-lib:int-sub
                    (f2cl-lib:fref perm-%data%
                                   (j)
                                   ((1 *))
                                   perm-%offset%)
                    1))))))))
(dcopy (f2cl-lib:int-sub n k)
 (f2cl-lib:array-slice dsigma double-float ((+ k 1)) ((1 *))) 1
 (f2cl-lib:array-slice d double-float ((+ k 1)) ((1 *))) 1)
(setf (f2cl-lib:fref dsigma-%data% (1) ((1 *)) dsigma-%offset%) zero)
(setf hlftol (/ tol two))
(if
 (<= (abs (f2cl-lib:fref dsigma-%data% (2) ((1 *)) dsigma-%offset%))
     hlftol)
 (setf (f2cl-lib:fref dsigma-%data% (2) ((1 *)) dsigma-%offset%)
         hlftol))
(cond
  ((> m n)
   (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
           (dlapy2 z1 (f2cl-lib:fref z-%data% (m) ((1 *)) z-%offset%)))
   (cond
     ((<= (f2cl-lib:fref z (1) ((1 *))) tol)
      (setf c one)
      (setf s zero)
      (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) tol))
     (t
      (setf c (/ z1 (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)))
      (setf s
              (/ (- (f2cl-lib:fref z-%data% (m) ((1 *)) z-%offset%))
                 (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)))))
   (drot 1 (f2cl-lib:array-slice vf double-float (m) ((1 *))) 1
    (f2cl-lib:array-slice vf double-float (1) ((1 *))) 1 c s)
```

```
           (drot 1 (f2cl-lib:array-slice vl double-float (m) ((1 *))) 1
            (f2cl-lib:array-slice vl double-float (1) ((1 *))) 1 c s))
          (t
           (cond
             ((<= (abs z1) tol)
              (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) tol))
             (t
              (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) z1)))))
        (dcopy (f2cl-lib:int-sub k 1)
         (f2cl-lib:array-slice zw double-float (2) ((1 *))) 1
         (f2cl-lib:array-slice z double-float (2) ((1 *))) 1)
        (dcopy (f2cl-lib:int-sub n 1)
         (f2cl-lib:array-slice vfw double-float (2) ((1 *))) 1
         (f2cl-lib:array-slice vf double-float (2) ((1 *))) 1)
        (dcopy (f2cl-lib:int-sub n 1)
         (f2cl-lib:array-slice vlw double-float (2) ((1 *))) 1
         (f2cl-lib:array-slice vl double-float (2) ((1 *))) 1)
end_label
        (return
         (values nil
                 nil
                 nil
                 nil
                 k
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 nil
                 givptr
                 nil
                 nil
                 nil
                 nil
                 c
                 s
                 info))))))
```

7.62 dlasd8 LAPACK

⟨*dlasd8.input*⟩≡

```
)set break resume
)sys rm -f dlasd8.output
)spool dlasd8.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasd8.help*⟩≡

```
======================================================================
dlasd8 examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DLASD8 - the square roots of the roots of the secular equation,

SYNOPSIS
       SUBROUTINE DLASD8( ICOMPQ, K, D, Z, VF, VL, DIFL, DIFR, LDDIFR, DSIGMA,
                          WORK, INFO )

           INTEGER        ICOMPQ, INFO, K, LDDIFR

           DOUBLE         PRECISION D( * ), DIFL( *  ),  DIFR(  LDDIFR,  *  ),
                          DSIGMA( * ), VF( * ), VL( * ), WORK( * ), Z( * )

PURPOSE
       DLASD8  finds the square roots of the roots of the secular equation, as
       defined by the values in DSIGMA and Z. It makes the  appropriate  calls
       to  DLASD4, and stores, for each  element in D, the distance to its two
       nearest poles (elements in DSIGMA). It also updates the arrays  VF  and
       VL,  the first and last components of all the right singular vectors of
       the original bidiagonal matrix.

       DLASD8 is called from DLASD6.


ARGUMENTS
       ICOMPQ  (input) INTEGER
               Specifies whether singular vectors are to be computed  in  fac-
               tored form in the calling routine:
               = 0: Compute singular values only.
               = 1: Compute singular vectors in factored form as well.

       K       (input) INTEGER
               The  number  of  terms in the rational function to be solved by
               DLASD4.  K >= 1.

       D       (output) DOUBLE PRECISION array, dimension ( K )
               On output, D contains the updated singular values.
```

```
Z       (input) DOUBLE PRECISION array, dimension ( K )
        The first K elements of this array contain the components of
        the deflation-adjusted updating row vector.

VF      (input/output) DOUBLE PRECISION array, dimension ( K )
        On entry, VF contains information passed through DBEDE8. On
        exit, VF contains the first K components of the first compo-
        nents of all right singular vectors of the bidiagonal matrix.

VL      (input/output) DOUBLE PRECISION array, dimension ( K )
        On entry, VL contains information passed through DBEDE8. On
        exit, VL contains the first K components of the last components
        of all right singular vectors of the bidiagonal matrix.

DIFL    (output) DOUBLE PRECISION array, dimension ( K )
        On exit, DIFL(I) = D(I) - DSIGMA(I).

DIFR    (output) DOUBLE PRECISION array,
        dimension ( LDDIFR, 2 ) if ICOMPQ = 1 and dimension ( K ) if
        ICOMPQ = 0. On exit, DIFR(I,1) = D(I) - DSIGMA(I+1), DIFR(K,1)
        is not defined and will not be referenced.

        If ICOMPQ = 1, DIFR(1:K,2) is an array containing the normaliz-
        ing factors for the right singular vector matrix.

LDDIFR  (input) INTEGER
        The leading dimension of DIFR, must be at least K.

DSIGMA  (input) DOUBLE PRECISION array, dimension ( K )
        The first K elements of this array contain the old roots of the
        deflated updating problem. These are the poles of the secular
        equation.

WORK    (workspace) DOUBLE PRECISION array, dimension at least 3 * K

INFO    (output) INTEGER
        = 0:  successful exit.
        < 0:  if INFO = -i, the i-th argument had an illegal value.
        > 0:  if INFO = 1, an singular value did not converge
```

⟨*LAPACK dlasd8*⟩≡

```
(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dlasd8 (icompq k d z vf vl difl difr lddifr dsigma work info)
    (declare (type (simple-array double-float (*)) work dsigma difr difl vl vf z d)
             (type fixnum info lddifr k icompq))
    (f2cl-lib:with-multi-array-data
        ((d double-float d-%data% d-%offset%)
         (z double-float z-%data% z-%offset%)
         (vf double-float vf-%data% vf-%offset%)
         (vl double-float vl-%data% vl-%offset%)
         (difl double-float difl-%data% difl-%offset%)
         (difr double-float difr-%data% difr-%offset%)
         (dsigma double-float dsigma-%data% dsigma-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((diflj 0.0) (difrj 0.0) (dj 0.0) (dsigj 0.0) (dsigjp 0.0)
             (rho 0.0) (temp 0.0) (i 0) (iwk1 0) (iwk2 0) (iwk2i 0) (iwk3 0)
             (iwk3i 0) (j 0))
        (declare (type (double-float) diflj difrj dj dsigj dsigjp rho temp)
                 (type fixnum i iwk1 iwk2 iwk2i iwk3 iwk3i j))
        (setf info 0)
        (cond
          ((or (< icompq 0) (> icompq 1))
           (setf info -1))
          ((< k 1)
           (setf info -2))
          ((< lddifr k)
           (setf info -9)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DLASD8" (f2cl-lib:int-sub info))
           (go end_label)))
        (cond
          ((= k 1)
           (setf (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)
                   (abs (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)))
           (setf (f2cl-lib:fref difl-%data% (1) ((1 *)) difl-%offset%)
                   (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))
           (cond
             ((= icompq 1)
              (setf (f2cl-lib:fref difl-%data% (2) ((1 *)) difl-%offset%) one)
              (setf (f2cl-lib:fref difr-%data%
                                   (1 2)
                                   ((1 lddifr) (1 *))
```

```
                              difr-%offset%)
                 one)))
     (go end_label)))
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                ((> i k) nil)
    (tagbody
      (setf (f2cl-lib:fref dsigma-%data% (i) ((1 *)) dsigma-%offset%)
              (-
               (multiple-value-bind (ret-val var-0 var-1)
                   (dlamc3
                    (f2cl-lib:fref dsigma-%data%
                                   (i)
                                   ((1 *))
                                   dsigma-%offset%)
                    (f2cl-lib:fref dsigma-%data%
                                   (i)
                                   ((1 *))
                                   dsigma-%offset%))
                 (declare (ignore))
                 (setf (f2cl-lib:fref dsigma-%data%
                                      (i)
                                      ((1 *))
                                      dsigma-%offset%)
                       var-0)
                 (setf (f2cl-lib:fref dsigma-%data%
                                      (i)
                                      ((1 *))
                                      dsigma-%offset%)
                       var-1)
                 ret-val)
               (f2cl-lib:fref dsigma-%data%
                              (i)
                              ((1 *))
                              dsigma-%offset%)))))
  (setf iwk1 1)
  (setf iwk2 (f2cl-lib:int-add iwk1 k))
  (setf iwk3 (f2cl-lib:int-add iwk2 k))
  (setf iwk2i (f2cl-lib:int-sub iwk2 1))
  (setf iwk3i (f2cl-lib:int-sub iwk3 1))
  (setf rho (dnrm2 k z 1))
  (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
      (dlascl "G" 0 0 rho one k 1 z k info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                     var-8))
    (setf info var-9))
```

```
(setf rho (* rho rho))
(dlaset "A" k 1 one one
 (f2cl-lib:array-slice work double-float (iwk3) ((1 *))) k)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j k) nil)
  (tagbody
    (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
        (dlasd4 k j dsigma z
         (f2cl-lib:array-slice work double-float (iwk1) ((1 *))) rho
         (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
         (f2cl-lib:array-slice work double-float (iwk2) ((1 *))) info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-7))
      (setf (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%) var-6)
      (setf info var-8))
    (cond
      ((/= info 0)
       (go end_label)))
    (setf (f2cl-lib:fref work-%data%
                         ((f2cl-lib:int-add iwk3i j))
                         ((1 *))
                         work-%offset%)
            (*
             (f2cl-lib:fref work-%data%
                            ((f2cl-lib:int-add iwk3i j))
                            ((1 *))
                            work-%offset%)
             (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
             (f2cl-lib:fref work-%data%
                            ((f2cl-lib:int-add iwk2i j))
                            ((1 *))
                            work-%offset%)))
    (setf (f2cl-lib:fref difl-%data% (j) ((1 *)) difl-%offset%)
            (- (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)))
    (setf (f2cl-lib:fref difr-%data%
                         (j 1)
                         ((1 lddifr) (1 *))
                         difr-%offset%)
            (-
             (f2cl-lib:fref work-%data%
                            ((f2cl-lib:int-add j 1))
                            ((1 *))
                            work-%offset%)))
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))) nil)
      (tagbody
```

```
                (setf (f2cl-lib:fref work-%data%
                                     ((f2cl-lib:int-add iwk3i i))
                                     ((1 *))
                                     work-%offset%)
                      (/
                       (/
                        (*
                         (f2cl-lib:fref work-%data%
                                        ((f2cl-lib:int-add iwk3i i))
                                        ((1 *))
                                        work-%offset%)
                         (f2cl-lib:fref work-%data%
                                        (i)
                                        ((1 *))
                                        work-%offset%)
                         (f2cl-lib:fref work-%data%
                                        ((f2cl-lib:int-add iwk2i i))
                                        ((1 *))
                                        work-%offset%))
                        (-
                         (f2cl-lib:fref dsigma-%data%
                                        (i)
                                        ((1 *))
                                        dsigma-%offset%)
                         (f2cl-lib:fref dsigma-%data%
                                        (j)
                                        ((1 *))
                                        dsigma-%offset%)))
                       (+
                        (f2cl-lib:fref dsigma-%data%
                                       (i)
                                       ((1 *))
                                       dsigma-%offset%)
                        (f2cl-lib:fref dsigma-%data%
                                       (j)
                                       ((1 *))
                                       dsigma-%offset%))))))
          (f2cl-lib:fdo (i (f2cl-lib:int-add j 1) (f2cl-lib:int-add i 1))
                        ((> i k) nil)
            (tagbody
              (setf (f2cl-lib:fref work-%data%
                                   ((f2cl-lib:int-add iwk3i i))
                                   ((1 *))
                                   work-%offset%)
                    (/
                     (/
```

```
                        (*
                         (f2cl-lib:fref work-%data%
                                        ((f2cl-lib:int-add iwk3i i))
                                        ((1 *))
                                        work-%offset%)
                         (f2cl-lib:fref work-%data%
                                        (i)
                                        ((1 *))
                                        work-%offset%)
                         (f2cl-lib:fref work-%data%
                                        ((f2cl-lib:int-add iwk2i i))
                                        ((1 *))
                                        work-%offset%))
                        (-
                         (f2cl-lib:fref dsigma-%data%
                                        (i)
                                        ((1 *))
                                        dsigma-%offset%)
                         (f2cl-lib:fref dsigma-%data%
                                        (j)
                                        ((1 *))
                                        dsigma-%offset%)))
                       (+
                        (f2cl-lib:fref dsigma-%data%
                                       (i)
                                       ((1 *))
                                       dsigma-%offset%)
                        (f2cl-lib:fref dsigma-%data%
                                       (j)
                                       ((1 *))
                                       dsigma-%offset%))))))))
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i k) nil)
            (tagbody
              (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
                      (f2cl-lib:sign
                       (f2cl-lib:fsqrt
                        (abs
                         (f2cl-lib:fref work-%data%
                                        ((f2cl-lib:int-add iwk3i i))
                                        ((1 *))
                                        work-%offset%)))
                       (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)))))
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                        ((> j k) nil)
            (tagbody
```

```
(setf diflj (f2cl-lib:fref difl-%data% (j) ((1 *)) difl-%offset%))
(setf dj (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))
(setf dsigj
        (-
         (f2cl-lib:fref dsigma-%data%
                        (j)
                        ((1 *))
                        dsigma-%offset%)))
(cond
  ((< j k)
   (setf difrj
           (-
            (f2cl-lib:fref difr-%data%
                           (j 1)
                           ((1 lddifr) (1 *))
                           difr-%offset%)))
   (setf dsigjp
           (-
            (f2cl-lib:fref dsigma-%data%
                           ((f2cl-lib:int-add j 1))
                           ((1 *))
                           dsigma-%offset%)))))
(setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
        (/
         (/ (- (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%))
            diflj)
         (+
          (f2cl-lib:fref dsigma-%data% (j) ((1 *)) dsigma-%offset%)
          dj)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))) nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
            (/
             (/ (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
                (-
                 (multiple-value-bind (ret-val var-0 var-1)
                     (dlamc3
                      (f2cl-lib:fref dsigma-%data%
                                     (i)
                                     ((1 *))
                                     dsigma-%offset%)
                      dsigj)
                   (declare (ignore))
                   (setf (f2cl-lib:fref dsigma-%data%
                                        (i)
```

```
                                          ((1 *))
                                          dsigma-%offset%)
                                  var-0)
                            (setf dsigj var-1)
                            ret-val)
                           diflj))
                      (+
                       (f2cl-lib:fref dsigma-%data%
                                      (i)
                                      ((1 *))
                                      dsigma-%offset%)
                       dj)))))
          (f2cl-lib:fdo (i (f2cl-lib:int-add j 1) (f2cl-lib:int-add i 1))
                       ((> i k) nil)
            (tagbody
              (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
                    (/
                     (/ (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
                        (+
                         (multiple-value-bind (ret-val var-0 var-1)
                             (dlamc3
                              (f2cl-lib:fref dsigma-%data%
                                             (i)
                                             ((1 *))
                                             dsigma-%offset%)
                              dsigjp)
                           (declare (ignore))
                           (setf (f2cl-lib:fref dsigma-%data%
                                                (i)
                                                ((1 *))
                                                dsigma-%offset%)
                                 var-0)
                           (setf dsigjp var-1)
                           ret-val)
                         difrj))
                      (+
                       (f2cl-lib:fref dsigma-%data%
                                      (i)
                                      ((1 *))
                                      dsigma-%offset%)
                       dj)))))
          (setf temp (dnrm2 k work 1))
          (setf (f2cl-lib:fref work-%data%
                               ((f2cl-lib:int-add iwk2i j))
                               ((1 *))
                               work-%offset%)
```

```
                (/ (ddot k work 1 vf 1) temp))
          (setf (f2cl-lib:fref work-%data%
                               ((f2cl-lib:int-add iwk3i j))
                               ((1 *))
                               work-%offset%)
                (/ (ddot k work 1 vl 1) temp))
          (cond
            ((= icompq 1)
             (setf (f2cl-lib:fref difr-%data%
                                  (j 2)
                                  ((1 lddifr) (1 *))
                                  difr-%offset%)
                   temp)))))
      (dcopy k (f2cl-lib:array-slice work double-float (iwk2) ((1 *))) 1 vf 1)
      (dcopy k (f2cl-lib:array-slice work double-float (iwk3) ((1 *))) 1 vl 1)
end_label
      (return (values nil nil nil nil nil nil nil nil nil nil nil info))))))
```

7.63 dlasda LAPACK

⟨*dlasda.input*⟩≡

```
)set break resume
)sys rm -f dlasda.output
)spool dlasda.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasda.help*⟩≡

```
======================================================================
dlasda examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DLASDA - divide and conquer approach, DLASDA computes the singular
       value decomposition (SVD) of a real upper bidiagonal N-by-M matrix B
       with diagonal D and offdiagonal E, where M = N + SQRE

SYNOPSIS
       SUBROUTINE DLASDA( ICOMPQ, SMLSIZ, N, SQRE, D, E, U, LDU, VT, K, DIFL,
                          DIFR, Z, POLES, GIVPTR, GIVCOL, LDGCOL, PERM,
                          GIVNUM, C, S, WORK, IWORK, INFO )

           INTEGER        ICOMPQ, INFO, LDGCOL, LDU, N, SMLSIZ, SQRE

           INTEGER        GIVCOL( LDGCOL, * ), GIVPTR( * ), IWORK( * ), K( *
                          ), PERM( LDGCOL, * )

           DOUBLE         PRECISION C( * ), D( * ), DIFL( LDU, * ), DIFR( LDU,
                          * ), E( * ), GIVNUM( LDU, * ), POLES( LDU, * ), S( *
                          ), U( LDU, * ), VT( LDU, * ), WORK( * ), Z( LDU, * )

PURPOSE
       Using a divide and conquer approach, DLASDA computes the singular value
       decomposition (SVD) of a real upper bidiagonal N-by-M matrix B with
       diagonal D and offdiagonal E, where M = N + SQRE. The algorithm com-
       putes the singular values in the SVD B = U * S * VT. The orthogonal
       matrices U and VT are optionally computed in compact form.

       A related subroutine, DLASD0, computes the singular values and the sin-
       gular vectors in explicit form.


ARGUMENTS
       ICOMPQ (input) INTEGER Specifies whether singular vectors are to be
       computed in compact form, as follows = 0: Compute singular values only.
       = 1: Compute singular vectors of upper bidiagonal matrix in compact
       form.

       SMLSIZ (input) INTEGER The maximum size of the subproblems at the bot-
```

```
tom of the computation tree.

N      (input) INTEGER
       The row dimension of the upper bidiagonal matrix. This  is  also
       the dimension of the main diagonal array D.

SQRE   (input) INTEGER
       Specifies  the  column dimension of the bidiagonal matrix.  = 0:
       The bidiagonal matrix has column dimension M = N;
       = 1: The bidiagonal matrix has column dimension M = N + 1.

D      (input/output) DOUBLE PRECISION array, dimension ( N )
       On entry D contains the main diagonal of the bidiagonal  matrix.
       On exit D, if INFO = 0, contains its singular values.

E      (input) DOUBLE PRECISION array, dimension ( M-1 )
       Contains  the  subdiagonal entries of the bidiagonal matrix.  On
       exit, E has been destroyed.

U      (output) DOUBLE PRECISION array,
       dimension ( LDU, SMLSIZ ) if ICOMPQ = 1, and not  referenced  if
       ICOMPQ = 0. If ICOMPQ = 1, on exit, U contains the left singular
       vector matrices of all subproblems at the bottom level.

LDU    (input) INTEGER, LDU = > N.
       The leading dimension  of  arrays  U,  VT,  DIFL,  DIFR,  POLES,
       GIVNUM, and Z.

VT     (output) DOUBLE PRECISION array,
       dimension ( LDU, SMLSIZ+1 ) if ICOMPQ = 1, and not referenced if
       ICOMPQ = 0. If ICOMPQ = 1, on exit, VT' contains the right  sin-
       gular vector matrices of all subproblems at the bottom level.

K      (output) INTEGER array,
       dimension ( N ) if ICOMPQ = 1 and dimension 1 if ICOMPQ = 0.  If
       ICOMPQ = 1, on exit, K(I) is the dimension of the  I-th  secular
       equation on the computation tree.

DIFL   (output) DOUBLE PRECISION array, dimension ( LDU, NLVL ),
       where NLVL = floor(log_2 (N/SMLSIZ))).

DIFR   (output) DOUBLE PRECISION array,
       dimension ( LDU, 2 * NLVL ) if ICOMPQ = 1 and dimension ( N ) if
       ICOMPQ = 0.  If ICOMPQ = 1, on exit, DIFL(1:N, I) and  DIFR(1:N,
       2  * I - 1) record distances between singular values on the I-th
       level and singular values on the (I -1)-th level, and  DIFR(1:N,
```

2 * I) contains the normalizing factors for the right singular vector matrix. See DLASD8 for details.

Z (output) DOUBLE PRECISION array,
dimension (LDU, NLVL) if ICOMPQ = 1 and dimension (N) if ICOMPQ = 0. The first K elements of Z(1, I) contain the components of the deflation-adjusted updating row vector for subproblems on the I-th level.

POLES (output) DOUBLE PRECISION array,
dimension (LDU, 2 * NLVL) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, POLES(1, 2*I - 1) and POLES(1, 2*I) contain the new and old singular values involved in the secular equations on the I-th level.

GIVPTR (output) INTEGER array, dimension (N) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, GIVPTR(I) records the number of Givens rotations performed on the I-th problem on the computation tree.

GIVCOL (output) INTEGER array, dimension (LDGCOL, 2 * NLVL) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, for each I, GIVCOL(1, 2 *I - 1) and GIVCOL(1, 2 *I) record the locations of Givens rotations performed on the I-th level on the computation tree.

LDGCOL (input) INTEGER, LDGCOL = > N. The leading dimension of arrays GIVCOL and PERM.

PERM (output) INTEGER array,
dimension (LDGCOL, NLVL) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, PERM(1, I) records permutations done on the I-th level of the computation tree.

GIVNUM (output) DOUBLE PRECISION array, dimension (LDU, 2 * NLVL) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, for each I, GIVNUM(1, 2 *I - 1) and GIVNUM(1, 2 *I) record the C- and S- values of Givens rotations performed on the I-th level on the computation tree.

C (output) DOUBLE PRECISION array,
dimension (N) if ICOMPQ = 1, and dimension 1 if ICOMPQ = 0. If ICOMPQ = 1 and the I-th subproblem is not square, on exit, C(I) contains the C-value of a Givens rotation related to the right null space of the I-th subproblem.

```
S       (output) DOUBLE PRECISION array, dimension ( N ) if
        ICOMPQ = 1, and dimension 1 if ICOMPQ = 0. If ICOMPQ = 1 and the
        I-th subproblem is not square, on exit, S( I ) contains  the  S-
        value  of  a  Givens rotation related to the right null space of
        the I-th subproblem.

WORK    (workspace) DOUBLE PRECISION array, dimension
        (6 * N + (SMLSIZ + 1)*(SMLSIZ + 1)).

IWORK   (workspace) INTEGER array.
        Dimension must be at least (7 * N).

INFO    (output) INTEGER
        = 0:  successful exit.
        < 0:  if INFO = -i, the i-th argument had an illegal value.
        > 0:  if INFO = 1, an singular value did not converge
```

⟨*LAPACK dlasda*⟩≡
```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dlasda
         (icompq smlsiz n sqre d e u ldu vt k difl difr z poles givptr givcol
          ldgcol perm givnum c s work iwork info)
    (declare (type (simple-array fixnum (*)) iwork perm givcol givptr k)
             (type (simple-array double-float (*)) work s c givnum poles z difr difl
                                                   vt u e d)
             (type fixnum info ldgcol ldu sqre n smlsiz icompq))
    (f2cl-lib:with-multi-array-data
        ((d double-float d-%data% d-%offset%)
         (e double-float e-%data% e-%offset%)
         (u double-float u-%data% u-%offset%)
         (vt double-float vt-%data% vt-%offset%)
         (difl double-float difl-%data% difl-%offset%)
         (difr double-float difr-%data% difr-%offset%)
         (z double-float z-%data% z-%offset%)
         (poles double-float poles-%data% poles-%offset%)
         (givnum double-float givnum-%data% givnum-%offset%)
         (c double-float c-%data% c-%offset%)
         (s double-float s-%data% s-%offset%)
         (work double-float work-%data% work-%offset%)
         (k fixnum k-%data% k-%offset%)
         (givptr fixnum givptr-%data% givptr-%offset%)
         (givcol fixnum givcol-%data% givcol-%offset%)
         (perm fixnum perm-%data% perm-%offset%)
         (iwork fixnum iwork-%data% iwork-%offset%))
      (prog ((alpha 0.0) (beta 0.0) (i 0) (i1 0) (ic 0) (idxq 0) (idxqi 0)
             (im1 0) (inode 0) (itemp 0) (iwk 0) (j 0) (lf 0) (ll 0) (lvl 0)
             (lvl2 0) (m 0) (ncc 0) (nd 0) (ndb1 0) (ndiml 0) (ndimr 0) (nl 0)
             (nlf 0) (nlp1 0) (nlvl 0) (nr 0) (nrf 0) (nrp1 0) (nru 0)
             (nwork1 0) (nwork2 0) (smlszp 0) (sqrei 0) (vf 0) (vfi 0) (vl 0)
             (vli 0))
        (declare (type (double-float) alpha beta)
                 (type fixnum i i1 ic idxq idxqi im1 inode itemp
                                       iwk j lf ll lvl lvl2 m ncc nd ndb1
                                       ndiml ndimr nl nlf nlp1 nlvl nr nrf
                                       nrp1 nru nwork1 nwork2 smlszp sqrei
                                       vf vfi vl vli))
        (setf info 0)
        (cond
          ((or (< icompq 0) (> icompq 1))
           (setf info -1))
          ((< smlsiz 3)
```

```
   (setf info -2))
  ((< n 0)
   (setf info -3))
  ((or (< sqre 0) (> sqre 1))
   (setf info -4))
  ((< ldu (f2cl-lib:int-add n sqre))
   (setf info -8))
  ((< ldgcol n)
   (setf info -17)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DLASDA" (f2cl-lib:int-sub info))
   (go end_label)))
(setf m (f2cl-lib:int-add n sqre))
(cond
  ((<= n smlsiz)
   (cond
     ((= icompq 0)
      (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
             var-9 var-10 var-11 var-12 var-13 var-14 var-15)
          (dlasdq "U" sqre n 0 0 0 d e vt ldu u ldu u ldu work info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                         var-7 var-8 var-9 var-10 var-11 var-12 var-13
                         var-14))
        (setf info var-15)))
     (t
      (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
             var-9 var-10 var-11 var-12 var-13 var-14 var-15)
          (dlasdq "U" sqre n m n 0 d e vt ldu u ldu u ldu work info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                         var-7 var-8 var-9 var-10 var-11 var-12 var-13
                         var-14))
        (setf info var-15))))
   (go end_label)))
(setf inode 1)
(setf ndiml (f2cl-lib:int-add inode n))
(setf ndimr (f2cl-lib:int-add ndiml n))
(setf idxq (f2cl-lib:int-add ndimr n))
(setf iwk (f2cl-lib:int-add idxq n))
(setf ncc 0)
(setf nru 0)
(setf smlszp (f2cl-lib:int-add smlsiz 1))
```

```
(setf vf 1)
(setf vl (f2cl-lib:int-add vf m))
(setf nwork1 (f2cl-lib:int-add vl m))
(setf nwork2
        (f2cl-lib:int-add nwork1 (f2cl-lib:int-mul smlszp smlszp)))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
    (dlasdt n nlvl nd
     (f2cl-lib:array-slice iwork fixnum (inode) ((1 *)))
     (f2cl-lib:array-slice iwork fixnum (ndiml) ((1 *)))
     (f2cl-lib:array-slice iwork fixnum (ndimr) ((1 *)))
     smlsiz)
  (declare (ignore var-0 var-3 var-4 var-5 var-6))
  (setf nlvl var-1)
  (setf nd var-2))
(setf ndb1 (the fixnum (truncate (+ nd 1) 2)))
(f2cl-lib:fdo (i ndb1 (f2cl-lib:int-add i 1))
              ((> i nd) nil)
  (tagbody
    (setf i1 (f2cl-lib:int-sub i 1))
    (setf ic
            (f2cl-lib:fref iwork-%data%
                           ((f2cl-lib:int-add inode i1))
                           ((1 *))
                           iwork-%offset%))
    (setf nl
            (f2cl-lib:fref iwork-%data%
                           ((f2cl-lib:int-add ndiml i1))
                           ((1 *))
                           iwork-%offset%))
    (setf nlp1 (f2cl-lib:int-add nl 1))
    (setf nr
            (f2cl-lib:fref iwork-%data%
                           ((f2cl-lib:int-add ndimr i1))
                           ((1 *))
                           iwork-%offset%))
    (setf nlf (f2cl-lib:int-sub ic nl))
    (setf nrf (f2cl-lib:int-add ic 1))
    (setf idxqi (f2cl-lib:int-sub (f2cl-lib:int-add idxq nlf) 2))
    (setf vfi (f2cl-lib:int-sub (f2cl-lib:int-add vf nlf) 1))
    (setf vli (f2cl-lib:int-sub (f2cl-lib:int-add vl nlf) 1))
    (setf sqrei 1)
    (cond
      ((= icompq 0)
       (dlaset "A" nlp1 nlp1 zero one
        (f2cl-lib:array-slice work double-float (nwork1) ((1 *)))
        smlszp)
```

```
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13 var-14 var-15)
     (dlasdq "U" sqrei nl nlp1 nru ncc
      (f2cl-lib:array-slice d double-float (nlf) ((1 *)))
      (f2cl-lib:array-slice e double-float (nlf) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork1) ((1 *)))
      smlszp
      (f2cl-lib:array-slice work double-float (nwork2) ((1 *)))
      nl
      (f2cl-lib:array-slice work double-float (nwork2) ((1 *)))
      nl
      (f2cl-lib:array-slice work double-float (nwork2) ((1 *)))
      info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12 var-13
                    var-14))
   (setf info var-15))
 (setf itemp
         (f2cl-lib:int-add nwork1 (f2cl-lib:int-mul nl smlszp)))
 (dcopy nlp1
  (f2cl-lib:array-slice work double-float (nwork1) ((1 *))) 1
  (f2cl-lib:array-slice work double-float (vfi) ((1 *))) 1)
 (dcopy nlp1
  (f2cl-lib:array-slice work double-float (itemp) ((1 *))) 1
  (f2cl-lib:array-slice work double-float (vli) ((1 *))) 1))
(t
 (dlaset "A" nl nl zero one
  (f2cl-lib:array-slice u double-float (nlf 1) ((1 ldu) (1 *)))
  ldu)
 (dlaset "A" nlp1 nlp1 zero one
  (f2cl-lib:array-slice vt double-float (nlf 1) ((1 ldu) (1 *)))
  ldu)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13 var-14 var-15)
     (dlasdq "U" sqrei nl nlp1 nl ncc
      (f2cl-lib:array-slice d double-float (nlf) ((1 *)))
      (f2cl-lib:array-slice e double-float (nlf) ((1 *)))
      (f2cl-lib:array-slice vt
                            double-float
                            (nlf 1)
                            ((1 ldu) (1 *)))
      ldu
      (f2cl-lib:array-slice u
                            double-float
```

```
                                (nlf 1)
                                ((1 ldu) (1 *)))
            ldu
            (f2cl-lib:array-slice u
                                double-float
                                (nlf 1)
                                ((1 ldu) (1 *)))
            ldu
            (f2cl-lib:array-slice work double-float (nwork1) ((1 *)))
            info)
       (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                        var-7 var-8 var-9 var-10 var-11 var-12 var-13
                        var-14))
       (setf info var-15))
     (dcopy nlp1
      (f2cl-lib:array-slice vt double-float (nlf 1) ((1 ldu) (1 *)))
      1 (f2cl-lib:array-slice work double-float (vfi) ((1 *))) 1)
     (dcopy nlp1
      (f2cl-lib:array-slice vt
                            double-float
                            (nlf nlp1)
                            ((1 ldu) (1 *)))
      1 (f2cl-lib:array-slice work double-float (vli) ((1 *))) 1)))
  (cond
    ((/= info 0)
     (go end_label)))
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                ((> j nl) nil)
    (tagbody
      (setf (f2cl-lib:fref iwork-%data%
                           ((f2cl-lib:int-add idxqi j))
                           ((1 *))
                           iwork-%offset%)
              j)))
  (cond
    ((and (= i nd) (= sqre 0))
     (setf sqrei 0))
    (t
     (setf sqrei 1)))
  (setf idxqi (f2cl-lib:int-add idxqi nlp1))
  (setf vfi (f2cl-lib:int-add vfi nlp1))
  (setf vli (f2cl-lib:int-add vli nlp1))
  (setf nrp1 (f2cl-lib:int-add nr sqrei))
  (cond
    ((= icompq 0)
     (dlaset "A" nrp1 nrp1 zero one
```

```
 (f2cl-lib:array-slice work double-float (nwork1) ((1 *)))
 smlszp)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13 var-14 var-15)
    (dlasdq "U" sqrei nr nrp1 nru ncc
     (f2cl-lib:array-slice d double-float (nrf) ((1 *)))
     (f2cl-lib:array-slice e double-float (nrf) ((1 *)))
     (f2cl-lib:array-slice work double-float (nwork1) ((1 *)))
     smlszp
     (f2cl-lib:array-slice work double-float (nwork2) ((1 *)))
     nr
     (f2cl-lib:array-slice work double-float (nwork2) ((1 *)))
     nr
     (f2cl-lib:array-slice work double-float (nwork2) ((1 *)))
     info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12 var-13
                   var-14))
  (setf info var-15))
(setf itemp
        (f2cl-lib:int-add nwork1
                          (f2cl-lib:int-mul
                           (f2cl-lib:int-sub nrp1 1)
                           smlszp)))
(dcopy nrp1
 (f2cl-lib:array-slice work double-float (nwork1) ((1 *))) 1
 (f2cl-lib:array-slice work double-float (vfi) ((1 *))) 1)
(dcopy nrp1
 (f2cl-lib:array-slice work double-float (itemp) ((1 *))) 1
 (f2cl-lib:array-slice work double-float (vli) ((1 *))) 1))
(t
 (dlaset "A" nr nr zero one
  (f2cl-lib:array-slice u double-float (nrf 1) ((1 ldu) (1 *)))
  ldu)
 (dlaset "A" nrp1 nrp1 zero one
  (f2cl-lib:array-slice vt double-float (nrf 1) ((1 ldu) (1 *)))
  ldu)
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13 var-14 var-15)
     (dlasdq "U" sqrei nr nrp1 nr ncc
      (f2cl-lib:array-slice d double-float (nrf) ((1 *)))
      (f2cl-lib:array-slice e double-float (nrf) ((1 *)))
      (f2cl-lib:array-slice vt
                            double-float
```

```
                                    (nrf 1)
                                    ((1 ldu) (1 *)))
                 ldu
                 (f2cl-lib:array-slice u
                                    double-float
                                    (nrf 1)
                                    ((1 ldu) (1 *)))
                 ldu
                 (f2cl-lib:array-slice u
                                    double-float
                                    (nrf 1)
                                    ((1 ldu) (1 *)))
                 ldu
                 (f2cl-lib:array-slice work double-float (nwork1) ((1 *)))
                 info)
               (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                var-7 var-8 var-9 var-10 var-11 var-12 var-13
                                var-14))
               (setf info var-15))
             (dcopy nrp1
              (f2cl-lib:array-slice vt double-float (nrf 1) ((1 ldu) (1 *)))
              1 (f2cl-lib:array-slice work double-float (vfi) ((1 *))) 1)
             (dcopy nrp1
              (f2cl-lib:array-slice vt
                                    double-float
                                    (nrf nrp1)
                                    ((1 ldu) (1 *)))
              1 (f2cl-lib:array-slice work double-float (vli) ((1 *))) 1)))
         (cond
           ((/= info 0)
            (go end_label)))
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j nr) nil)
           (tagbody
             (setf (f2cl-lib:fref iwork-%data%
                                  ((f2cl-lib:int-add idxqi j))
                                  ((1 *))
                                  iwork-%offset%)
                   j)))))
    (setf j (expt 2 nlvl))
    (f2cl-lib:fdo (lvl nlvl (f2cl-lib:int-add lvl (f2cl-lib:int-sub 1)))
                  ((> lvl 1) nil)
      (tagbody
        (setf lvl2 (f2cl-lib:int-sub (f2cl-lib:int-mul lvl 2) 1))
        (cond
          ((= lvl 1)
```

```
          (setf lf 1)
          (setf ll 1))
         (t
          (setf lf (expt 2 (f2cl-lib:int-sub lvl 1)))
          (setf ll (f2cl-lib:int-sub (f2cl-lib:int-mul 2 lf) 1))))
        (f2cl-lib:fdo (i lf (f2cl-lib:int-add i 1))
                      ((> i ll) nil)
          (tagbody
            (setf im1 (f2cl-lib:int-sub i 1))
            (setf ic
                    (f2cl-lib:fref iwork-%data%
                                   ((f2cl-lib:int-add inode im1))
                                   ((1 *))
                                   iwork-%offset%))
            (setf nl
                    (f2cl-lib:fref iwork-%data%
                                   ((f2cl-lib:int-add ndiml im1))
                                   ((1 *))
                                   iwork-%offset%))
            (setf nr
                    (f2cl-lib:fref iwork-%data%
                                   ((f2cl-lib:int-add ndimr im1))
                                   ((1 *))
                                   iwork-%offset%))
            (setf nlf (f2cl-lib:int-sub ic nl))
            (setf nrf (f2cl-lib:int-add ic 1))
            (cond
              ((= i ll)
               (setf sqrei sqre))
              (t
               (setf sqrei 1)))
            (setf vfi (f2cl-lib:int-sub (f2cl-lib:int-add vf nlf) 1))
            (setf vli (f2cl-lib:int-sub (f2cl-lib:int-add vl nlf) 1))
            (setf idxqi (f2cl-lib:int-sub (f2cl-lib:int-add idxq nlf) 1))
            (setf alpha (f2cl-lib:fref d-%data% (ic) ((1 *)) d-%offset%))
            (setf beta (f2cl-lib:fref e-%data% (ic) ((1 *)) e-%offset%))
            (cond
              ((= icompq 0)
               (multiple-value-bind
                     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                      var-9 var-10 var-11 var-12 var-13 var-14 var-15
                      var-16 var-17 var-18 var-19 var-20 var-21 var-22
                      var-23 var-24 var-25)
                   (dlasd6 icompq nl nr sqrei
                    (f2cl-lib:array-slice d double-float (nlf) ((1 *)))
                    (f2cl-lib:array-slice work double-float (vfi) ((1 *)))
```

```
      (f2cl-lib:array-slice work double-float (vli) ((1 *)))
      alpha beta
      (f2cl-lib:array-slice iwork
                            fixnum
                            (idxqi)
                            ((1 *)))
      perm
      (f2cl-lib:fref givptr-%data%
                     (1)
                     ((1 *))
                     givptr-%offset%)
      givcol ldgcol givnum ldu poles difl difr z
      (f2cl-lib:fref k-%data% (1) ((1 *)) k-%offset%)
      (f2cl-lib:fref c-%data% (1) ((1 *)) c-%offset%)
      (f2cl-lib:fref s-%data% (1) ((1 *)) s-%offset%)
      (f2cl-lib:array-slice work
                            double-float
                            (nwork1)
                            ((1 *)))
      (f2cl-lib:array-slice iwork
                            fixnum
                            (iwk)
                            ((1 *)))
      info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                     var-9 var-10 var-12 var-13 var-14 var-15
                     var-16 var-17 var-18 var-19 var-23
                     var-24))
    (setf alpha var-7)
    (setf beta var-8)
    (setf (f2cl-lib:fref givptr-%data%
                         (1)
                         ((1 *))
                         givptr-%offset%)
          var-11)
    (setf (f2cl-lib:fref k-%data% (1) ((1 *)) k-%offset%)
          var-20)
    (setf (f2cl-lib:fref c-%data% (1) ((1 *)) c-%offset%)
          var-21)
    (setf (f2cl-lib:fref s-%data% (1) ((1 *)) s-%offset%)
          var-22)
    (setf info var-25)))
  (t
   (setf j (f2cl-lib:int-sub j 1))
   (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
```

```
  var-9 var-10 var-11 var-12 var-13 var-14 var-15
  var-16 var-17 var-18 var-19 var-20 var-21 var-22
  var-23 var-24 var-25)
(dlasd6 icompq nl nr sqrei
 (f2cl-lib:array-slice d double-float (nlf) ((1 *)))
 (f2cl-lib:array-slice work double-float (vfi) ((1 *)))
 (f2cl-lib:array-slice work double-float (vli) ((1 *)))
 alpha beta
 (f2cl-lib:array-slice iwork
                       fixnum
                       (idxqi)
                       ((1 *)))
 (f2cl-lib:array-slice perm
                       fixnum
                       (nlf lvl)
                       ((1 ldgcol) (1 *)))
 (f2cl-lib:fref givptr-%data%
                (j)
                ((1 *))
                givptr-%offset%)
 (f2cl-lib:array-slice givcol
                       fixnum
                       (nlf lvl2)
                       ((1 ldgcol) (1 *)))
 ldgcol
 (f2cl-lib:array-slice givnum
                       double-float
                       (nlf lvl2)
                       ((1 ldu) (1 *)))
 ldu
 (f2cl-lib:array-slice poles
                       double-float
                       (nlf lvl2)
                       ((1 ldu) (1 *)))
 (f2cl-lib:array-slice difl
                       double-float
                       (nlf lvl)
                       ((1 ldu) (1 *)))
 (f2cl-lib:array-slice difr
                       double-float
                       (nlf lvl2)
                       ((1 ldu) (1 *)))
 (f2cl-lib:array-slice z
                       double-float
                       (nlf lvl)
                       ((1 ldu) (1 *)))
```

```
                        (f2cl-lib:fref k-%data% (j) ((1 *)) k-%offset%)
                        (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%)
                        (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%)
                        (f2cl-lib:array-slice work
                                              double-float
                                              (nwork1)
                                              ((1 *)))
                        (f2cl-lib:array-slice iwork
                                              fixnum
                                              (iwk)
                                              ((1 *)))
                        info)
                    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                     var-9 var-10 var-12 var-13 var-14 var-15
                                     var-16 var-17 var-18 var-19 var-23
                                     var-24))
                    (setf alpha var-7)
                    (setf beta var-8)
                    (setf (f2cl-lib:fref givptr-%data%
                                         (j)
                                         ((1 *))
                                         givptr-%offset%)
                            var-11)
                    (setf (f2cl-lib:fref k-%data% (j) ((1 *)) k-%offset%)
                            var-20)
                    (setf (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%)
                            var-21)
                    (setf (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%)
                            var-22)
                    (setf info var-25))))
                (cond
                  ((/= info 0)
                   (go end_label)))))))
end_label
       (return
        (values nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
```

```
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                nil
                info))))))
```

7.64 dlasdq LAPACK

⟨*dlasdq.input*⟩≡

```
)set break resume
)sys rm -f dlasdq.output
)spool dlasdq.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasdq.help*⟩≡

```
======================================================================
dlasdq examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DLASDQ  -  the  singular  value decomposition (SVD) of a real (upper or
       lower) bidiagonal matrix with diagonal D and offdiagonal E,  accumulat-
       ing the transformations if desired

SYNOPSIS
       SUBROUTINE DLASDQ( UPLO,  SQRE,  N,  NCVT, NRU, NCC, D, E, VT, LDVT, U,
                          LDU, C, LDC, WORK, INFO )

           CHARACTER      UPLO

           INTEGER        INFO, LDC, LDU, LDVT, N, NCC, NCVT, NRU, SQRE

           DOUBLE         PRECISION C( LDC, * ), D( * ), E( * ), U( LDU, *  ),
                          VT( LDVT, * ), WORK( * )

PURPOSE
       DLASDQ computes the singular value decomposition (SVD) of a real (upper
       or lower) bidiagonal matrix with diagonal D and offdiagonal E,  accumu-
       lating the transformations if desired. Letting B denote the input bidi-
       agonal matrix, the algorithm computes orthogonal matrices Q and P  such
       that  B = Q * S * P' (P' denotes the transpose of P). The singular val-
       ues S are overwritten on D.

       The input matrix U  is changed to U  * Q  if desired.
       The input matrix VT is changed to P' * VT if desired.
       The input matrix C  is changed to Q' * C  if desired.

       See "Computing  Small Singular Values of Bidiagonal Matrices With Guar-
       anteed High Relative Accuracy," by J. Demmel and W. Kahan, LAPACK Work-
       ing Note #3, for a detailed description of the algorithm.


ARGUMENTS
       UPLO  (input) CHARACTER*1
             On entry, UPLO specifies whether the input bidiagonal  matrix  is
             upper or lower bidiagonal, and wether it is square are not.  UPLO
```

= 'U' or 'u' B is upper bidiagonal. UPLO = 'L' or 'l' B is lower bidiagonal.

SQRE (input) INTEGER
= 0: then the input matrix is N-by-N.
= 1: then the input matrix is N-by-(N+1) if UPLU = 'U' and (N+1)-by-N if UPLU = 'L'.

The bidiagonal matrix has N = NL + NR + 1 rows and M = N + SQRE >= N columns.

N (input) INTEGER
On entry, N specifies the number of rows and columns in the matrix. N must be at least 0.

NCVT (input) INTEGER
On entry, NCVT specifies the number of columns of the matrix VT. NCVT must be at least 0.

NRU (input) INTEGER
On entry, NRU specifies the number of rows of the matrix U. NRU must be at least 0.

NCC (input) INTEGER
On entry, NCC specifies the number of columns of the matrix C. NCC must be at least 0.

D (input/output) DOUBLE PRECISION array, dimension (N)
On entry, D contains the diagonal entries of the bidiagonal matrix whose SVD is desired. On normal exit, D contains the singular values in ascending order.

E (input/output) DOUBLE PRECISION array.
dimension is (N-1) if SQRE = 0 and N if SQRE = 1. On entry, the entries of E contain the offdiagonal entries of the bidiagonal matrix whose SVD is desired. On normal exit, E will contain 0. If the algorithm does not converge, D and E will contain the diagonal and superdiagonal entries of a bidiagonal matrix orthogonally equivalent to the one given as input.

VT (input/output) DOUBLE PRECISION array, dimension (LDVT, NCVT)
On entry, contains a matrix which on exit has been premultiplied by P', dimension N-by-NCVT if SQRE = 0 and (N+1)-by-NCVT if SQRE = 1 (not referenced if NCVT=0).

LDVT (input) INTEGER

```
      On entry, LDVT specifies the leading dimension of VT as  declared
      in the calling (sub) program. LDVT must be at least 1. If NCVT is
      nonzero LDVT must also be at least N.

U     (input/output) DOUBLE PRECISION array, dimension (LDU, N)
      On entry, contains a  matrix which on exit  has  been  postmulti-
      plied  by  Q,  dimension NRU-by-N if SQRE = 0 and NRU-by-(N+1) if
      SQRE = 1 (not referenced if NRU=0).

LDU   (input) INTEGER
      On entry, LDU  specifies the leading dimension of U  as  declared
      in  the calling (sub) program. LDU must be at least max( 1, NRU )
      .

C     (input/output) DOUBLE PRECISION array, dimension (LDC, NCC)
      On entry, contains an N-by-NCC matrix which on exit has been pre-
      multiplied by Q'  dimension N-by-NCC if SQRE = 0 and (N+1)-by-NCC
      if SQRE = 1 (not referenced if NCC=0).

LDC   (input) INTEGER
      On entry, LDC  specifies the leading dimension of C  as  declared
      in  the  calling (sub) program. LDC must be at least 1. If NCC is
      nonzero, LDC must also be at least N.

WORK  (workspace) DOUBLE PRECISION array, dimension (4*N)
      Workspace. Only referenced  if  one  of  NCVT,  NRU,  or  NCC  is
      nonzero, and if N is at least 2.

INFO  (output) INTEGER
      On  exit, a value of 0 indicates a successful exit.  If INFO < 0,
      argument number -INFO is illegal.  If INFO > 0, the algorithm did
      not  converge, and INFO specifies how many superdiagonals did not
      converge.
```

⟨*LAPACK dlasdq*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dlasdq (uplo sqre n ncvt nru ncc d e vt ldvt u ldu c ldc work info)
    (declare (type (simple-array double-float (*)) work c u vt e d)
             (type fixnum info ldc ldu ldvt ncc nru ncvt n sqre)
             (type character uplo))
    (f2cl-lib:with-multi-array-data
        ((uplo character uplo-%data% uplo-%offset%)
         (d double-float d-%data% d-%offset%)
         (e double-float e-%data% e-%offset%)
         (vt double-float vt-%data% vt-%offset%)
         (u double-float u-%data% u-%offset%)
         (c double-float c-%data% c-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((cs 0.0) (r 0.0) (smin 0.0) (sn 0.0) (i 0) (isub 0) (iuplo 0)
             (j 0) (np1 0) (sqre1 0) (rotate nil))
        (declare (type (double-float) cs r smin sn)
                 (type fixnum i isub iuplo j np1 sqre1)
                 (type (member t nil) rotate))
        (setf info 0)
        (setf iuplo 0)
        (if (char-equal uplo #\U) (setf iuplo 1))
        (if (char-equal uplo #\L) (setf iuplo 2))
        (cond
          ((= iuplo 0)
           (setf info -1))
          ((or (< sqre 0) (> sqre 1))
           (setf info -2))
          ((< n 0)
           (setf info -3))
          ((< ncvt 0)
           (setf info -4))
          ((< nru 0)
           (setf info -5))
          ((< ncc 0)
           (setf info -6))
          ((or (and (= ncvt 0) (< ldvt 1))
               (and (> ncvt 0)
                    (< ldvt
                       (max (the fixnum 1)
                            (the fixnum n)))))
           (setf info -10))
          ((< ldu (max (the fixnum 1) (the fixnum nru)))
           (setf info -12))
          ((or (and (= ncc 0) (< ldc 1))
```

```
       (and (> ncc 0)
            (< ldc
               (max (the fixnum 1)
                    (the fixnum n)))))
   (setf info -14)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DLASDQ" (f2cl-lib:int-sub info))
   (go end_label)))
(if (= n 0) (go end_label))
(setf rotate (or (> ncvt 0) (> nru 0) (> ncc 0)))
(setf np1 (f2cl-lib:int-add n 1))
(setf sqre1 sqre)
(cond
  ((and (= iuplo 1) (= sqre1 1))
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                 ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
     (tagbody
       (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
           (dlartg (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
            (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%) cs sn r)
         (declare (ignore var-0 var-1))
         (setf cs var-2)
         (setf sn var-3)
         (setf r var-4))
       (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) r)
       (setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
               (* sn
                  (f2cl-lib:fref d-%data%
                                 ((f2cl-lib:int-add i 1))
                                 ((1 *))
                                 d-%offset%)))
       (setf (f2cl-lib:fref d-%data%
                            ((f2cl-lib:int-add i 1))
                            ((1 *))
                            d-%offset%)
               (* cs
                  (f2cl-lib:fref d-%data%
                                 ((f2cl-lib:int-add i 1))
                                 ((1 *))
                                 d-%offset%)))
       (cond
         (rotate
          (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
```

```
                    cs)
              (setf (f2cl-lib:fref work-%data%
                                   ((f2cl-lib:int-add n i))
                                   ((1 *))
                                   work-%offset%)
                    sn)))))
      (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
          (dlartg (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
           (f2cl-lib:fref e-%data% (n) ((1 *)) e-%offset%) cs sn r)
        (declare (ignore var-0 var-1))
        (setf cs var-2)
        (setf sn var-3)
        (setf r var-4))
      (setf (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%) r)
      (setf (f2cl-lib:fref e-%data% (n) ((1 *)) e-%offset%) zero)
      (cond
        (rotate
         (setf (f2cl-lib:fref work-%data% (n) ((1 *)) work-%offset%) cs)
         (setf (f2cl-lib:fref work-%data%
                              ((f2cl-lib:int-add n n))
                              ((1 *))
                              work-%offset%)
                sn)))
      (setf iuplo 2)
      (setf sqre1 0)
      (if (> ncvt 0)
          (dlasr "L" "V" "F" np1 ncvt
           (f2cl-lib:array-slice work double-float (1) ((1 *)))
           (f2cl-lib:array-slice work double-float (np1) ((1 *))) vt
           ldvt))))
  (cond
    ((= iuplo 2)
     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                   ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
       (tagbody
         (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
             (dlartg (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
              (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%) cs sn r)
           (declare (ignore var-0 var-1))
           (setf cs var-2)
           (setf sn var-3)
           (setf r var-4))
         (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) r)
         (setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
                 (* sn
                    (f2cl-lib:fref d-%data%
```

```
                                    ((f2cl-lib:int-add i 1))
                                    ((1 *))
                                    d-%offset%)))
     (setf (f2cl-lib:fref d-%data%
                          ((f2cl-lib:int-add i 1))
                          ((1 *))
                          d-%offset%)
            (* cs
               (f2cl-lib:fref d-%data%
                              ((f2cl-lib:int-add i 1))
                              ((1 *))
                              d-%offset%)))
     (cond
       (rotate
        (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
              cs)
        (setf (f2cl-lib:fref work-%data%
                             ((f2cl-lib:int-add n i))
                             ((1 *))
                             work-%offset%)
              sn)))))
(cond
  ((= sqre1 1)
   (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
       (dlartg (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
        (f2cl-lib:fref e-%data% (n) ((1 *)) e-%offset%) cs sn r)
     (declare (ignore var-0 var-1))
     (setf cs var-2)
     (setf sn var-3)
     (setf r var-4))
   (setf (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%) r)
   (cond
     (rotate
      (setf (f2cl-lib:fref work-%data% (n) ((1 *)) work-%offset%) cs)
      (setf (f2cl-lib:fref work-%data%
                           ((f2cl-lib:int-add n n))
                           ((1 *))
                           work-%offset%)
            sn)))))
(cond
  ((> nru 0)
   (cond
     ((= sqre1 0)
      (dlasr "R" "V" "F" nru n
       (f2cl-lib:array-slice work double-float (1) ((1 *)))
       (f2cl-lib:array-slice work double-float (np1) ((1 *))) u ldu))
```

```
          (t
           (dlasr "R" "V" "F" nru np1
            (f2cl-lib:array-slice work double-float (1) ((1 *)))
            (f2cl-lib:array-slice work double-float (np1) ((1 *))) u
            ldu)))))
     (cond
       ((> ncc 0)
        (cond
          ((= sqre1 0)
           (dlasr "L" "V" "F" n ncc
            (f2cl-lib:array-slice work double-float (1) ((1 *)))
            (f2cl-lib:array-slice work double-float (np1) ((1 *))) c ldc))
          (t
           (dlasr "L" "V" "F" np1 ncc
            (f2cl-lib:array-slice work double-float (1) ((1 *)))
            (f2cl-lib:array-slice work double-float (np1) ((1 *))) c
            ldc)))))))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
       var-10 var-11 var-12 var-13 var-14)
    (dbdsqr "U" n ncvt nru ncc d e vt ldvt u ldu c ldc work info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                   var-8 var-9 var-10 var-11 var-12 var-13))
  (setf info var-14))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i n) nil)
  (tagbody
    (setf isub i)
    (setf smin (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
    (f2cl-lib:fdo (j (f2cl-lib:int-add i 1) (f2cl-lib:int-add j 1))
                  ((> j n) nil)
      (tagbody
        (cond
          ((< (f2cl-lib:fref d (j) ((1 *))) smin)
           (setf isub j)
           (setf smin
                   (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))))))
    (cond
      ((/= isub i)
       (setf (f2cl-lib:fref d-%data% (isub) ((1 *)) d-%offset%)
             (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
       (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) smin)
       (if (> ncvt 0)
           (dswap ncvt
            (f2cl-lib:array-slice vt
                                  double-float
```

```
                                        (isub 1)
                                        ((1 ldvt) (1 *)))
                     ldvt
                     (f2cl-lib:array-slice vt
                                        double-float
                                        (i 1)
                                        ((1 ldvt) (1 *)))
                     ldvt))
                (if (> nru 0)
                    (dswap nru
                     (f2cl-lib:array-slice u
                                        double-float
                                        (1 isub)
                                        ((1 ldu) (1 *)))
                     1
                     (f2cl-lib:array-slice u double-float (1 i) ((1 ldu) (1 *)))
                     1))
                (if (> ncc 0)
                    (dswap ncc
                     (f2cl-lib:array-slice c
                                        double-float
                                        (isub 1)
                                        ((1 ldc) (1 *)))
                     ldc
                     (f2cl-lib:array-slice c double-float (i 1) ((1 ldc) (1 *)))
                     ldc))))))
  end_label
         (return
          (values nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  nil
                  info))))))
```

7.65 dlasdt LAPACK

⟨*dlasdt.input*⟩≡

```
)set break resume
)sys rm -f dlasdt.output
)spool dlasdt.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasdt.help*⟩≡

```
  ==========================================================================
  dlasdt examples
  ==========================================================================

  ==========================================================================
  Man Page Details
  ==========================================================================

  NAME
         DLASDT - a tree of subproblems for bidiagonal divide and conquer

  SYNOPSIS
         SUBROUTINE DLASDT( N, LVL, ND, INODE, NDIML, NDIMR, MSUB )

             INTEGER        LVL, MSUB, N, ND

             INTEGER        INODE( * ), NDIML( * ), NDIMR( * )

  PURPOSE
         DLASDT creates a tree of subproblems for bidiagonal divide and conquer.


  ARGUMENTS
         N      (input) INTEGER
                On entry, the number of  diagonal  elements  of  the  bidiagonal
                matrix.

         LVL    (output) INTEGER
                On exit, the number of levels on the computation tree.

         ND     (output) INTEGER
                On exit, the number of nodes on the tree.

         INODE  (output) INTEGER array, dimension ( N )
                On exit, centers of subproblems.

         NDIML  (output) INTEGER array, dimension ( N )
                On exit, row dimensions of left children.

         NDIMR  (output) INTEGER array, dimension ( N )
                On exit, row dimensions of right children.

         MSUB   (input) INTEGER.
                On  entry, the maximum row dimension each subproblem at the bot-
                tom of the tree can be of.
```

⟨*LAPACK dlasdt*⟩≡

```
(let* ((two 2.0))
  (declare (type (double-float 2.0 2.0) two))
  (defun dlasdt (n lvl nd inode ndiml ndimr msub)
    (declare (type (simple-array fixnum (*)) ndimr ndiml inode)
             (type fixnum msub nd lvl n))
    (f2cl-lib:with-multi-array-data
        ((inode fixnum inode-%data% inode-%offset%)
         (ndiml fixnum ndiml-%data% ndiml-%offset%)
         (ndimr fixnum ndimr-%data% ndimr-%offset%))
      (prog ((temp 0.0) (i 0) (il 0) (ir 0) (llst 0) (maxn 0) (ncrnt 0)
             (nlvl 0))
        (declare (type (double-float) temp)
                 (type fixnum i il ir llst maxn ncrnt nlvl))
        (setf maxn (max (the fixnum 1) (the fixnum n)))
        (setf temp
         (/
          (f2cl-lib:flog
           (/ (coerce (realpart maxn) 'double-float)
              (coerce (realpart (f2cl-lib:int-add msub 1)) 'double-float)))
          (f2cl-lib:flog two)))
        (setf lvl (f2cl-lib:int-add (f2cl-lib:int temp) 1))
        (setf i (the fixnum (truncate n 2)))
        (setf (f2cl-lib:fref inode-%data% (1) ((1 *)) inode-%offset%)
                (f2cl-lib:int-add i 1))
        (setf (f2cl-lib:fref ndiml-%data% (1) ((1 *)) ndiml-%offset%) i)
        (setf (f2cl-lib:fref ndimr-%data% (1) ((1 *)) ndimr-%offset%)
                (f2cl-lib:int-sub n i 1))
        (setf il 0)
        (setf ir 1)
        (setf llst 1)
        (f2cl-lib:fdo (nlvl 1 (f2cl-lib:int-add nlvl 1))
                      ((> nlvl (f2cl-lib:int-add lvl (f2cl-lib:int-sub 1))) nil)
          (tagbody
            (f2cl-lib:fdo (i 0 (f2cl-lib:int-add i 1))
                          ((> i (f2cl-lib:int-add llst (f2cl-lib:int-sub 1)))
                           nil)
              (tagbody
                (setf il (f2cl-lib:int-add il 2))
                (setf ir (f2cl-lib:int-add ir 2))
                (setf ncrnt (f2cl-lib:int-add llst i))
                (setf (f2cl-lib:fref ndiml-%data% (il) ((1 *)) ndiml-%offset%)
                        (the fixnum
                             (truncate
                              (f2cl-lib:fref ndiml-%data%
                                             (ncrnt)
```

```
                                ((1 *))
                                ndiml-%offset%)
                     2)))
          (setf (f2cl-lib:fref ndimr-%data% (il) ((1 *)) ndimr-%offset%)
                (f2cl-lib:int-sub
                 (f2cl-lib:fref ndiml-%data%
                                (ncrnt)
                                ((1 *))
                                ndiml-%offset%)
                 (f2cl-lib:fref ndiml-%data%
                                (il)
                                ((1 *))
                                ndiml-%offset%)
                 1))
          (setf (f2cl-lib:fref inode-%data% (il) ((1 *)) inode-%offset%)
                (f2cl-lib:int-sub
                 (f2cl-lib:fref inode-%data%
                                (ncrnt)
                                ((1 *))
                                inode-%offset%)
                 (f2cl-lib:fref ndimr-%data%
                                (il)
                                ((1 *))
                                ndimr-%offset%)
                 1))
          (setf (f2cl-lib:fref ndiml-%data% (ir) ((1 *)) ndiml-%offset%)
                (the fixnum
                     (truncate
                      (f2cl-lib:fref ndimr-%data%
                                     (ncrnt)
                                     ((1 *))
                                     ndimr-%offset%)
                      2)))
          (setf (f2cl-lib:fref ndimr-%data% (ir) ((1 *)) ndimr-%offset%)
                (f2cl-lib:int-sub
                 (f2cl-lib:fref ndimr-%data%
                                (ncrnt)
                                ((1 *))
                                ndimr-%offset%)
                 (f2cl-lib:fref ndiml-%data%
                                (ir)
                                ((1 *))
                                ndiml-%offset%)
                 1))
          (setf (f2cl-lib:fref inode-%data% (ir) ((1 *)) inode-%offset%)
                (f2cl-lib:int-add
```

```
                          (f2cl-lib:fref inode-%data%
                                         (ncrnt)
                                         ((1 *))
                                         inode-%offset%)
                          (f2cl-lib:fref ndiml-%data%
                                         (ir)
                                         ((1 *))
                                         ndiml-%offset%)
                          1))))
              (setf llst (f2cl-lib:int-mul llst 2))))
         (setf nd (f2cl-lib:int-sub (f2cl-lib:int-mul llst 2) 1))
   end_label
         (return (values nil lvl nd nil nil nil nil))))))
```

7.66 dlaset LAPACK

⟨*dlaset.input*⟩≡

```
  )set break resume
  )sys rm -f dlaset.output
  )spool dlaset.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dlaset.help*⟩≡

```
  ======================================================================
  dlaset examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLASET  -  an  m-by-n matrix A to BETA on the diagonal and ALPHA on the
         offdiagonals

  SYNOPSIS
         SUBROUTINE DLASET( UPLO, M, N, ALPHA, BETA, A, LDA )

             CHARACTER      UPLO

             INTEGER        LDA, M, N

             DOUBLE         PRECISION ALPHA, BETA

             DOUBLE         PRECISION A( LDA, * )

  PURPOSE
         DLASET initializes an m-by-n matrix A to BETA on the diagonal and ALPHA
         on the offdiagonals.


  ARGUMENTS
         UPLO    (input) CHARACTER*1
                 Specifies  the  part  of  the  matrix  A  to  be  set.   = 'U':
                 Upper triangular part is set;  the  strictly  lower  triangular
                 part of A is not changed.  = 'L':      Lower triangular part is
                 set; the strictly upper triangular part of A  is  not  changed.
                 Otherwise:  All of the matrix A is set.

         M       (input) INTEGER
                 The number of rows of the matrix A.  M >= 0.

         N       (input) INTEGER
                 The number of columns of the matrix A.  N >= 0.

         ALPHA   (input) DOUBLE PRECISION
                 The constant to which the offdiagonal elements are to be set.
```

```
BETA    (input) DOUBLE PRECISION
        The constant to which the diagonal elements are to be set.

A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
        On exit, the leading m-by-n submatrix of A is set as follows:

        if  UPLO  =  'U', A(i,j) = ALPHA, 1<=i<=j-1, 1<=j<=n, if UPLO =
        'L', A(i,j) = ALPHA, j+1<=i<=m, 1<=j<=n, otherwise,      A(i,j)
        = ALPHA, 1<=i<=m, 1<=j<=n, i.ne.j,

        and, for all UPLO, A(i,i) = BETA, 1<=i<=min(m,n).

LDA     (input) INTEGER
        The leading dimension of the array A.  LDA >= max(1,M).
```

⟨*LAPACK dlaset*⟩≡

```
(defun dlaset (uplo m n alpha beta a lda)
  (declare (type (simple-array double-float (*)) a)
           (type (double-float) beta alpha)
           (type fixnum lda n m)
           (type character uplo))
  (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (a double-float a-%data% a-%offset%))
    (prog ((i 0) (j 0))
      (declare (type fixnum j i))
      (cond
        ((char-equal uplo #\U)
         (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i
                               (min
                                (the fixnum
                                     (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                (the fixnum m)))
                            nil)
               (tagbody
                 (setf (f2cl-lib:fref a-%data%
                                      (i j)
                                      ((1 lda) (1 *))
                                      a-%offset%)
                       alpha))))))
        ((char-equal uplo #\L)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j
                           (min (the fixnum m)
                                (the fixnum n)))
                        nil)
           (tagbody
             (f2cl-lib:fdo (i (f2cl-lib:int-add j 1) (f2cl-lib:int-add i 1))
                           ((> i m) nil)
               (tagbody
                 (setf (f2cl-lib:fref a-%data%
                                      (i j)
                                      ((1 lda) (1 *))
                                      a-%offset%)
                       alpha))))))
        (t
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
```

```
                    ((> j n) nil)
          (tagbody
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                          ((> i m) nil)
              (tagbody
                (setf (f2cl-lib:fref a-%data%
                                     (i j)
                                     ((1 lda) (1 *))
                                     a-%offset%)
                      alpha)))))))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i
                          (min (the fixnum m)
                               (the fixnum n)))
                       nil)
          (tagbody
           (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%) beta)))
        (return (values nil nil nil nil nil nil nil)))))
```

7.67 dlasq1 LAPACK

⟨*dlasq1.input*⟩≡

```
)set break resume
)sys rm -f dlasq1.output
)spool dlasq1.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasq1.help*⟩≡

```
  ======================================================================
  dlasq1 examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLASQ1  -  the  singular values of a real N-by-N bidiagonal matrix with
         diagonal D and off-diagonal E

  SYNOPSIS
         SUBROUTINE DLASQ1( N, D, E, WORK, INFO )

             INTEGER        INFO, N

             DOUBLE         PRECISION D( * ), E( * ), WORK( * )

  PURPOSE
         DLASQ1 computes the singular values of a real N-by-N bidiagonal  matrix
         with diagonal D and off-diagonal E. The singular values are computed to
         high relative accuracy, in the absence  of  denormalization,  underflow
         and overflow. The algorithm was first presented in

         "Accurate  singular  values  and  differential  qd algorithms" by K. V.
         Fernando and B. N. Parlett, Numer. Math., Vol-67, No. 2,  pp.  191-230,
         1994,

         and  the  present  implementation is described in "An implementation of
         the dqds Algorithm (Positive Case)", LAPACK Working Note.


  ARGUMENTS
         N     (input) INTEGER
               The number of rows and columns in the matrix. N >= 0.

         D     (input/output) DOUBLE PRECISION array, dimension (N)
               On entry, D contains the  diagonal  elements  of  the  bidiagonal
               matrix  whose SVD is desired. On normal exit, D contains the sin-
               gular values in decreasing order.

         E     (input/output) DOUBLE PRECISION array, dimension (N)
               On entry, elements E(1:N-1) contain the off-diagonal elements  of
               the  bidiagonal matrix whose SVD is desired.  On exit, E is over-
```

```
      written.

WORK  (workspace) DOUBLE PRECISION array, dimension (4*N)

INFO  (output) INTEGER
      = 0: successful exit
      < 0: if INFO = -i, the i-th argument had an illegal value
      > 0: the algorithm failed = 1, a split was marked by  a  positive
      value  in  E  = 2, current block of Z not diagonalized after 30*N
      iterations (in inner while loop) = 3,  termination  criterion  of
      outer  while  loop not met (program created more than N unreduced
      blocks)
```

⟨*LAPACK dlasq1*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dlasq1 (n d e work info)
    (declare (type (simple-array double-float (*)) work e d)
             (type fixnum info n))
    (f2cl-lib:with-multi-array-data
        ((d double-float d-%data% d-%offset%)
         (e double-float e-%data% e-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((eps 0.0) (scale 0.0) (safmin 0.0) (sigmn 0.0) (sigmx 0.0) (i 0)
             (iinfo 0))
        (declare (type (double-float) eps scale safmin sigmn sigmx)
                 (type fixnum i iinfo))
        (setf info 0)
        (cond
          ((< n 0)
           (setf info -2)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DLASQ1" (f2cl-lib:int-sub info))
           (go end_label))
          ((= n 0)
           (go end_label))
          ((= n 1)
           (setf (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)
                   (abs (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)))
           (go end_label))
          ((= n 2)
           (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
               (dlas2 (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)
                (f2cl-lib:fref e-%data% (1) ((1 *)) e-%offset%)
                (f2cl-lib:fref d-%data% (2) ((1 *)) d-%offset%) sigmn sigmx)
             (declare (ignore var-0 var-1 var-2))
             (setf sigmn var-3)
             (setf sigmx var-4))
           (setf (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%) sigmx)
           (setf (f2cl-lib:fref d-%data% (2) ((1 *)) d-%offset%) sigmn)
           (go end_label)))
        (setf sigmx zero)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
          (tagbody
            (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                    (abs (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)))
            (setf sigmx
```

```
            (max sigmx
                 (abs
                  (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%))))))
(setf (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
        (abs (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)))
(cond
  ((= sigmx zero)
   (multiple-value-bind (var-0 var-1 var-2 var-3)
       (dlasrt "D" n d iinfo)
     (declare (ignore var-0 var-1 var-2))
     (setf iinfo var-3))
   (go end_label)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i n) nil)
  (tagbody
    (setf sigmx
            (max sigmx
                 (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)))))
(setf eps (dlamch "Precision"))
(setf safmin (dlamch "Safe minimum"))
(setf scale (f2cl-lib:fsqrt (/ eps safmin)))
(dcopy n d 1 (f2cl-lib:array-slice work double-float (1) ((1 *))) 2)
(dcopy (f2cl-lib:int-sub n 1) e 1
 (f2cl-lib:array-slice work double-float (2) ((1 *))) 2)
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
    (dlascl "G" 0 0 sigmx scale
     (f2cl-lib:int-sub (f2cl-lib:int-mul 2 n) 1) 1 work
     (f2cl-lib:int-sub (f2cl-lib:int-mul 2 n) 1) iinfo)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                   var-8))
  (setf iinfo var-9))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i
                  (f2cl-lib:int-add (f2cl-lib:int-mul 2 n)
                                    (f2cl-lib:int-sub 1)))
               nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
            (expt (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
                  2))))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-mul 2 n))
                     ((1 *))
                     work-%offset%)
        zero)
```

```
          (multiple-value-bind (var-0 var-1 var-2)
              (dlasq2 n work info)
            (declare (ignore var-0 var-1))
            (setf info var-2))
          (cond
            ((= info 0)
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i n) nil)
               (tagbody
                (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                       (f2cl-lib:fsqrt
                        (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)))))
             (multiple-value-bind
                   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
                 (dlascl "G" 0 0 scale sigmx n 1 d n iinfo)
               (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                                var-8))
               (setf iinfo var-9))))
  end_label
          (return (values nil nil nil nil info))))))
```

7.68 dlasq2 LAPACK

⟨*dlasq2.input*⟩≡

```
)set break resume
)sys rm -f dlasq2.output
)spool dlasq2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasq2.help*⟩≡

```
==========================================================================
dlasq2 examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

NAME
       DLASQ2  - all the eigenvalues of the symmetric positive definite tridi-
       agonal matrix associated with the qd array Z to high relative  accuracy
       are  computed to high relative accuracy, in the absence of denormaliza-
       tion, underflow and overflow

SYNOPSIS
       SUBROUTINE DLASQ2( N, Z, INFO )

           INTEGER        INFO, N

           DOUBLE         PRECISION Z( * )

PURPOSE
       DLASQ2 computes all the eigenvalues of the symmetric positive  definite
       tridiagonal  matrix  associated  with  the  qd array Z to high relative
       accuracy are computed to high relative  accuracy,  in  the  absence  of
       denormalization, underflow and overflow.

       To  see  the  relation  of Z to the tridiagonal matrix, let L be a unit
       lower bidiagonal matrix with subdiagonals Z(2,4,6,,..) and let U be  an
       upper  bidiagonal  matrix with 1's above and diagonal Z(1,3,5,,..). The
       tridiagonal is L*U or, if you  prefer,  the  symmetric  tridiagonal  to
       which it is similar.

       Note  :  DLASQ2  defines  a  logical  variable,  IEEE, which is true on
       machines which follow ieee-754 floating-point standard  in  their  han-
       dling  of  infinities  and  NaNs, and false otherwise. This variable is
       passed to DLAZQ3.


ARGUMENTS
       N     (input) INTEGER
             The number of rows and columns in the matrix. N >= 0.

       Z     (workspace) DOUBLE PRECISION array, dimension ( 4*N )
             On entry Z holds the qd array. On exit, entries 1 to N  hold  the
```

```
       eigenvalues  in decreasing order, Z( 2*N+1 ) holds the trace, and
       Z( 2*N+2 ) holds the sum of the eigenvalues. If N >  2,  then  Z(
       2*N+3  ) holds the iteration count, Z( 2*N+4 ) holds NDIVS/NIN^2,
       and Z( 2*N+5 ) holds the percentage of shifts that failed.

 INFO  (output) INTEGER
       = 0: successful exit
       < 0: if the i-th argument is a scalar and had an  illegal  value,
       then  INFO = -i, if the i-th argument is an array and the j-entry
       had an illegal value, then INFO = -(i*100+j) > 0:  the  algorithm
       failed = 1, a split was marked by a positive value in E = 2, cur-
       rent block of Z not diagonalized after 30*N iterations (in  inner
       while  loop)  =  3, termination criterion of outer while loop not
       met (program created more than N unreduced blocks)

FURTHER DETAILS
       The shifts are accumulated in SIGMA. Iteration count is in ITER.  Ping-
       pong is controlled by PP (alternates between 0 and 1).
```

⟨*LAPACK dlasq2*⟩≡

```
(let* ((cbias 1.5)
       (zero 0.0)
       (half 0.5)
       (one 1.0)
       (two 2.0)
       (four 4.0)
       (hundrd 100.0))
  (declare (type (double-float 1.5 1.5) cbias)
           (type (double-float 0.0 0.0) zero)
           (type (double-float 0.5 0.5) half)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two)
           (type (double-float 4.0 4.0) four)
           (type (double-float 100.0 100.0) hundrd))
  (defun dlasq2 (n z info)
    (declare (type (simple-array double-float (*)) z)
             (type fixnum info n))
    (f2cl-lib:with-multi-array-data
        ((z double-float z-%data% z-%offset%))
      (prog ((d 0.0) (desig 0.0) (dmin 0.0) (e 0.0) (emax 0.0) (emin 0.0)
             (eps 0.0) (oldemn 0.0) (qmax 0.0) (qmin 0.0) (s 0.0) (safmin 0.0)
             (sigma 0.0) (temp 0.0) (tol 0.0) (tol2 0.0) (zmax 0.0) (i0 0)
             (i4 0) (iinfo 0) (ipn4 0) (iter 0) (iwhila 0) (iwhilb 0) (k 0)
             (n0 0) (nbig 0) (ndiv 0) (nfail 0) (pp 0) (splt 0) (ieee nil)
             (trace$ 0.0) (t$ 0.0))
        (declare (type (double-float) t$ trace$ d desig dmin e emax emin eps
                                      oldemn qmax qmin s safmin sigma temp tol
                                      tol2 zmax)
                 (type fixnum i0 i4 iinfo ipn4 iter iwhila iwhilb
                                       k n0 nbig ndiv nfail pp splt)
                 (type (member t nil) ieee))
        (setf info 0)
        (setf eps (dlamch "Precision"))
        (setf safmin (dlamch "Safe minimum"))
        (setf tol (* eps hundrd))
        (setf tol2 (expt tol 2))
        (cond
          ((< n 0)
           (setf info -1)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DLASQ2" 1)
           (go end_label))
          ((= n 0)
           (go end_label))
```

```
((= n 1)
 (cond
   ((< (f2cl-lib:fref z (1) ((1 *))) zero)
    (setf info -201)
    (error
     " ** On entry to ~a parameter number ~a had an illegal value~%"
     "DLASQ2" 2)))
 (go end_label))
((= n 2)
 (cond
   ((or (< (f2cl-lib:fref z (2) ((1 *))) zero)
        (< (f2cl-lib:fref z (3) ((1 *))) zero))
    (setf info -2)
    (error
     " ** On entry to ~a parameter number ~a had an illegal value~%"
     "DLASQ2" 2)
    (go end_label))
   ((> (f2cl-lib:fref z (3) ((1 *))) (f2cl-lib:fref z (1) ((1 *))))
    (setf d (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%))
    (setf (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%)
            (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%))
    (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) d)))
 (setf (f2cl-lib:fref z-%data% (5) ((1 *)) z-%offset%)
         (+ (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
            (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%)
            (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%)))
 (cond
   ((> (f2cl-lib:fref z (2) ((1 *)))
       (* (f2cl-lib:fref z (3) ((1 *))) tol2))
    (setf t$
            (* half
               (+
                (- (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
                   (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%))
                (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%))))
    (setf s
            (* (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%)
               (/ (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%)
                  t$)))
    (cond
      ((<= s t$)
       (setf s
               (* (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%)
                  (/ (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%)
                     (* t$
                        (+ one (f2cl-lib:fsqrt (+ one (/ s t$)))))))))))
```

```
          (t
           (setf s
                 (* (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%)
                    (/ (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%)
                       (+ t$
                          (* (f2cl-lib:fsqrt t$)
                             (f2cl-lib:fsqrt (+ t$ s)))))))))
      (setf t$
            (+ (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
               (+ s (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%))))
      (setf (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%)
            (* (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%)
               (/ (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
                  t$)))
      (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) t$)))
   (setf (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%)
         (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%))
   (setf (f2cl-lib:fref z-%data% (6) ((1 *)) z-%offset%)
         (+ (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%)
            (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)))
   (go end_label)))
(setf (f2cl-lib:fref z-%data%
                     ((f2cl-lib:int-mul 2 n))
                     ((1 *))
                     z-%offset%)
        zero)
(setf emin (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%))
(setf qmax zero)
(setf zmax zero)
(setf d zero)
(setf e zero)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 2))
              ((> k
                  (f2cl-lib:int-mul 2
                                    (f2cl-lib:int-add n
                                                      (f2cl-lib:int-sub
                                                       1))))
               nil)
  (tagbody
    (cond
      ((< (f2cl-lib:fref z (k) ((1 *))) zero)
       (setf info (f2cl-lib:int-sub (f2cl-lib:int-add 200 k)))
       (error
        " ** On entry to ~a parameter number ~a had an illegal value~%"
        "DLASQ2" 2)
       (go end_label))
```

```
          ((< (f2cl-lib:fref z ((f2cl-lib:int-add k 1)) ((1 *))) zero)
           (setf info (f2cl-lib:int-sub (f2cl-lib:int-add 200 k 1)))
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DLASQ2" 2)
           (go end_label)))
         (setf d (+ d (f2cl-lib:fref z-%data% (k) ((1 *)) z-%offset%)))
         (setf e
                 (+ e
                    (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-add k 1))
                                   ((1 *))
                                   z-%offset%)))
         (setf qmax
                 (max qmax (f2cl-lib:fref z-%data% (k) ((1 *)) z-%offset%)))
         (setf emin
                 (min emin
                      (f2cl-lib:fref z-%data%
                                     ((f2cl-lib:int-add k 1))
                                     ((1 *))
                                     z-%offset%)))
         (setf zmax
                 (max qmax
                      zmax
                      (f2cl-lib:fref z-%data%
                                     ((f2cl-lib:int-add k 1))
                                     ((1 *))
                                     z-%offset%)))))
(cond
  ((<
    (f2cl-lib:fref z
                   ((f2cl-lib:int-add (f2cl-lib:int-mul 2 n)
                                      (f2cl-lib:int-sub 1)))
                   ((1 *)))
    zero)
   (setf info
           (f2cl-lib:int-sub
            (f2cl-lib:int-sub
             (f2cl-lib:int-add 200 (f2cl-lib:int-mul 2 n))
             1)))
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DLASQ2" 2)
   (go end_label)))
(setf d
        (+ d
```

```
           (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 n) 1))
                          ((1 *))
                          z-%offset%)))
(setf qmax
        (max qmax
             (f2cl-lib:fref z-%data%
                            ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 n)
                                               1))
                            ((1 *))
                            z-%offset%)))
(setf zmax (max qmax zmax))
(cond
  ((= e zero)
   (f2cl-lib:fdo (k 2 (f2cl-lib:int-add k 1))
                 ((> k n) nil)
     (tagbody
       (setf (f2cl-lib:fref z-%data% (k) ((1 *)) z-%offset%)
             (f2cl-lib:fref z-%data%
                            ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 k)
                                               1))
                            ((1 *))
                            z-%offset%))))
   (multiple-value-bind (var-0 var-1 var-2 var-3)
       (dlasrt "D" n z iinfo)
     (declare (ignore var-0 var-1 var-2))
     (setf iinfo var-3))
   (setf (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 n) 1))
                        ((1 *))
                        z-%offset%)
           d)
   (go end_label)))
(setf trace$ (+ d e))
(cond
  ((= trace$ zero)
   (setf (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 n) 1))
                        ((1 *))
                        z-%offset%)
           zero)
   (go end_label)))
(setf ieee
        (and (= (ilaenv 10 "DLASQ2" "N" 1 2 3 4) 1)
             (= (ilaenv 11 "DLASQ2" "N" 1 2 3 4) 1)))
(f2cl-lib:fdo (k (f2cl-lib:int-mul 2 n)
```

```
                    (f2cl-lib:int-add k (f2cl-lib:int-sub 2)))
                   ((> k 2) nil)
  (tagbody
    (setf (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-mul 2 k))
                         ((1 *))
                         z-%offset%)
          zero)
    (setf (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 k) 1))
                         ((1 *))
                         z-%offset%)
          (f2cl-lib:fref z-%data% (k) ((1 *)) z-%offset%))
    (setf (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 k) 2))
                         ((1 *))
                         z-%offset%)
          zero)
    (setf (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 k) 3))
                         ((1 *))
                         z-%offset%)
          (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub k 1))
                         ((1 *))
                         z-%offset%))))
(setf i0 1)
(setf n0 n)
(cond
  ((<
    (* cbias
       (f2cl-lib:fref z
                      ((f2cl-lib:int-add (f2cl-lib:int-mul 4 i0)
                                         (f2cl-lib:int-sub 3)))
                      ((1 *))))
    (f2cl-lib:fref z
                   ((f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
                                      (f2cl-lib:int-sub 3)))
                   ((1 *))))
   (setf ipn4 (f2cl-lib:int-mul 4 (f2cl-lib:int-add i0 n0)))
   (f2cl-lib:fdo (i4 (f2cl-lib:int-mul 4 i0) (f2cl-lib:int-add i4 4))
                 ((> i4
                     (f2cl-lib:int-mul 2
                                       (f2cl-lib:int-add i0
                                                         n0
                                                         (f2cl-lib:int-sub
```

```
                                                                    1))))
                    nil)
          (tagbody
            (setf temp
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub i4 3))
                                 ((1 *))
                                 z-%offset%))
            (setf (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub i4 3))
                                 ((1 *))
                                 z-%offset%)
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub ipn4 i4 3))
                                 ((1 *))
                                 z-%offset%))
            (setf (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub ipn4 i4 3))
                                 ((1 *))
                                 z-%offset%)
                  temp)
            (setf temp
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub i4 1))
                                 ((1 *))
                                 z-%offset%))
            (setf (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub i4 1))
                                 ((1 *))
                                 z-%offset%)
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub ipn4 i4 5))
                                 ((1 *))
                                 z-%offset%))
            (setf (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub ipn4 i4 5))
                                 ((1 *))
                                 z-%offset%)
                  temp)))))
    (setf pp 0)
    (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                  ((> k 2) nil)
      (tagbody
        (setf d
              (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-sub
```

```
                                (f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
                                                  pp)
                                3))
                               ((1 *))
                               z-%offset%))
          (f2cl-lib:fdo (i4
                         (f2cl-lib:int-add
                          (f2cl-lib:int-mul 4
                                            (f2cl-lib:int-add n0
                                                              (f2cl-lib:int-sub
                                                               1)))
                          pp)
                         (f2cl-lib:int-add i4 (f2cl-lib:int-sub 4)))
                        ((> i4 (f2cl-lib:int-add (f2cl-lib:int-mul 4 i0) pp))
                         nil)
            (tagbody
              (cond
                ((<=
                  (f2cl-lib:fref z
                                 ((f2cl-lib:int-add i4 (f2cl-lib:int-sub 1)))
                                 ((1 *)))
                  (* tol2 d))
                 (setf (f2cl-lib:fref z-%data%
                                      ((f2cl-lib:int-sub i4 1))
                                      ((1 *))
                                      z-%offset%)
                         (- zero))
                 (setf d
                         (f2cl-lib:fref z-%data%
                                        ((f2cl-lib:int-sub i4 3))
                                        ((1 *))
                                        z-%offset%)))
                (t
                 (setf d
                         (*
                          (f2cl-lib:fref z-%data%
                                         ((f2cl-lib:int-sub i4 3))
                                         ((1 *))
                                         z-%offset%)
                          (/ d
                             (+ d
                                (f2cl-lib:fref z-%data%
                                               ((f2cl-lib:int-sub i4 1))
                                               ((1 *))
                                               z-%offset%)))))))))
          (setf emin
```

```
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-add (f2cl-lib:int-mul 4 i0)
                                          pp
                                          1))
                       ((1 *))
                       z-%offset%))
(setf d
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-sub
                         (f2cl-lib:int-add (f2cl-lib:int-mul 4 i0)
                                           pp)
                         3))
                       ((1 *))
                       z-%offset%))
(f2cl-lib:fdo (i4 (f2cl-lib:int-add (f2cl-lib:int-mul 4 i0) pp)
               (f2cl-lib:int-add i4 4))
              ((> i4
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul 4
                                     (f2cl-lib:int-add n0
                                                       (f2cl-lib:int-sub
                                                        1)))
                   pp))
               nil)
  (tagbody
    (setf (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub
                           (f2cl-lib:int-add i4
                                             (f2cl-lib:int-mul -1
                                                               2
                                                               pp))
                           2))
                         ((1 *))
                         z-%offset%)
          (+ d
             (f2cl-lib:fref z-%data%
                            ((f2cl-lib:int-sub i4 1))
                            ((1 *))
                            z-%offset%)))
    (cond
      ((<=
        (f2cl-lib:fref z
                       ((f2cl-lib:int-add i4 (f2cl-lib:int-sub 1)))
                       ((1 *)))
        (* tol2 d))
       (setf (f2cl-lib:fref z-%data%
```

```
                        ((f2cl-lib:int-sub i4 1))
                        ((1 *))
                        z-%offset%)
                (- zero))
         (setf (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub
                          (f2cl-lib:int-add i4
                                            (f2cl-lib:int-mul
                                             -1
                                             2
                                             pp))
                          2))
                        ((1 *))
                        z-%offset%)
                d)
         (setf (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-add i4
                                           (f2cl-lib:int-mul -1
                                                             2
                                                             pp)))
                        ((1 *))
                        z-%offset%)
                zero)
         (setf d
                (f2cl-lib:fref z-%data%
                               ((f2cl-lib:int-add i4 1))
                               ((1 *))
                               z-%offset%)))
        ((and
          (<
           (* safmin
              (f2cl-lib:fref z ((f2cl-lib:int-add i4 1)) ((1 *))))
           (f2cl-lib:fref z
                          ((f2cl-lib:int-add i4
                                             (f2cl-lib:int-mul -1
                                                               2
                                                               pp)
                                             (f2cl-lib:int-sub 2)))
                          ((1 *))))
          (<
           (* safmin
              (f2cl-lib:fref z
                             ((f2cl-lib:int-add i4
                                                (f2cl-lib:int-mul -1
                                                                  2
                                                                  pp)
```

```
                                          (f2cl-lib:int-sub
                                           2)))
                     ((1 *))))
  (f2cl-lib:fref z ((f2cl-lib:int-add i4 1)) ((1 *)))))
 (setf temp
       (/
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-add i4 1))
                       ((1 *))
                       z-%offset%)
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-sub
                         (f2cl-lib:int-add i4
                                          (f2cl-lib:int-mul
                                           -1
                                           2
                                           pp))
                         2))
                       ((1 *))
                       z-%offset%)))
 (setf (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-add i4
                                         (f2cl-lib:int-mul -1
                                                           2
                                                           pp)))
                      ((1 *))
                      z-%offset%)
       (*
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-sub i4 1))
                       ((1 *))
                       z-%offset%)
        temp))
 (setf d (* d temp)))
(t
 (setf (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-add i4
                                         (f2cl-lib:int-mul -1
                                                           2
                                                           pp)))
                      ((1 *))
                      z-%offset%)
       (*
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-add i4 1))
                       ((1 *))
```

```
                                        z-%offset%)
                                (/
                                 (f2cl-lib:fref z-%data%
                                                ((f2cl-lib:int-sub i4 1))
                                                ((1 *))
                                                z-%offset%)
                                 (f2cl-lib:fref z-%data%
                                                ((f2cl-lib:int-sub
                                                  (f2cl-lib:int-add i4
                                                                    (f2cl-lib:int-mul
                                                                     -1
                                                                     2
                                                                     pp))
                                                  2))
                                                ((1 *))
                                                z-%offset%))))
                           (setf d
                                 (*
                                  (f2cl-lib:fref z-%data%
                                                 ((f2cl-lib:int-add i4 1))
                                                 ((1 *))
                                                 z-%offset%)
                                  (/ d
                                     (f2cl-lib:fref z-%data%
                                                    ((f2cl-lib:int-sub
                                                      (f2cl-lib:int-add i4
                                                                        (f2cl-lib:int-mul
                                                                         -1
                                                                         2
                                                                         pp))
                                                      2))
                                                    ((1 *))
                                                    z-%offset%))))))
                      (setf emin
                            (min emin
                                 (f2cl-lib:fref z-%data%
                                                ((f2cl-lib:int-add i4
                                                                   (f2cl-lib:int-mul
                                                                    -1
                                                                    2
                                                                    pp)))
                                                ((1 *))
                                                z-%offset%)))))
                (setf (f2cl-lib:fref z-%data%
                                     ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0)
                                                        pp
```

```
                                                2))
                                ((1 *))
                                z-%offset%)
                        d)
            (setf qmax
                    (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 i0)
                                                      pp
                                                      2))
                                   ((1 *))
                                   z-%offset%))
            (f2cl-lib:fdo (i4
                           (f2cl-lib:int-add (f2cl-lib:int-mul 4 i0)
                                             (f2cl-lib:int-sub pp)
                                             2)
                           (f2cl-lib:int-add i4 4))
                          ((> i4
                              (f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
                                                (f2cl-lib:int-sub pp)
                                                (f2cl-lib:int-sub 2)))
                           nil)
              (tagbody
                (setf qmax
                      (max qmax
                           (f2cl-lib:fref z-%data% (i4) ((1 *)) z-%offset%)))))
            (setf pp (f2cl-lib:int-sub 1 pp))))
        (setf iter 2)
        (setf nfail 0)
        (setf ndiv (f2cl-lib:int-mul 2 (f2cl-lib:int-sub n0 i0)))
        (f2cl-lib:fdo (iwhila 1 (f2cl-lib:int-add iwhila 1))
                      ((> iwhila (f2cl-lib:int-add n 1)) nil)
          (tagbody
            (if (< n0 1) (go label150))
            (setf desig zero)
            (cond
              ((= n0 n)
               (setf sigma zero))
              (t
               (setf sigma
                       (-
                        (f2cl-lib:fref z-%data%
                                       ((f2cl-lib:int-sub
                                         (f2cl-lib:int-mul 4 n0)
                                         1))
                                       ((1 *))
                                       z-%offset%)))))
```

```
(cond
  ((< sigma zero)
   (setf info 1)
   (go end_label)))
(setf emax zero)
(cond
  ((> n0 i0)
   (setf emin
         (abs
          (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub
                           (f2cl-lib:int-mul 4 n0)
                           5))
                         ((1 *))
                         z-%offset%))))
  (t
   (setf emin zero)))
(setf qmin
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0)
                                          3))
                       ((1 *))
                       z-%offset%))
(setf qmax qmin)
(f2cl-lib:fdo (i4 (f2cl-lib:int-mul 4 n0)
               (f2cl-lib:int-add i4 (f2cl-lib:int-sub 4)))
              ((> i4 8) nil)
  (tagbody
    (if
     (<=
      (f2cl-lib:fref z-%data%
                     ((f2cl-lib:int-sub i4 5))
                     ((1 *))
                     z-%offset%)
      zero)
     (go label100))
    (cond
      ((>= qmin (* four emax))
       (setf qmin
             (min qmin
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub i4 3))
                                 ((1 *))
                                 z-%offset%)))
       (setf emax
             (max emax
```

```
                                  (f2cl-lib:fref z-%data%
                                                 ((f2cl-lib:int-sub i4 5))
                                                 ((1 *))
                                                 z-%offset%)))))
                    (setf qmax
                            (max qmax
                                 (+
                                  (f2cl-lib:fref z-%data%
                                                 ((f2cl-lib:int-sub i4 7))
                                                 ((1 *))
                                                 z-%offset%)
                                  (f2cl-lib:fref z-%data%
                                                 ((f2cl-lib:int-sub i4 5))
                                                 ((1 *))
                                                 z-%offset%))))
                    (setf emin
                            (min emin
                                 (f2cl-lib:fref z-%data%
                                                ((f2cl-lib:int-sub i4 5))
                                                ((1 *))
                                                z-%offset%)))))
              (setf i4 4)
  label100
              (setf i0 (the fixnum (truncate i4 4)))
              (setf (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0) 1))
                                   ((1 *))
                                   z-%offset%)
                    emin)
              (setf dmin
                    (-
                     (max zero
                          (+ qmin
                             (* (- two)
                                (f2cl-lib:fsqrt qmin)
                                (f2cl-lib:fsqrt emax))))))
              (setf pp 0)
              (setf nbig
                    (f2cl-lib:int-mul 30
                                      (f2cl-lib:int-add
                                       (f2cl-lib:int-sub n0 i0)
                                       1)))
              (f2cl-lib:fdo (iwhilb 1 (f2cl-lib:int-add iwhilb 1))
                            ((> iwhilb nbig) nil)
                (tagbody
                  (if (> i0 n0) (go label130))
```

```
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11)
    (dlasq3 i0 n0 z pp dmin sigma desig qmax nfail iter ndiv
     ieee)
  (declare (ignore var-0 var-2 var-3 var-11))
  (setf n0 var-1)
  (setf dmin var-4)
  (setf sigma var-5)
  (setf desig var-6)
  (setf qmax var-7)
  (setf nfail var-8)
  (setf iter var-9)
  (setf ndiv var-10))
(setf pp (f2cl-lib:int-sub 1 pp))
(cond
  ((and (= pp 0)
        (>= (f2cl-lib:int-add n0 (f2cl-lib:int-sub i0)) 3))
   (cond
     ((or
       (<= (f2cl-lib:fref z ((f2cl-lib:int-mul 4 n0)) ((1 *)))
           (* tol2 qmax))
       (<=
        (f2cl-lib:fref z
                       ((f2cl-lib:int-add
                         (f2cl-lib:int-mul 4 n0)
                         (f2cl-lib:int-sub 1)))
                       ((1 *)))
        (* tol2 sigma)))
      (setf splt (f2cl-lib:int-sub i0 1))
      (setf qmax
              (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-sub
                               (f2cl-lib:int-mul 4 i0)
                               3))
                             ((1 *))
                             z-%offset%))
      (setf emin
              (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-sub
                               (f2cl-lib:int-mul 4 i0)
                               1))
                             ((1 *))
                             z-%offset%))
      (setf oldemn
              (f2cl-lib:fref z-%data%
```

```
                        ((f2cl-lib:int-mul 4 i0))
                        ((1 *))
                        z-%offset%))
(f2cl-lib:fdo (i4 (f2cl-lib:int-mul 4 i0)
               (f2cl-lib:int-add i4 4))
              ((> i4
                  (f2cl-lib:int-mul 4
                                    (f2cl-lib:int-add n0
                                                      (f2cl-lib
                                                       3))))
               nil)
  (tagbody
    (cond
      ((or
        (<= (f2cl-lib:fref z (i4) ((1 *)))
            (* tol2
               (f2cl-lib:fref z
                              ((f2cl-lib:int-add i4
                                                 (f2cl-lib:int-
                                                  3)))
                              ((1 *)))))
        (<=
         (f2cl-lib:fref z
                        ((f2cl-lib:int-add i4
                                           (f2cl-lib:int-sub
                                            1)))
                        ((1 *)))
         (* tol2 sigma)))
       (setf (f2cl-lib:fref z-%data%
                            ((f2cl-lib:int-sub i4 1))
                            ((1 *))
                            z-%offset%)
               (- sigma))
       (setf splt (the fixnum (truncate i4 4)))
       (setf qmax zero)
       (setf emin
               (f2cl-lib:fref z-%data%
                              ((f2cl-lib:int-add i4 3))
                              ((1 *))
                              z-%offset%))
       (setf oldemn
               (f2cl-lib:fref z-%data%
                              ((f2cl-lib:int-add i4 4))
                              ((1 *))
                              z-%offset%)))
       (t
```

```
                              (setf qmax
                                    (max qmax
                                         (f2cl-lib:fref z-%data%
                                                        ((f2cl-lib:int-add i4
                                                                           1))
                                                        ((1 *))
                                                        z-%offset%)))
                              (setf emin
                                    (min emin
                                         (f2cl-lib:fref z-%data%
                                                        ((f2cl-lib:int-sub i4
                                                                           1))
                                                        ((1 *))
                                                        z-%offset%)))
                              (setf oldemn
                                    (min oldemn
                                         (f2cl-lib:fref z-%data%
                                                        (i4)
                                                        ((1 *))
                                                        z-%offset%)))))))
                        (setf (f2cl-lib:fref z-%data%
                                             ((f2cl-lib:int-sub
                                               (f2cl-lib:int-mul 4 n0)
                                               1))
                                             ((1 *))
                                             z-%offset%)
                              emin)
                        (setf (f2cl-lib:fref z-%data%
                                             ((f2cl-lib:int-mul 4 n0))
                                             ((1 *))
                                             z-%offset%)
                              oldemn)
                        (setf i0 (f2cl-lib:int-add splt 1))))))
              (setf info 2)
              (go end_label)
  label130))
         (setf info 3)
         (go end_label)
  label150
         (f2cl-lib:fdo (k 2 (f2cl-lib:int-add k 1))
                       ((> k n) nil)
           (tagbody
             (setf (f2cl-lib:fref z-%data% (k) ((1 *)) z-%offset%)
                   (f2cl-lib:fref z-%data%
                                  ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 k)
                                                     3))
```

```
                                ((1 *))
                                z-%offset%))))
        (multiple-value-bind (var-0 var-1 var-2 var-3)
            (dlasrt "D" n z iinfo)
          (declare (ignore var-0 var-1 var-2))
          (setf iinfo var-3))
        (setf e zero)
        (f2cl-lib:fdo (k n (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
                      ((> k 1) nil)
          (tagbody
            (setf e (+ e (f2cl-lib:fref z-%data% (k) ((1 *)) z-%offset%)))))
        (setf (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-add (f2cl-lib:int-mul 2 n) 1))
                             ((1 *))
                             z-%offset%)
                trace$)
        (setf (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-add (f2cl-lib:int-mul 2 n) 2))
                             ((1 *))
                             z-%offset%)
                e)
        (setf (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-add (f2cl-lib:int-mul 2 n) 3))
                             ((1 *))
                             z-%offset%)
                (coerce (realpart iter) 'double-float))
        (setf (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-add (f2cl-lib:int-mul 2 n) 4))
                             ((1 *))
                             z-%offset%)
                (/ (coerce (realpart ndiv) 'double-float)
                   (coerce (realpart (expt n 2)) 'double-float)))
        (setf (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-add (f2cl-lib:int-mul 2 n) 5))
                             ((1 *))
                             z-%offset%)
                (/ (* hundrd nfail)
                   (coerce (realpart iter) 'double-float)))
        (go end_label)
  end_label
        (return (values nil nil info))))))
```

7.69 dlasq3 LAPACK

⟨*dlasq3.input*⟩≡

```
)set break resume
)sys rm -f dlasq3.output
)spool dlasq3.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasq3.help*⟩≡

```
  ========================================================================
  dlasq3 examples
  ========================================================================

  ========================================================================
  Man Page Details
  ========================================================================

  NAME
         DLASQ3 - for deflation, computes a shift (TAU) and calls dqds

  SYNOPSIS
         SUBROUTINE DLASQ3( I0,  N0,  Z,  PP,  DMIN,  SIGMA, DESIG, QMAX, NFAIL,
                            ITER, NDIV, IEEE )

             LOGICAL        IEEE

             INTEGER        I0, ITER, N0, NDIV, NFAIL, PP

             DOUBLE         PRECISION DESIG, DMIN, QMAX, SIGMA

             DOUBLE         PRECISION Z( * )

  PURPOSE
         DLASQ3 checks for deflation, computes a shift (TAU) and calls dqds.  In
         case of failure it changes shifts, and tries again until output is pos-
         itive.


  ARGUMENTS
         I0     (input) INTEGER
                First index.

         N0     (input) INTEGER
                Last index.

         Z      (input) DOUBLE PRECISION array, dimension ( 4*N )
                Z holds the qd array.

         PP     (input) INTEGER
                PP=0 for ping, PP=1 for pong.

         DMIN   (output) DOUBLE PRECISION
                Minimum value of d.
```

```
SIGMA  (output) DOUBLE PRECISION
       Sum of shifts used in current segment.

DESIG  (input/output) DOUBLE PRECISION
       Lower order part of SIGMA

QMAX   (input) DOUBLE PRECISION
       Maximum value of q.

NFAIL  (output) INTEGER
       Number of times shift was too big.

ITER   (output) INTEGER
       Number of iterations.

NDIV   (output) INTEGER
       Number of divisions.

TTYPE  (output) INTEGER
       Shift type.

IEEE   (input) LOGICAL
       Flag for IEEE or non IEEE arithmetic (passed to DLASQ5).
```

⟨*LAPACK dlasq3*⟩≡

```
(let* ((cbias 1.5)
       (zero 0.0)
       (qurtr 0.25)
       (half 0.5)
       (one 1.0)
       (two 2.0)
       (hundrd 100.0))
  (declare (type (double-float 1.5 1.5) cbias)
           (type (double-float 0.0 0.0) zero)
           (type (double-float 0.25 0.25) qurtr)
           (type (double-float 0.5 0.5) half)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two)
           (type (double-float 100.0 100.0) hundrd))
  (let ((ttype 0)
        (f2cl-lib:dmin1 zero)
        (dmin2 zero)
        (dn zero)
        (dn1 zero)
        (dn2 zero)
        (tau zero))
    (declare (type (double-float) tau dn2 dn1 dn dmin2 f2cl-lib:dmin1)
             (type fixnum ttype))
    (defun dlasq3 (i0 n0 z pp dmin sigma desig qmax nfail iter ndiv ieee)
      (declare (type (member t nil) ieee)
               (type (double-float) qmax desig sigma dmin)
               (type (simple-array double-float (*)) z)
               (type fixnum ndiv iter nfail pp n0 i0))
      (f2cl-lib:with-multi-array-data
          ((z double-float z-%data% z-%offset%))
        (prog ((eps 0.0) (s 0.0) (safmin 0.0) (temp 0.0) (tol 0.0) (tol2 0.0)
               (ipn4 0) (j4 0) (n0in 0) (nn 0) (t$ 0.0))
          (declare (type (double-float) t$ eps s safmin temp tol tol2)
                   (type fixnum ipn4 j4 n0in nn))
          (setf n0in n0)
          (setf eps (dlamch "Precision"))
          (setf safmin (dlamch "Safe minimum"))
          (setf tol (* eps hundrd))
          (setf tol2 (expt tol 2))
  label10
          (if (< n0 i0) (go end_label))
          (if (= n0 i0) (go label20))
          (setf nn (f2cl-lib:int-add (f2cl-lib:int-mul 4 n0) pp))
          (if (= n0 (f2cl-lib:int-add i0 1)) (go label40))
          (if
```

```
            (and
             (>
              (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-sub nn 5))
                             ((1 *))
                             z-%offset%)
              (* tol2
                 (+ sigma
                    (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-sub nn 3))
                                   ((1 *))
                                   z-%offset%))))
             (>
              (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-sub
                               (f2cl-lib:int-add nn (f2cl-lib:int-mul -1 2 pp))
                               4))
                             ((1 *))
                             z-%offset%)
              (* tol2
                 (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-sub nn 7))
                                ((1 *))
                                z-%offset%))))
             (go label30))
  label20
            (setf (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0) 3))
                                 ((1 *))
                                 z-%offset%)
                  (+
                   (f2cl-lib:fref z-%data%
                                  ((f2cl-lib:int-sub
                                    (f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
                                                      pp)
                                    3))
                                  ((1 *))
                                  z-%offset%)
                   sigma))
            (setf n0 (f2cl-lib:int-sub n0 1))
            (go label10)
  label30
            (if
             (and
              (>
               (f2cl-lib:fref z-%data%
```

```
                              ((f2cl-lib:int-sub nn 9))
                              ((1 *))
                              z-%offset%)
                 (* tol2 sigma))
                (>
                 (f2cl-lib:fref z-%data%
                              ((f2cl-lib:int-sub
                                (f2cl-lib:int-add nn (f2cl-lib:int-mul -1 2 pp))
                                8))
                              ((1 *))
                              z-%offset%)
                 (* tol2
                    (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-sub nn 11))
                                   ((1 *))
                                   z-%offset%))))
               (go label50))
  label40
           (cond
             ((>
               (f2cl-lib:fref z
                              ((f2cl-lib:int-add nn (f2cl-lib:int-sub 3)))
                              ((1 *)))
               (f2cl-lib:fref z
                              ((f2cl-lib:int-add nn (f2cl-lib:int-sub 7)))
                              ((1 *))))
              (setf s
                      (f2cl-lib:fref z-%data%
                                     ((f2cl-lib:int-sub nn 3))
                                     ((1 *))
                                     z-%offset%))
              (setf (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-sub nn 3))
                                   ((1 *))
                                   z-%offset%)
                      (f2cl-lib:fref z-%data%
                                     ((f2cl-lib:int-sub nn 7))
                                     ((1 *))
                                     z-%offset%))
              (setf (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-sub nn 7))
                                   ((1 *))
                                   z-%offset%)
                      s)))
           (cond
             ((>
```

```
 (f2cl-lib:fref z
                ((f2cl-lib:int-add nn (f2cl-lib:int-sub 5)))
                ((1 *)))
 (*
  (f2cl-lib:fref z
                 ((f2cl-lib:int-add nn (f2cl-lib:int-sub 3)))
                 ((1 *)))
  tol2))
(setf t$
        (* half
           (+
            (-
             (f2cl-lib:fref z-%data%
                            ((f2cl-lib:int-sub nn 7))
                            ((1 *))
                            z-%offset%)
             (f2cl-lib:fref z-%data%
                            ((f2cl-lib:int-sub nn 3))
                            ((1 *))
                            z-%offset%))
            (f2cl-lib:fref z-%data%
                           ((f2cl-lib:int-sub nn 5))
                           ((1 *))
                           z-%offset%))))
(setf s
        (*
         (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub nn 3))
                        ((1 *))
                        z-%offset%)
         (/
          (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub nn 5))
                         ((1 *))
                         z-%offset%)
          t$)))
(cond
  ((<= s t$)
   (setf s
           (*
            (f2cl-lib:fref z-%data%
                           ((f2cl-lib:int-sub nn 3))
                           ((1 *))
                           z-%offset%)
            (/
             (f2cl-lib:fref z-%data%
```

```
                                ((f2cl-lib:int-sub nn 5))
                                ((1 *))
                                z-%offset%)
                     (* t$ (+ one (f2cl-lib:fsqrt (+ one (/ s t$)))))))))
           (t
            (setf s
                  (*
                   (f2cl-lib:fref z-%data%
                                  ((f2cl-lib:int-sub nn 3))
                                  ((1 *))
                                  z-%offset%)
                   (/
                    (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-sub nn 5))
                                   ((1 *))
                                   z-%offset%)
                    (+ t$
                       (* (f2cl-lib:fsqrt t$)
                          (f2cl-lib:fsqrt (+ t$ s)))))))))
       (setf t$
             (+
              (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-sub nn 7))
                             ((1 *))
                             z-%offset%)
              (+ s
                 (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-sub nn 5))
                                ((1 *))
                                z-%offset%))))
       (setf (f2cl-lib:fref z-%data%
                            ((f2cl-lib:int-sub nn 3))
                            ((1 *))
                            z-%offset%)
             (*
              (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-sub nn 3))
                             ((1 *))
                             z-%offset%)
              (/
               (f2cl-lib:fref z-%data%
                              ((f2cl-lib:int-sub nn 7))
                              ((1 *))
                              z-%offset%)
               t$)))
       (setf (f2cl-lib:fref z-%data%
```

```
                                ((f2cl-lib:int-sub nn 7))
                                ((1 *))
                                z-%offset%)
                    t$)))
          (setf (f2cl-lib:fref z-%data%
                               ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0) 7))
                               ((1 *))
                               z-%offset%)
                (+
                 (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-sub nn 7))
                                ((1 *))
                                z-%offset%)
                 sigma))
          (setf (f2cl-lib:fref z-%data%
                               ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0) 3))
                               ((1 *))
                               z-%offset%)
                (+
                 (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-sub nn 3))
                                ((1 *))
                                z-%offset%)
                 sigma))
          (setf n0 (f2cl-lib:int-sub n0 2))
          (go label10)
label50
          (cond
            ((or (<= dmin zero) (< n0 n0in))
             (cond
               ((<
                 (* cbias
                    (f2cl-lib:fref z
                                   ((f2cl-lib:int-add (f2cl-lib:int-mul 4 i0)
                                                      pp
                                                      (f2cl-lib:int-sub 3)))
                                   ((1 *))))
                 (f2cl-lib:fref z
                                ((f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
                                                   pp
                                                   (f2cl-lib:int-sub 3)))
                                ((1 *))))
                (setf ipn4 (f2cl-lib:int-mul 4 (f2cl-lib:int-add i0 n0)))
                (f2cl-lib:fdo (j4 (f2cl-lib:int-mul 4 i0)
                                  (f2cl-lib:int-add j4 4))
                              ((> j4
```

```
                    (f2cl-lib:int-mul 2
                                      (f2cl-lib:int-add i0
                                                        n0
                                                        (f2cl-lib:int-s
                                                         1))))
                nil)
  (tagbody
    (setf temp
          (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub j4 3))
                         ((1 *))
                         z-%offset%))
    (setf (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub j4 3))
                         ((1 *))
                         z-%offset%)
          (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub ipn4 j4 3))
                         ((1 *))
                         z-%offset%))
    (setf (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub ipn4 j4 3))
                         ((1 *))
                         z-%offset%)
          temp)
    (setf temp
          (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub j4 2))
                         ((1 *))
                         z-%offset%))
    (setf (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub j4 2))
                         ((1 *))
                         z-%offset%)
          (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub ipn4 j4 2))
                         ((1 *))
                         z-%offset%))
    (setf (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub ipn4 j4 2))
                         ((1 *))
                         z-%offset%)
          temp)
    (setf temp
          (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub j4 1))
```

```
                                ((1 *))
                                z-%offset%))
            (setf (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub j4 1))
                                 ((1 *))
                                 z-%offset%)
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub ipn4 j4 5))
                                 ((1 *))
                                 z-%offset%))
            (setf (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub ipn4 j4 5))
                                 ((1 *))
                                 z-%offset%)
                  temp)
            (setf temp
                  (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%))
            (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub ipn4 j4 4))
                                 ((1 *))
                                 z-%offset%))
            (setf (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub ipn4 j4 4))
                                 ((1 *))
                                 z-%offset%)
                  temp)))
          (cond
            ((<= (f2cl-lib:int-add n0 (f2cl-lib:int-sub i0)) 4)
             (setf (f2cl-lib:fref z-%data%
                                  ((f2cl-lib:int-sub
                                    (f2cl-lib:int-add
                                     (f2cl-lib:int-mul 4 n0)
                                     pp)
                                    1))
                                  ((1 *))
                                  z-%offset%)
                   (f2cl-lib:fref z-%data%
                                  ((f2cl-lib:int-sub
                                    (f2cl-lib:int-add
                                     (f2cl-lib:int-mul 4 i0)
                                     pp)
                                    1))
                                  ((1 *))
                                  z-%offset%))
             (setf (f2cl-lib:fref z-%data%
```

```
                                ((f2cl-lib:int-sub
                                  (f2cl-lib:int-mul 4 n0)
                                  pp))
                                ((1 *))
                                z-%offset%)
                   (f2cl-lib:fref z-%data%
                                  ((f2cl-lib:int-sub
                                    (f2cl-lib:int-mul 4 i0)
                                    pp))
                                  ((1 *))
                                  z-%offset%))))
          (setf dmin2
                (min dmin2
                     (f2cl-lib:fref z-%data%
                                    ((f2cl-lib:int-sub
                                      (f2cl-lib:int-add
                                       (f2cl-lib:int-mul 4 n0)
                                       pp)
                                      1))
                                    ((1 *))
                                    z-%offset%)))
          (setf (f2cl-lib:fref z-%data%
                               ((f2cl-lib:int-sub
                                 (f2cl-lib:int-add
                                  (f2cl-lib:int-mul 4 n0)
                                  pp)
                                 1))
                               ((1 *))
                               z-%offset%)
                (min
                 (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-sub
                                  (f2cl-lib:int-add
                                   (f2cl-lib:int-mul 4 n0)
                                   pp)
                                  1))
                                ((1 *))
                                z-%offset%)
                 (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-sub
                                  (f2cl-lib:int-add
                                   (f2cl-lib:int-mul 4 i0)
                                   pp)
                                  1))
                                ((1 *))
                                z-%offset%)
```

```
                (f2cl-lib:fref z-%data%
                               ((f2cl-lib:int-add
                                 (f2cl-lib:int-mul 4 i0)
                                 pp
                                 3))
                               ((1 *))
                               z-%offset%)))
          (setf (f2cl-lib:fref z-%data%
                               ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0)
                                                  pp))
                               ((1 *))
                               z-%offset%)
                (min
                 (f2cl-lib:fref z-%data%
                               ((f2cl-lib:int-sub
                                 (f2cl-lib:int-mul 4 n0)
                                 pp))
                               ((1 *))
                               z-%offset%)
                 (f2cl-lib:fref z-%data%
                               ((f2cl-lib:int-sub
                                 (f2cl-lib:int-mul 4 i0)
                                 pp))
                               ((1 *))
                               z-%offset%)
                 (f2cl-lib:fref z-%data%
                               ((f2cl-lib:int-add
                                 (f2cl-lib:int-sub
                                  (f2cl-lib:int-mul 4 i0)
                                  pp)
                                 4))
                               ((1 *))
                               z-%offset%)))
          (setf qmax
                (max qmax
                     (f2cl-lib:fref z-%data%
                                    ((f2cl-lib:int-sub
                                      (f2cl-lib:int-add
                                       (f2cl-lib:int-mul 4 i0)
                                       pp)
                                      3))
                                    ((1 *))
                                    z-%offset%)
                     (f2cl-lib:fref z-%data%
                                    ((f2cl-lib:int-add
                                      (f2cl-lib:int-mul 4 i0)
```

```
                                          pp
                                          1))
                                        ((1 *))
                                        z-%offset%)))
                (setf dmin (- zero))))))
          (cond
            ((or (< dmin zero)
                 (< (* safmin qmax)
                    (min
                     (f2cl-lib:fref z
                                    ((f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
                                                       pp
                                                       (f2cl-lib:int-sub 1)))
                                    ((1 *)))
                     (f2cl-lib:fref z
                                    ((f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
                                                       pp
                                                       (f2cl-lib:int-sub 9)))
                                    ((1 *)))
                     (+ dmin2
                        (f2cl-lib:fref z
                                       ((f2cl-lib:int-add
                                         (f2cl-lib:int-mul 4 n0)
                                         (f2cl-lib:int-sub pp)))
                                       ((1 *)))))))
             (tagbody
               (multiple-value-bind
                     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                      var-9 var-10 var-11 var-12)
                   (dlasq4 i0 n0 z pp n0in dmin f2cl-lib:dmin1 dmin2 dn dn1 dn2
                    tau ttype)
                 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                  var-7 var-8 var-9 var-10))
                 (setf tau var-11)
                 (setf ttype var-12))
label80
               (multiple-value-bind
                     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                      var-9 var-10 var-11)
                   (dlasq5 i0 n0 z pp tau dmin f2cl-lib:dmin1 dmin2 dn dn1 dn2
                    ieee)
                 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-6 var-11))
                 (setf dmin var-5)
                 (setf dmin2 var-7)
                 (setf dn var-8)
                 (setf dn1 var-9)
```

```
  (setf dn2 var-10))
(setf ndiv
       (f2cl-lib:int-add ndiv
                         (f2cl-lib:int-add
                          (f2cl-lib:int-sub n0 i0)
                          2)))
(setf iter (f2cl-lib:int-add iter 1))
(cond
  ((and (>= dmin zero) (> f2cl-lib:dmin1 zero))
   (go label100))
  ((and (< dmin zero)
        (> f2cl-lib:dmin1 zero)
        (<
         (f2cl-lib:fref z
                        ((f2cl-lib:int-add
                          (f2cl-lib:int-mul 4
                                            (f2cl-lib:int-add n0
                                                              (f2cl-lib:int-sub
                                                               1)))
                          (f2cl-lib:int-sub pp)))
                        ((1 *)))
         (* tol (+ sigma dn1)))
        (< (abs dn) (* tol sigma)))
   (setf (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-add
                          (f2cl-lib:int-sub
                           (f2cl-lib:int-mul 4
                                             (f2cl-lib:int-sub
                                              n0
                                              1))
                           pp)
                          2))
                        ((1 *))
                        z-%offset%)
           zero)
   (setf dmin zero)
   (go label100))
  ((< dmin zero)
   (setf nfail (f2cl-lib:int-add nfail 1))
   (cond
     ((< ttype (f2cl-lib:int-sub 22))
      (setf tau zero))
     ((> f2cl-lib:dmin1 zero)
      (setf tau (* (+ tau dmin) (- one (* two eps))))
      (setf ttype (f2cl-lib:int-sub ttype 11)))
     (t
```

```
                    (setf tau (* qurtr tau))
                    (setf ttype (f2cl-lib:int-sub ttype 12))))
                  (go label80))
                 ((/= dmin dmin)
                  (setf tau zero)
                  (go label80))
                 (t
                  (go label90))))))
   label90
           (multiple-value-bind
                 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
               (dlasq6 i0 n0 z pp dmin f2cl-lib:dmin1 dmin2 dn dn1 dn2)
             (declare (ignore var-0 var-1 var-2 var-3 var-5))
             (setf dmin var-4)
             (setf dmin2 var-6)
             (setf dn var-7)
             (setf dn1 var-8)
             (setf dn2 var-9))
           (setf ndiv
                   (f2cl-lib:int-add ndiv
                                     (f2cl-lib:int-add (f2cl-lib:int-sub n0 i0)
                                                       2)))
           (setf iter (f2cl-lib:int-add iter 1))
           (setf tau zero)
   label100
           (cond
             ((< tau sigma)
              (setf desig (+ desig tau))
              (setf t$ (+ sigma desig))
              (setf desig (- desig (- t$ sigma))))
             (t
              (setf t$ (+ sigma tau))
              (setf desig (+ (- sigma (- t$ tau)) desig))))
           (setf sigma t$)
   end_label
           (return
            (values nil
                    n0
                    nil
                    nil
                    dmin
                    sigma
                    desig
                    qmax
                    nfail
                    iter
```

```
                    ndiv
                    nil)))))))
```

7.70 dlasq4 LAPACK

⟨*dlasq4.input*⟩≡

```
)set break resume
)sys rm -f dlasq4.output
)spool dlasq4.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasq4.help*⟩≡

```
======================================================================
dlasq4 examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DLASQ4  -  an approximation TAU to the smallest eigenvalue using values
       of d from the previous transform

SYNOPSIS
       SUBROUTINE DLASQ4( I0, N0, Z, PP, N0IN, DMIN, DMIN1,  DMIN2,  DN,  DN1,
                          DN2, TAU, TTYPE )

           INTEGER        I0, N0, N0IN, PP, TTYPE

           DOUBLE         PRECISION DMIN, DMIN1, DMIN2, DN, DN1, DN2, TAU

           DOUBLE         PRECISION Z( * )

PURPOSE
       DLASQ4  computes  an approximation TAU to the smallest eigenvalue using
       values of d from the previous transform.

       I0    (input) INTEGER
             First index.

       N0    (input) INTEGER
             Last index.

       Z     (input) DOUBLE PRECISION array, dimension ( 4*N )
             Z holds the qd array.

       PP    (input) INTEGER
             PP=0 for ping, PP=1 for pong.

       N0IN  (input) INTEGER
             The value of N0 at start of EIGTEST.

       DMIN  (input) DOUBLE PRECISION
             Minimum value of d.

       DMIN1 (input) DOUBLE PRECISION
```

```
      Minimum value of d, excluding D( N0 ).

DMIN2 (input) DOUBLE PRECISION
      Minimum value of d, excluding D( N0 ) and D( N0-1 ).

DN    (input) DOUBLE PRECISION
      d(N)

DN1   (input) DOUBLE PRECISION
      d(N-1)

DN2   (input) DOUBLE PRECISION
      d(N-2)

TAU   (output) DOUBLE PRECISION
      This is the shift.

TTYPE (output) INTEGER
      Shift type.
```

⟨*LAPACK dlasq4*⟩≡

```
(let* ((cnst1 0.563)
       (cnst2 1.01)
       (cnst3 1.05)
       (qurtr 0.25)
       (third$ 0.333)
       (half 0.5)
       (zero 0.0)
       (one 1.0)
       (two 2.0)
       (hundrd 100.0))
  (declare (type (double-float 0.563 0.563) cnst1)
           (type (double-float 1.01 1.01) cnst2)
           (type (double-float 1.05 1.05) cnst3)
           (type (double-float 0.25 0.25) qurtr)
           (type (double-float 0.333 0.333) third$)
           (type (double-float 0.5 0.5) half)
           (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two)
           (type (double-float 100.0 100.0) hundrd))
  (let ((g zero))
    (declare (type (double-float) g))
    (defun dlasq4
           (i0 n0 z pp n0in dmin f2cl-lib:dmin1 dmin2 dn dn1 dn2 tau ttype)
      (declare (type (double-float) tau dn2 dn1 dn dmin2 f2cl-lib:dmin1 dmin)
               (type (simple-array double-float (*)) z)
               (type fixnum ttype n0in pp n0 i0))
      (f2cl-lib:with-multi-array-data
          ((z double-float z-%data% z-%offset%))
        (prog ((a2 0.0) (b1 0.0) (b2 0.0) (gam 0.0) (gap1 0.0) (gap2 0.0)
               (s 0.0) (i4 0) (nn 0) (np 0) (sqrt$ 0.0f0))
          (declare (type (single-float) sqrt$)
                   (type (double-float) a2 b1 b2 gam gap1 gap2 s)
                   (type fixnum i4 nn np))
          (cond
            ((<= dmin zero)
             (setf tau (- dmin))
             (setf ttype -1)
             (go end_label)))
          (setf nn (f2cl-lib:int-add (f2cl-lib:int-mul 4 n0) pp))
          (cond
            ((= n0in n0)
             (cond
               ((or (= dmin dn) (= dmin dn1))
                (setf b1
```

```
                (*
                 (f2cl-lib:fsqrt
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub nn 3))
                                 ((1 *))
                                 z-%offset%))
                 (f2cl-lib:fsqrt
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub nn 5))
                                 ((1 *))
                                 z-%offset%))))
          (setf b2
                (*
                 (f2cl-lib:fsqrt
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub nn 7))
                                 ((1 *))
                                 z-%offset%))
                 (f2cl-lib:fsqrt
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub nn 9))
                                 ((1 *))
                                 z-%offset%))))
          (setf a2
                (+
                 (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-sub nn 7))
                                ((1 *))
                                z-%offset%)
                 (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-sub nn 5))
                                ((1 *))
                                z-%offset%)))
          (cond
            ((and (= dmin dn) (= f2cl-lib:dmin1 dn1))
             (setf gap2 (- dmin2 a2 (* dmin2 qurtr)))
             (cond
               ((and (> gap2 zero) (> gap2 b2))
                (setf gap1 (- a2 dn (* (/ b2 gap2) b2))))
               (t
                (setf gap1 (- a2 dn (+ b1 b2)))))
             (cond
               ((and (> gap1 zero) (> gap1 b1))
                (setf s (max (- dn (* (/ b1 gap1) b1)) (* half dmin)))
                (setf ttype -2))
               (t
```

```
    (setf s zero)
    (if (> dn b1) (setf s (- dn b1)))
    (if (> a2 (+ b1 b2)) (setf s (min s (- a2 (+ b1 b2)))))
    (setf s (max s (* third$ dmin)))
    (setf ttype -3))))
(t
 (tagbody
   (setf ttype -4)
   (setf s (* qurtr dmin))
   (cond
     ((= dmin dn)
      (setf gam dn)
      (setf a2 zero)
      (if
       (>
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-sub nn 5))
                       ((1 *))
                       z-%offset%)
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-sub nn 7))
                       ((1 *))
                       z-%offset%))
       (go end_label))
      (setf b2
              (/
               (f2cl-lib:fref z-%data%
                              ((f2cl-lib:int-sub nn 5))
                              ((1 *))
                              z-%offset%)
               (f2cl-lib:fref z-%data%
                              ((f2cl-lib:int-sub nn 7))
                              ((1 *))
                              z-%offset%)))
      (setf np (f2cl-lib:int-sub nn 9)))
     (t
      (setf np (f2cl-lib:int-sub nn (f2cl-lib:int-mul 2 pp)))
      (setf b2
              (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-sub np 2))
                             ((1 *))
                             z-%offset%))
      (setf gam dn1)
      (if
       (>
        (f2cl-lib:fref z-%data%
```

```
                                ((f2cl-lib:int-sub np 4))
                                ((1 *))
                                z-%offset%)
                  (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-sub np 2))
                                ((1 *))
                                z-%offset%))
                 (go end_label))
                (setf a2
                       (/
                        (f2cl-lib:fref z-%data%
                                       ((f2cl-lib:int-sub np 4))
                                       ((1 *))
                                       z-%offset%)
                        (f2cl-lib:fref z-%data%
                                       ((f2cl-lib:int-sub np 2))
                                       ((1 *))
                                       z-%offset%)))
                (if
                 (>
                  (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-sub nn 9))
                                ((1 *))
                                z-%offset%)
                  (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-sub nn 11))
                                ((1 *))
                                z-%offset%))
                 (go end_label))
                (setf b2
                       (/
                        (f2cl-lib:fref z-%data%
                                       ((f2cl-lib:int-sub nn 9))
                                       ((1 *))
                                       z-%offset%)
                        (f2cl-lib:fref z-%data%
                                       ((f2cl-lib:int-sub nn 11))
                                       ((1 *))
                                       z-%offset%)))
                (setf np (f2cl-lib:int-sub nn 13))))
           (setf a2 (+ a2 b2))
           (f2cl-lib:fdo (i4 np
                          (f2cl-lib:int-add i4 (f2cl-lib:int-sub 4)))
                         ((> i4
                             (f2cl-lib:int-add
                              (f2cl-lib:int-mul 4 i0)
```

```
                                    (f2cl-lib:int-sub 1)
                                    pp))
                                nil)
                          (tagbody
                            (if (= b2 zero) (go label20))
                            (setf b1 b2)
                            (if
                             (> (f2cl-lib:fref z-%data% (i4) ((1 *)) z-%offset%)
                                (f2cl-lib:fref z-%data%
                                               ((f2cl-lib:int-sub i4 2))
                                               ((1 *))
                                               z-%offset%))
                             (go end_label))
                            (setf b2
                                    (* b2
                                       (/
                                        (f2cl-lib:fref z-%data%
                                                       (i4)
                                                       ((1 *))
                                                       z-%offset%)
                                        (f2cl-lib:fref z-%data%
                                                       ((f2cl-lib:int-sub i4 2))
                                                       ((1 *))
                                                       z-%offset%))))
                            (setf a2 (+ a2 b2))
                            (if (or (< (* hundrd (max b2 b1)) a2) (< cnst1 a2))
                                (go label20))))
label20
                       (setf a2 (* cnst3 a2))
                       (if (< a2 cnst1)
                           (setf s
                                   (/ (* gam (- one (f2cl-lib:fsqrt a2)))
                                      (+ one a2))))))))
                 ((= dmin dn2)
                  (setf ttype -5)
                  (setf s (* qurtr dmin))
                  (setf np (f2cl-lib:int-sub nn (f2cl-lib:int-mul 2 pp)))
                  (setf b1
                          (f2cl-lib:fref z-%data%
                                         ((f2cl-lib:int-sub np 2))
                                         ((1 *))
                                         z-%offset%))
                  (setf b2
                          (f2cl-lib:fref z-%data%
                                         ((f2cl-lib:int-sub np 6))
                                         ((1 *))
```

```
                        z-%offset%))
(setf gam dn2)
(if
 (or
  (>
   (f2cl-lib:fref z-%data%
                  ((f2cl-lib:int-sub np 8))
                  ((1 *))
                  z-%offset%)
   b2)
  (>
   (f2cl-lib:fref z-%data%
                  ((f2cl-lib:int-sub np 4))
                  ((1 *))
                  z-%offset%)
   b1))
 (go end_label))
(setf a2
       (*
        (/
         (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub np 8))
                        ((1 *))
                        z-%offset%)
         b2)
        (+ one
           (/
            (f2cl-lib:fref z-%data%
                           ((f2cl-lib:int-sub np 4))
                           ((1 *))
                           z-%offset%)
            b1))))
(cond
  ((> (f2cl-lib:int-add n0 (f2cl-lib:int-sub i0)) 2)
   (tagbody
     (setf b2
             (/
              (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-sub nn 13))
                             ((1 *))
                             z-%offset%)
              (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-sub nn 15))
                             ((1 *))
                             z-%offset%)))
     (setf a2 (+ a2 b2))
```

```
                    (f2cl-lib:fdo (i4
                                   (f2cl-lib:int-add nn (f2cl-lib:int-sub 17))
                                   (f2cl-lib:int-add i4 (f2cl-lib:int-sub 4)))
                                  ((> i4
                                      (f2cl-lib:int-add
                                       (f2cl-lib:int-mul 4 i0)
                                       (f2cl-lib:int-sub 1)
                                       pp))
                                   nil)
                      (tagbody
                        (if (= b2 zero) (go label40))
                        (setf b1 b2)
                        (if
                         (> (f2cl-lib:fref z-%data% (i4) ((1 *)) z-%offset%)
                            (f2cl-lib:fref z-%data%
                                           ((f2cl-lib:int-sub i4 2))
                                           ((1 *))
                                           z-%offset%))
                         (go end_label))
                        (setf b2
                                (* b2
                                   (/
                                    (f2cl-lib:fref z-%data%
                                                   (i4)
                                                   ((1 *))
                                                   z-%offset%)
                                    (f2cl-lib:fref z-%data%
                                                   ((f2cl-lib:int-sub i4 2))
                                                   ((1 *))
                                                   z-%offset%))))
                        (setf a2 (+ a2 b2))
                        (if (or (< (* hundrd (max b2 b1)) a2) (< cnst1 a2))
                            (go label40))))
label40
                     (setf a2 (* cnst3 a2)))))
                (if (< a2 cnst1)
                    (setf s
                          (/ (* gam (- one (f2cl-lib:fsqrt a2)))
                             (+ one a2)))))
               (t
                (cond
                  ((= ttype (f2cl-lib:int-sub 6))
                   (setf g (+ g (* third$ (- one g)))))
                  ((= ttype (f2cl-lib:int-sub 18))
                   (setf g (* qurtr third$)))
                  (t
```

```
          (setf g qurtr)))
       (setf s (* g dmin))
       (setf ttype -6))))
   ((= n0in (f2cl-lib:int-add n0 1))
    (cond
      ((and (= f2cl-lib:dmin1 dn1) (= dmin2 dn2))
       (tagbody
         (setf ttype -7)
         (setf s (* third$ f2cl-lib:dmin1))
         (if
          (>
           (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-sub nn 5))
                          ((1 *))
                          z-%offset%)
           (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-sub nn 7))
                          ((1 *))
                          z-%offset%))
          (go end_label))
         (setf b1
                (/
                 (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-sub nn 5))
                                ((1 *))
                                z-%offset%)
                 (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-sub nn 7))
                                ((1 *))
                                z-%offset%)))
         (setf b2 b1)
         (if (= b2 zero) (go label60))
         (f2cl-lib:fdo (i4
                        (f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
                                          (f2cl-lib:int-sub 9)
                                          pp)
                        (f2cl-lib:int-add i4 (f2cl-lib:int-sub 4)))
                       ((> i4
                           (f2cl-lib:int-add (f2cl-lib:int-mul 4 i0)
                                             (f2cl-lib:int-sub 1)
                                             pp))
                        nil)
           (tagbody
             (setf a2 b1)
             (if
              (> (f2cl-lib:fref z-%data% (i4) ((1 *)) z-%offset%)
```

```
                        (f2cl-lib:fref z-%data%
                                       ((f2cl-lib:int-sub i4 2))
                                       ((1 *))
                                       z-%offset%))
                       (go end_label))
                      (setf b1
                            (* b1
                               (/
                                (f2cl-lib:fref z-%data%
                                               (i4)
                                               ((1 *))
                                               z-%offset%)
                                (f2cl-lib:fref z-%data%
                                               ((f2cl-lib:int-sub i4 2))
                                               ((1 *))
                                               z-%offset%))))
                      (setf b2 (+ b2 b1))
                      (if (< (* hundrd (max b1 a2)) b2) (go label60))))
label60
                 (setf b2 (f2cl-lib:fsqrt (* cnst3 b2)))
                 (setf a2 (/ f2cl-lib:dmin1 (+ one (expt b2 2))))
                 (setf gap2 (- (* half dmin2) a2))
                 (cond
                   ((and (> gap2 zero) (> gap2 (* b2 a2)))
                    (setf s
                            (max s
                                 (* a2
                                    (+ one (* (- cnst2) a2 (/ b2 gap2) b2))))))
                   (t
                    (setf s (max s (* a2 (- one (* cnst2 b2)))))
                    (setf ttype -8)))))
              (t
               (setf s (* qurtr f2cl-lib:dmin1))
               (if (= f2cl-lib:dmin1 dn1) (setf s (* half f2cl-lib:dmin1)))
               (setf ttype -9))))
         ((= n0in (f2cl-lib:int-add n0 2))
          (cond
            ((and (= dmin2 dn2)
                  (<
                   (* two
                      (f2cl-lib:fref z
                                     ((f2cl-lib:int-add nn
                                                        (f2cl-lib:int-sub
                                                         5)))
                                     ((1 *))))
                   (f2cl-lib:fref z
```

```
                        ((f2cl-lib:int-add nn
                                           (f2cl-lib:int-sub 7)))
                        ((1 *)))))
          (tagbody
            (setf ttype -10)
            (setf s (* third$ dmin2))
            (if
             (>
              (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-sub nn 5))
                             ((1 *))
                             z-%offset%)
              (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-sub nn 7))
                             ((1 *))
                             z-%offset%))
             (go end_label))
            (setf b1
                  (/
                   (f2cl-lib:fref z-%data%
                                  ((f2cl-lib:int-sub nn 5))
                                  ((1 *))
                                  z-%offset%)
                   (f2cl-lib:fref z-%data%
                                  ((f2cl-lib:int-sub nn 7))
                                  ((1 *))
                                  z-%offset%)))
            (setf b2 b1)
            (if (= b2 zero) (go label80))
            (f2cl-lib:fdo (i4
                           (f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
                                             (f2cl-lib:int-sub 9)
                                             pp)
                           (f2cl-lib:int-add i4 (f2cl-lib:int-sub 4)))
                          ((> i4
                              (f2cl-lib:int-add (f2cl-lib:int-mul 4 i0)
                                                (f2cl-lib:int-sub 1)
                                                pp))
                           nil)
              (tagbody
                (if
                 (> (f2cl-lib:fref z-%data% (i4) ((1 *)) z-%offset%)
                    (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-sub i4 2))
                                   ((1 *))
                                   z-%offset%))
```

```
                     (go end_label))
                    (setf b1
                            (* b1
                               (/
                                (f2cl-lib:fref z-%data%
                                               (i4)
                                               ((1 *))
                                               z-%offset%)
                                (f2cl-lib:fref z-%data%
                                               ((f2cl-lib:int-sub i4 2))
                                               ((1 *))
                                               z-%offset%))))
                    (setf b2 (+ b2 b1))
                    (if (< (* hundrd b1) b2) (go label80))))
  label80
                  (setf b2 (f2cl-lib:fsqrt (* cnst3 b2)))
                  (setf a2 (/ dmin2 (+ one (expt b2 2))))
                  (setf gap2
                        (-
                         (+
                          (f2cl-lib:fref z-%data%
                                         ((f2cl-lib:int-sub nn 7))
                                         ((1 *))
                                         z-%offset%)
                          (f2cl-lib:fref z-%data%
                                         ((f2cl-lib:int-sub nn 9))
                                         ((1 *))
                                         z-%offset%))
                         (*
                          (f2cl-lib:fsqrt
                           (f2cl-lib:fref z-%data%
                                          ((f2cl-lib:int-sub nn 11))
                                          ((1 *))
                                          z-%offset%))
                          (f2cl-lib:fsqrt
                           (f2cl-lib:fref z-%data%
                                          ((f2cl-lib:int-sub nn 9))
                                          ((1 *))
                                          z-%offset%)))
                         a2))
                  (cond
                    ((and (> gap2 zero) (> gap2 (* b2 a2)))
                     (setf s
                             (max s
                                  (* a2
                                     (+ one (* (- cnst2) a2 (/ b2 gap2) b2))))))
```

```
                    (t
                     (setf s (max s (* a2 (- one (* cnst2 b2)))))))))
                (t
                 (setf s (* qurtr dmin2))
                 (setf ttype -11))))
             ((> n0in (f2cl-lib:int-add n0 2))
              (setf s zero)
              (setf ttype -12)))
            (setf tau s)
  end_label
            (return
             (values nil nil nil nil nil nil nil nil nil nil nil tau ttype)))))))
```

7.71 dlasq5 LAPACK

⟨*dlasq5.input*⟩≡

```
)set break resume
)sys rm -f dlasq5.output
)spool dlasq5.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasq5.help*⟩≡

```
=====================================================================
dlasq5 examples
=====================================================================

=====================================================================
Man Page Details
=====================================================================

NAME
       DLASQ5  -  one  dqds  transform in ping-pong form, one version for IEEE
       machines another for non IEEE machines

SYNOPSIS
       SUBROUTINE DLASQ5( I0, N0, Z, PP, TAU, DMIN, DMIN1,  DMIN2,  DN,  DNM1,
                          DNM2, IEEE )

           LOGICAL         IEEE

           INTEGER         I0, N0, PP

           DOUBLE          PRECISION DMIN, DMIN1, DMIN2, DN, DNM1, DNM2, TAU

           DOUBLE          PRECISION Z( * )

PURPOSE
       DLASQ5  computes  one dqds transform in ping-pong form, one version for
       IEEE machines another for non IEEE machines.


ARGUMENTS
       I0     (input) INTEGER
              First index.

       N0     (input) INTEGER
              Last index.

       Z      (input) DOUBLE PRECISION array, dimension ( 4*N )
              Z holds the qd array. EMIN is stored in Z(4*N0) to avoid an extra
              argument.

       PP     (input) INTEGER
              PP=0 for ping, PP=1 for pong.

       TAU    (input) DOUBLE PRECISION
              This is the shift.
```

```
DMIN  (output) DOUBLE PRECISION
      Minimum value of d.

      DMIN1  (output) DOUBLE PRECISION Minimum value of d, excluding D(
      N0 ).

      DMIN2 (output) DOUBLE PRECISION Minimum value of d, excluding  D(
      N0 ) and D( N0-1 ).

DN    (output) DOUBLE PRECISION
      d(N0), the last value of d.

DNM1  (output) DOUBLE PRECISION
      d(N0-1).

DNM2  (output) DOUBLE PRECISION
      d(N0-2).

IEEE  (input) LOGICAL
      Flag for IEEE or non IEEE arithmetic.
```

⟨*LAPACK dlasq5*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dlasq5 (i0 n0 z pp tau dmin f2cl-lib:dmin1 dmin2 dn dnm1 dnm2 ieee)
    (declare (type (member t nil) ieee)
             (type (double-float) dnm2 dnm1 dn dmin2 f2cl-lib:dmin1 dmin tau)
             (type (simple-array double-float (*)) z)
             (type fixnum pp n0 i0))
    (f2cl-lib:with-multi-array-data
        ((z double-float z-%data% z-%offset%))
      (prog ((d 0.0) (emin 0.0) (temp 0.0) (j4 0) (j4p2 0))
        (declare (type (double-float) d emin temp)
                 (type fixnum j4 j4p2))
        (if (<= (f2cl-lib:int-sub n0 i0 1) 0) (go end_label))
        (setf j4
                (f2cl-lib:int-sub (f2cl-lib:int-add (f2cl-lib:int-mul 4 i0) pp)
                                  3))
        (setf emin
                (f2cl-lib:fref z-%data%
                               ((f2cl-lib:int-add j4 4))
                               ((1 *))
                               z-%offset%))
        (setf d (- (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%) tau))
        (setf dmin d)
        (setf f2cl-lib:dmin1
                (- (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)))
        (cond
          (ieee
           (cond
             ((= pp 0)
              (f2cl-lib:fdo (j4 (f2cl-lib:int-mul 4 i0) (f2cl-lib:int-add j4 4))
                            ((> j4
                                (f2cl-lib:int-mul 4
                                                  (f2cl-lib:int-add n0
                                                                    (f2cl-lib:int-sub
                                                                     3))))
                             nil)
                (tagbody
                  (setf (f2cl-lib:fref z-%data%
                                       ((f2cl-lib:int-sub j4 2))
                                       ((1 *))
                                       z-%offset%)
                        (+ d
                           (f2cl-lib:fref z-%data%
                                          ((f2cl-lib:int-sub j4 1))
                                          ((1 *))
```

```
                                    z-%offset%)))
          (setf temp
                (/
                 (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-add j4 1))
                                ((1 *))
                                z-%offset%)
                 (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-sub j4 2))
                                ((1 *))
                                z-%offset%)))
          (setf d (- (* d temp) tau))
          (setf dmin (min dmin d))
          (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
                (*
                 (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-sub j4 1))
                                ((1 *))
                                z-%offset%)
                 temp))
          (setf emin
                (min (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
                     emin)))))
        (t
         (f2cl-lib:fdo (j4 (f2cl-lib:int-mul 4 i0) (f2cl-lib:int-add j4 4))
                       ((> j4
                           (f2cl-lib:int-mul 4
                                             (f2cl-lib:int-add n0
                                                               (f2cl-lib:int-sub
                                                                3))))
                        nil)
           (tagbody
             (setf (f2cl-lib:fref z-%data%
                                  ((f2cl-lib:int-sub j4 3))
                                  ((1 *))
                                  z-%offset%)
                   (+ d
                      (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)))
             (setf temp
                   (/
                    (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-add j4 2))
                                   ((1 *))
                                   z-%offset%)
                    (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-sub j4 3))
```

```
                                ((1 *))
                                z-%offset%)))
              (setf d (- (* d temp) tau))
              (setf dmin (min dmin d))
              (setf (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-sub j4 1))
                                   ((1 *))
                                   z-%offset%)
                      (* (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
                         temp))
              (setf emin
                      (min
                       (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-sub j4 1))
                                   ((1 *))
                                   z-%offset%)
                       emin))))))
      (setf dnm2 d)
      (setf dmin2 dmin)
      (setf j4
              (f2cl-lib:int-sub
               (f2cl-lib:int-mul 4 (f2cl-lib:int-sub n0 2))
               pp))
      (setf j4p2
              (f2cl-lib:int-sub
               (f2cl-lib:int-add j4 (f2cl-lib:int-mul 2 pp))
               1))
      (setf (f2cl-lib:fref z-%data%
                           ((f2cl-lib:int-sub j4 2))
                           ((1 *))
                           z-%offset%)
              (+ dnm2 (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)))
      (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
              (*
               (f2cl-lib:fref z-%data%
                              ((f2cl-lib:int-add j4p2 2))
                              ((1 *))
                              z-%offset%)
               (/ (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub j4 2))
                                 ((1 *))
                                 z-%offset%))))
      (setf dnm1
              (-
               (*
```

```
                (f2cl-lib:fref z-%data%
                               ((f2cl-lib:int-add j4p2 2))
                               ((1 *))
                               z-%offset%)
                (/ dnm2
                   (f2cl-lib:fref z-%data%
                                  ((f2cl-lib:int-sub j4 2))
                                  ((1 *))
                                  z-%offset%)))
               tau))
        (setf dmin (min dmin dnm1))
        (setf f2cl-lib:dmin1 dmin)
        (setf j4 (f2cl-lib:int-add j4 4))
        (setf j4p2
              (f2cl-lib:int-sub
               (f2cl-lib:int-add j4 (f2cl-lib:int-mul 2 pp))
               1))
        (setf (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-sub j4 2))
                             ((1 *))
                             z-%offset%)
              (+ dnm1 (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)))
        (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
              (*
               (f2cl-lib:fref z-%data%
                              ((f2cl-lib:int-add j4p2 2))
                              ((1 *))
                              z-%offset%)
               (/ (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub j4 2))
                                 ((1 *))
                                 z-%offset%))))
        (setf dn
              (-
               (*
                (f2cl-lib:fref z-%data%
                               ((f2cl-lib:int-add j4p2 2))
                               ((1 *))
                               z-%offset%)
                (/ dnm1
                   (f2cl-lib:fref z-%data%
                                  ((f2cl-lib:int-sub j4 2))
                                  ((1 *))
                                  z-%offset%)))
               tau))
```

```
 (setf dmin (min dmin dn)))
(t
 (cond
   ((= pp 0)
    (f2cl-lib:fdo (j4 (f2cl-lib:int-mul 4 i0) (f2cl-lib:int-add j4 4))
                  ((> j4
                      (f2cl-lib:int-mul 4
                                        (f2cl-lib:int-add n0
                                                          (f2cl-lib:int-sub
                                                           3))))
                   nil)
      (tagbody
        (setf (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-sub j4 2))
                             ((1 *))
                             z-%offset%)
                (+ d
                   (f2cl-lib:fref z-%data%
                                  ((f2cl-lib:int-sub j4 1))
                                  ((1 *))
                                  z-%offset%)))
        (cond
          ((< d zero)
           (go end_label))
          (t
           (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
                   (*
                    (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-add j4 1))
                                   ((1 *))
                                   z-%offset%)
                    (/
                     (f2cl-lib:fref z-%data%
                                    ((f2cl-lib:int-sub j4 1))
                                    ((1 *))
                                    z-%offset%)
                     (f2cl-lib:fref z-%data%
                                    ((f2cl-lib:int-sub j4 2))
                                    ((1 *))
                                    z-%offset%))))
           (setf d
                   (-
                    (*
                     (f2cl-lib:fref z-%data%
                                    ((f2cl-lib:int-add j4 1))
                                    ((1 *))
```

```
                                        z-%offset%)
                          (/ d
                             (f2cl-lib:fref z-%data%
                                            ((f2cl-lib:int-sub j4 2))
                                            ((1 *))
                                            z-%offset%)))
                          tau))))
          (setf dmin (min dmin d))
          (setf emin
                  (min emin
                       (f2cl-lib:fref z-%data%
                                      (j4)
                                      ((1 *))
                                      z-%offset%))))))
     (t
      (f2cl-lib:fdo (j4 (f2cl-lib:int-mul 4 i0) (f2cl-lib:int-add j4 4))
                    ((> j4
                        (f2cl-lib:int-mul 4
                                          (f2cl-lib:int-add n0
                                                            (f2cl-lib:int-sub
                                                             3))))
                     nil)
        (tagbody
          (setf (f2cl-lib:fref z-%data%
                               ((f2cl-lib:int-sub j4 3))
                               ((1 *))
                               z-%offset%)
                  (+ d
                     (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)))
          (cond
            ((< d zero)
             (go end_label))
            (t
             (setf (f2cl-lib:fref z-%data%
                                  ((f2cl-lib:int-sub j4 1))
                                  ((1 *))
                                  z-%offset%)
                     (*
                      (f2cl-lib:fref z-%data%
                                     ((f2cl-lib:int-add j4 2))
                                     ((1 *))
                                     z-%offset%)
                      (/
                       (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
                       (f2cl-lib:fref z-%data%
                                      ((f2cl-lib:int-sub j4 3))
```

```
                                        ((1 *))
                                        z-%offset%))))
                (setf d
                        (-
                         (*
                          (f2cl-lib:fref z-%data%
                                        ((f2cl-lib:int-add j4 2))
                                        ((1 *))
                                        z-%offset%)
                          (/ d
                             (f2cl-lib:fref z-%data%
                                           ((f2cl-lib:int-sub j4 3))
                                           ((1 *))
                                           z-%offset%)))
                         tau))))
               (setf dmin (min dmin d))
               (setf emin
                       (min emin
                            (f2cl-lib:fref z-%data%
                                        ((f2cl-lib:int-sub j4 1))
                                        ((1 *))
                                        z-%offset%)))))))
        (setf dnm2 d)
        (setf dmin2 dmin)
        (setf j4
                (f2cl-lib:int-sub
                 (f2cl-lib:int-mul 4 (f2cl-lib:int-sub n0 2))
                 pp))
        (setf j4p2
                (f2cl-lib:int-sub
                 (f2cl-lib:int-add j4 (f2cl-lib:int-mul 2 pp))
                 1))
        (setf (f2cl-lib:fref z-%data%
                             ((f2cl-lib:int-sub j4 2))
                             ((1 *))
                             z-%offset%)
                (+ dnm2 (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)))
        (cond
          ((< dnm2 zero)
           (go end_label))
          (t
           (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
                   (*
                    (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-add j4p2 2))
                                   ((1 *))
```

```
                                z-%offset%)
                        (/ (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)
                           (f2cl-lib:fref z-%data%
                                          ((f2cl-lib:int-sub j4 2))
                                          ((1 *))
                                          z-%offset%))))
           (setf dnm1
                 (-
                  (*
                   (f2cl-lib:fref z-%data%
                                  ((f2cl-lib:int-add j4p2 2))
                                  ((1 *))
                                  z-%offset%)
                   (/ dnm2
                      (f2cl-lib:fref z-%data%
                                     ((f2cl-lib:int-sub j4 2))
                                     ((1 *))
                                     z-%offset%)))
                  tau))))
       (setf dmin (min dmin dnm1))
       (setf f2cl-lib:dmin1 dmin)
       (setf j4 (f2cl-lib:int-add j4 4))
       (setf j4p2
             (f2cl-lib:int-sub
              (f2cl-lib:int-add j4 (f2cl-lib:int-mul 2 pp))
              1))
       (setf (f2cl-lib:fref z-%data%
                            ((f2cl-lib:int-sub j4 2))
                            ((1 *))
                            z-%offset%)
             (+ dnm1 (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)))
       (cond
         ((< dnm1 zero)
          (go end_label))
         (t
          (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
                (*
                 (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-add j4p2 2))
                                ((1 *))
                                z-%offset%)
                 (/ (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)
                    (f2cl-lib:fref z-%data%
                                   ((f2cl-lib:int-sub j4 2))
                                   ((1 *))
                                   z-%offset%))))
```

```
              (setf dn
                    (-
                     (*
                      (f2cl-lib:fref z-%data%
                                     ((f2cl-lib:int-add j4p2 2))
                                     ((1 *))
                                     z-%offset%)
                      (/ dnm1
                         (f2cl-lib:fref z-%data%
                                        ((f2cl-lib:int-sub j4 2))
                                        ((1 *))
                                        z-%offset%)))
                      tau))))
            (setf dmin (min dmin dn))))
       (setf (f2cl-lib:fref z-%data%
                            ((f2cl-lib:int-add j4 2))
                            ((1 *))
                            z-%offset%)
             dn)
       (setf (f2cl-lib:fref z-%data%
                            ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0) pp))
                            ((1 *))
                            z-%offset%)
             emin)
end_label
       (return
        (values nil
                nil
                nil
                nil
                nil
                dmin
                f2cl-lib:dmin1
                dmin2
                dn
                dnm1
                dnm2
                nil))))))
```

7.72 dlasq6 LAPACK

⟨*dlasq6.input*⟩≡

```
)set break resume
)sys rm -f dlasq6.output
)spool dlasq6.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasq6.help*⟩≡

```
======================================================================
dlasq6 examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DLASQ6  -  one  dqd  (shift equal to zero) transform in ping-pong form,
       with protection against underflow and overflow

SYNOPSIS
       SUBROUTINE DLASQ6( I0, N0, Z, PP, DMIN, DMIN1, DMIN2, DN, DNM1, DNM2 )

           INTEGER        I0, N0, PP

           DOUBLE         PRECISION DMIN, DMIN1, DMIN2, DN, DNM1, DNM2

           DOUBLE         PRECISION Z( * )

PURPOSE
       DLASQ6 computes one dqd (shift equal to zero)  transform  in  ping-pong
       form, with protection against underflow and overflow.


ARGUMENTS
       I0    (input) INTEGER
             First index.

       N0    (input) INTEGER
             Last index.

       Z     (input) DOUBLE PRECISION array, dimension ( 4*N )
             Z holds the qd array. EMIN is stored in Z(4*N0) to avoid an extra
             argument.

       PP    (input) INTEGER
             PP=0 for ping, PP=1 for pong.

       DMIN  (output) DOUBLE PRECISION
             Minimum value of d.

             DMIN1 (output) DOUBLE PRECISION Minimum value of d, excluding  D(
             N0 ).
```

```
      DMIN2  (output) DOUBLE PRECISION Minimum value of d, excluding D(
      N0 ) and D( N0-1 ).

DN    (output) DOUBLE PRECISION
      d(N0), the last value of d.

DNM1  (output) DOUBLE PRECISION
      d(N0-1).

DNM2  (output) DOUBLE PRECISION
      d(N0-2).
```

⟨*LAPACK dlasq6*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dlasq6 (i0 n0 z pp dmin f2cl-lib:dmin1 dmin2 dn dnm1 dnm2)
    (declare (type (double-float) dnm2 dnm1 dn dmin2 f2cl-lib:dmin1 dmin)
             (type (simple-array double-float (*)) z)
             (type fixnum pp n0 i0))
    (f2cl-lib:with-multi-array-data
        ((z double-float z-%data% z-%offset%))
      (prog ((d 0.0) (emin 0.0) (safmin 0.0) (temp 0.0) (j4 0) (j4p2 0))
        (declare (type (double-float) d emin safmin temp)
                 (type fixnum j4 j4p2))
        (if (<= (f2cl-lib:int-sub n0 i0 1) 0) (go end_label))
        (setf safmin (dlamch "Safe minimum"))
        (setf j4
                (f2cl-lib:int-sub (f2cl-lib:int-add (f2cl-lib:int-mul 4 i0) pp)
                                  3))
        (setf emin
                (f2cl-lib:fref z-%data%
                               ((f2cl-lib:int-add j4 4))
                               ((1 *))
                               z-%offset%))
        (setf d (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%))
        (setf dmin d)
        (cond
          ((= pp 0)
           (f2cl-lib:fdo (j4 (f2cl-lib:int-mul 4 i0) (f2cl-lib:int-add j4 4))
                         ((> j4
                             (f2cl-lib:int-mul 4
                                               (f2cl-lib:int-add n0
                                                                 (f2cl-lib:int-sub
                                                                  3))))
                          nil)
             (tagbody
               (setf (f2cl-lib:fref z-%data%
                                    ((f2cl-lib:int-sub j4 2))
                                    ((1 *))
                                    z-%offset%)
                     (+ d
                        (f2cl-lib:fref z-%data%
                                       ((f2cl-lib:int-sub j4 1))
                                       ((1 *))
                                       z-%offset%)))
               (cond
                 ((=
                   (f2cl-lib:fref z
```

```
                ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
                ((1 *)))
 zero)
 (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%) zero)
 (setf d
       (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-add j4 1))
                      ((1 *))
                      z-%offset%))
 (setf dmin d)
 (setf emin zero))
((and
  (<
   (* safmin
      (f2cl-lib:fref z ((f2cl-lib:int-add j4 1)) ((1 *))))
   (f2cl-lib:fref z
                  ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
                  ((1 *))))
  (<
   (* safmin
      (f2cl-lib:fref z
                     ((f2cl-lib:int-add j4
                                        (f2cl-lib:int-sub 2)))
                     ((1 *))))
   (f2cl-lib:fref z ((f2cl-lib:int-add j4 1)) ((1 *)))))
 (setf temp
       (/
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-add j4 1))
                       ((1 *))
                       z-%offset%)
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-sub j4 2))
                       ((1 *))
                       z-%offset%)))
 (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
       (*
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-sub j4 1))
                       ((1 *))
                       z-%offset%)
        temp))
 (setf d (* d temp)))
(t
 (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
       (*
```

```
                        (f2cl-lib:fref z-%data%
                                       ((f2cl-lib:int-add j4 1))
                                       ((1 *))
                                       z-%offset%)
                        (/
                         (f2cl-lib:fref z-%data%
                                        ((f2cl-lib:int-sub j4 1))
                                        ((1 *))
                                        z-%offset%)
                         (f2cl-lib:fref z-%data%
                                        ((f2cl-lib:int-sub j4 2))
                                        ((1 *))
                                        z-%offset%))))
                (setf d
                      (*
                       (f2cl-lib:fref z-%data%
                                      ((f2cl-lib:int-add j4 1))
                                      ((1 *))
                                      z-%offset%)
                       (/ d
                          (f2cl-lib:fref z-%data%
                                         ((f2cl-lib:int-sub j4 2))
                                         ((1 *))
                                         z-%offset%))))))
            (setf dmin (min dmin d))
            (setf emin
                  (min emin
                       (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%))))))
        (t
         (f2cl-lib:fdo (j4 (f2cl-lib:int-mul 4 i0) (f2cl-lib:int-add j4 4))
                       ((> j4
                           (f2cl-lib:int-mul 4
                                             (f2cl-lib:int-add n0
                                                               (f2cl-lib:int-sub
                                                                3))))
                        nil)
           (tagbody
             (setf (f2cl-lib:fref z-%data%
                                  ((f2cl-lib:int-sub j4 3))
                                  ((1 *))
                                  z-%offset%)
                   (+ d (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)))
             (cond
               ((=
                 (f2cl-lib:fref z
                                ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 3)))
```

```
                ((1 *)))
 zero)
 (setf (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-sub j4 1))
                      ((1 *))
                      z-%offset%)
       zero)
 (setf d
       (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-add j4 2))
                      ((1 *))
                      z-%offset%))
 (setf dmin d)
 (setf emin zero))
((and
  (<
   (* safmin
      (f2cl-lib:fref z ((f2cl-lib:int-add j4 2)) ((1 *))))
   (f2cl-lib:fref z
                  ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 3)))
                  ((1 *))))
  (<
   (* safmin
      (f2cl-lib:fref z
                     ((f2cl-lib:int-add j4
                                        (f2cl-lib:int-sub 3)))
                     ((1 *))))
   (f2cl-lib:fref z ((f2cl-lib:int-add j4 2)) ((1 *)))))
 (setf temp
       (/
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-add j4 2))
                       ((1 *))
                       z-%offset%)
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-sub j4 3))
                       ((1 *))
                       z-%offset%)))
 (setf (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-sub j4 1))
                      ((1 *))
                      z-%offset%)
       (* (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
          temp))
 (setf d (* d temp)))
(t
```

```
                (setf (f2cl-lib:fref z-%data%
                                     ((f2cl-lib:int-sub j4 1))
                                     ((1 *))
                                     z-%offset%)
                      (*
                       (f2cl-lib:fref z-%data%
                                      ((f2cl-lib:int-add j4 2))
                                      ((1 *))
                                      z-%offset%)
                       (/ (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
                          (f2cl-lib:fref z-%data%
                                        ((f2cl-lib:int-sub j4 3))
                                        ((1 *))
                                        z-%offset%))))
                (setf d
                      (*
                       (f2cl-lib:fref z-%data%
                                      ((f2cl-lib:int-add j4 2))
                                      ((1 *))
                                      z-%offset%)
                       (/ d
                          (f2cl-lib:fref z-%data%
                                        ((f2cl-lib:int-sub j4 3))
                                        ((1 *))
                                        z-%offset%))))))
          (setf dmin (min dmin d))
          (setf emin
                (min emin
                     (f2cl-lib:fref z-%data%
                                    ((f2cl-lib:int-sub j4 1))
                                    ((1 *))
                                    z-%offset%)))))))
(setf dnm2 d)
(setf dmin2 dmin)
(setf j4
      (f2cl-lib:int-sub (f2cl-lib:int-mul 4 (f2cl-lib:int-sub n0 2))
                        pp))
(setf j4p2
      (f2cl-lib:int-sub (f2cl-lib:int-add j4 (f2cl-lib:int-mul 2 pp))
                        1))
(setf (f2cl-lib:fref z-%data%
                     ((f2cl-lib:int-sub j4 2))
                     ((1 *))
                     z-%offset%)
      (+ dnm2 (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)))
(cond
```

```
((=
  (f2cl-lib:fref z
                 ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
                 ((1 *)))
  zero)
 (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%) zero)
 (setf dnm1
         (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-add j4p2 2))
                        ((1 *))
                        z-%offset%))
 (setf dmin dnm1)
 (setf emin zero))
((and
  (< (* safmin (f2cl-lib:fref z ((f2cl-lib:int-add j4p2 2)) ((1 *))))
     (f2cl-lib:fref z
                    ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
                    ((1 *))))
  (<
   (* safmin
      (f2cl-lib:fref z
                     ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
                     ((1 *))))
   (f2cl-lib:fref z ((f2cl-lib:int-add j4p2 2)) ((1 *)))))
 (setf temp
         (/
          (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-add j4p2 2))
                         ((1 *))
                         z-%offset%)
          (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub j4 2))
                         ((1 *))
                         z-%offset%)))
 (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
         (* (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%) temp))
 (setf dnm1 (* dnm2 temp)))
(t
 (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
         (*
          (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-add j4p2 2))
                         ((1 *))
                         z-%offset%)
          (/ (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)
             (f2cl-lib:fref z-%data%
```

```
                                ((f2cl-lib:int-sub j4 2))
                                ((1 *))
                                z-%offset%))))
   (setf dnm1
           (*
            (f2cl-lib:fref z-%data%
                           ((f2cl-lib:int-add j4p2 2))
                           ((1 *))
                           z-%offset%)
            (/ dnm2
               (f2cl-lib:fref z-%data%
                              ((f2cl-lib:int-sub j4 2))
                              ((1 *))
                              z-%offset%))))))
(setf dmin (min dmin dnm1))
(setf f2cl-lib:dmin1 dmin)
(setf j4 (f2cl-lib:int-add j4 4))
(setf j4p2
        (f2cl-lib:int-sub (f2cl-lib:int-add j4 (f2cl-lib:int-mul 2 pp))
                          1))
(setf (f2cl-lib:fref z-%data%
                     ((f2cl-lib:int-sub j4 2))
                     ((1 *))
                     z-%offset%)
        (+ dnm1 (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)))
(cond
  ((=
    (f2cl-lib:fref z
                   ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
                   ((1 *)))
    zero)
   (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%) zero)
   (setf dn
           (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-add j4p2 2))
                          ((1 *))
                          z-%offset%))
   (setf dmin dn)
   (setf emin zero))
  ((and
    (< (* safmin (f2cl-lib:fref z ((f2cl-lib:int-add j4p2 2)) ((1 *))))
       (f2cl-lib:fref z
                      ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
                      ((1 *))))
    (<
     (* safmin
```

```
                (f2cl-lib:fref z
                               ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
                               ((1 *))))
          (f2cl-lib:fref z ((f2cl-lib:int-add j4p2 2)) ((1 *)))))
         (setf temp
                 (/
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-add j4p2 2))
                                 ((1 *))
                                 z-%offset%)
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-sub j4 2))
                                 ((1 *))
                                 z-%offset%)))
         (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
                 (* (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%) temp))
         (setf dn (* dnm1 temp)))
        (t
         (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
                 (*
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-add j4p2 2))
                                 ((1 *))
                                 z-%offset%)
                  (/ (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)
                     (f2cl-lib:fref z-%data%
                                    ((f2cl-lib:int-sub j4 2))
                                    ((1 *))
                                    z-%offset%))))
         (setf dn
                 (*
                  (f2cl-lib:fref z-%data%
                                 ((f2cl-lib:int-add j4p2 2))
                                 ((1 *))
                                 z-%offset%)
                  (/ dnm1
                     (f2cl-lib:fref z-%data%
                                    ((f2cl-lib:int-sub j4 2))
                                    ((1 *))
                                    z-%offset%))))))
  (setf dmin (min dmin dn))
  (setf (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-add j4 2))
                       ((1 *))
                       z-%offset%)
        dn)
```

```
          (setf (f2cl-lib:fref z-%data%
                               ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0) pp))
                               ((1 *))
                               z-%offset%)
                emin)
   end_label
          (return
           (values nil nil nil nil dmin f2cl-lib:dmin1 dmin2 dn dnm1 dnm2))))))
```

7.73 dlasr LAPACK

⟨*dlasr.input*⟩≡

```
  )set break resume
  )sys rm -f dlasr.output
  )spool dlasr.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dlasr.help*⟩≡

```
  ==========================================================================
  dlasr examples
  ==========================================================================

  ==========================================================================
  Man Page Details
  ==========================================================================

  NAME
         DLASR - a sequence of plane rotations to a real matrix A,

  SYNOPSIS
         SUBROUTINE DLASR( SIDE, PIVOT, DIRECT, M, N, C, S, A, LDA )

             CHARACTER     DIRECT, PIVOT, SIDE

             INTEGER       LDA, M, N

             DOUBLE        PRECISION A( LDA, * ), C( * ), S( * )

  PURPOSE
         DLASR  applies  a  sequence of plane rotations to a real matrix A, from
         either the left or the right.

         When SIDE = 'L', the transformation takes the form

            A := P*A

         and when SIDE = 'R', the transformation takes the form

            A := A*P**T

         where P is an orthogonal matrix consisting of a  sequence  of  z  plane
         rotations,  with  z  = M when SIDE = 'L' and z = N when SIDE = 'R', and
         P**T is the transpose of P.

         When DIRECT = 'F' (Forward sequence), then

            P = P(z-1) * ... * P(2) * P(1)

         and when DIRECT = 'B' (Backward sequence), then

            P = P(1) * P(2) * ... * P(z-1)

         where P(k) is a plane rotation matrix defined by the 2-by-2 rotation
```

```
   R(k) = (  c(k)  s(k) )
        = ( -s(k)  c(k) ).
```

When PIVOT = 'V' (Variable pivot), the rotation is performed for the plane (k,k+1), i.e., P(k) has the form

```
   P(k) = (  1                                        )
          (       ...                                 )
          (              1                            )
          (                   c(k)  s(k)              )
          (                  -s(k)  c(k)              )
          (                                1          )
          (                                     ...   )
          (                                          1 )
```

where R(k) appears as a rank-2 modification to the identity matrix in rows and columns k and k+1.

When PIVOT = 'T' (Top pivot), the rotation is performed for the plane (1,k+1), so P(k) has the form

```
   P(k) = (  c(k)                   s(k)              )
          (         1                                 )
          (              ...                          )
          (                    1                      )
          ( -s(k)                  c(k)               )
          (                                1          )
          (                                     ...   )
          (                                          1 )
```

where R(k) appears in rows and columns 1 and k+1.

Similarly, when PIVOT = 'B' (Bottom pivot), the rotation is performed for the plane (k,z), giving P(k) the form

```
   P(k) = ( 1                                         )
          (      ...                                  )
          (             1                             )
          (                  c(k)                s(k) )
          (                        1                  )
          (                             ...           )
          (                                  1        )
          (                 -s(k)                c(k) )
```

where R(k) appears in rows and columns k and z. The rotations are per-

```
        formed without ever forming P(k) explicitly.


  ARGUMENTS
         SIDE    (input) CHARACTER*1
                 Specifies  whether  the plane rotation matrix P is applied to A
                 on the left or the right.  = 'L':  Left, compute A := P*A
                 = 'R':  Right, compute A:= A*P**T

         PIVOT   (input) CHARACTER*1
                 Specifies the plane for which P(k) is a plane rotation  matrix.
                 = 'V':  Variable pivot, the plane (k,k+1)
                 = 'T':  Top pivot, the plane (1,k+1)
                 = 'B':  Bottom pivot, the plane (k,z)

         DIRECT  (input) CHARACTER*1
                 Specifies  whether P is a forward or backward sequence of plane
                 rotations.  = 'F':  Forward, P = P(z-1)*...*P(2)*P(1)
                 = 'B':  Backward, P = P(1)*P(2)*...*P(z-1)

         M       (input) INTEGER
                 The number of rows of the matrix A.  If m <=  1,  an  immediate
                 return is effected.

         N       (input) INTEGER
                 The number of columns of the matrix A.  If n <= 1, an immediate
                 return is effected.

         C       (input) DOUBLE PRECISION array, dimension
                 (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' The cosines c(k) of the
                 plane rotations.

         S       (input) DOUBLE PRECISION array, dimension
                 (M-1)  if  SIDE = 'L' (N-1) if SIDE = 'R' The sines s(k) of the
                 plane rotations.  The 2-by-2 plane rotation part of the  matrix
                 P(k),  R(k),  has the form R(k) = (  c(k)  s(k) ) ( -s(k)  c(k)
                 ).

         A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
                 The M-by-N matrix A.  On exit, A is overwritten by P*A if  SIDE
                 = 'R' or by A*P**T if SIDE = 'L'.

         LDA     (input) INTEGER
                 The leading dimension of the array A.  LDA >= max(1,M).
```

⟨*LAPACK dlasr*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dlasr (side pivot direct m n c s a lda)
    (declare (type (simple-array double-float (*)) a s c)
             (type fixnum lda n m)
             (type character direct pivot side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (pivot character pivot-%data% pivot-%offset%)
         (direct character direct-%data% direct-%offset%)
         (c double-float c-%data% c-%offset%)
         (s double-float s-%data% s-%offset%)
         (a double-float a-%data% a-%offset%))
      (prog ((ctemp 0.0) (stemp 0.0) (temp 0.0) (i 0) (info 0) (j 0))
        (declare (type (double-float) ctemp stemp temp)
                 (type fixnum i info j))
        (setf info 0)
        (cond
          ((not (or (char-equal side #\L) (char-equal side #\R)))
           (setf info 1))
          ((not (or (char-equal pivot #\V) (char-equal pivot #\T) (char-equal pivot #
           (setf info 2))
          ((not (or (char-equal direct #\F) (char-equal direct #\B)))
           (setf info 3))
          ((< m 0)
           (setf info 4))
          ((< n 0)
           (setf info 5))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info 9)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DLASR" info)
           (go end_label)))
        (if (or (= m 0) (= n 0)) (go end_label))
        (cond
          ((char-equal side #\L)
           (cond
             ((char-equal pivot #\V)
              (cond
                ((char-equal direct #\F)
                 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
```

```
                    ((> j (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
                     nil)
                (tagbody
                  (setf ctemp
                          (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
                  (setf stemp
                          (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
                  (cond
                    ((or (/= ctemp one) (/= stemp zero))
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i n) nil)
                       (tagbody
                         (setf temp
                                 (f2cl-lib:fref a-%data%
                                                ((f2cl-lib:int-add j 1) i)
                                                ((1 lda) (1 *))
                                                a-%offset%))
                         (setf (f2cl-lib:fref a-%data%
                                              ((f2cl-lib:int-add j 1) i)
                                              ((1 lda) (1 *))
                                              a-%offset%)
                                 (- (* ctemp temp)
                                    (* stemp
                                       (f2cl-lib:fref a-%data%
                                                      (j i)
                                                      ((1 lda) (1 *))
                                                      a-%offset%))))
                         (setf (f2cl-lib:fref a-%data%
                                              (j i)
                                              ((1 lda) (1 *))
                                              a-%offset%)
                                 (+ (* stemp temp)
                                    (* ctemp
                                       (f2cl-lib:fref a-%data%
                                                      (j i)
                                                      ((1 lda) (1 *))
                                                      a-%offset%)))))))))))
            ((char-equal direct #\B)
             (f2cl-lib:fdo (j (f2cl-lib:int-add m (f2cl-lib:int-sub 1))
                              (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                           ((> j 1) nil)
               (tagbody
                 (setf ctemp
                         (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
                 (setf stemp
                         (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
```

```
                (cond
                  ((or (/= ctemp one) (/= stemp zero))
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf temp
                             (f2cl-lib:fref a-%data%
                                            ((f2cl-lib:int-add j 1) i)
                                            ((1 lda) (1 *))
                                            a-%offset%))
                       (setf (f2cl-lib:fref a-%data%
                                            ((f2cl-lib:int-add j 1) i)
                                            ((1 lda) (1 *))
                                            a-%offset%)
                             (- (* ctemp temp)
                                (* stemp
                                   (f2cl-lib:fref a-%data%
                                                  (j i)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))))
                       (setf (f2cl-lib:fref a-%data%
                                            (j i)
                                            ((1 lda) (1 *))
                                            a-%offset%)
                             (+ (* stemp temp)
                                (* ctemp
                                   (f2cl-lib:fref a-%data%
                                                  (j i)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))))))))))
           ((char-equal pivot #\T)
            (cond
              ((char-equal direct #\F)
               (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                             ((> j m) nil)
                 (tagbody
                   (setf ctemp
                         (f2cl-lib:fref c-%data%
                                        ((f2cl-lib:int-sub j 1))
                                        ((1 *))
                                        c-%offset%))
                   (setf stemp
                         (f2cl-lib:fref s-%data%
                                        ((f2cl-lib:int-sub j 1))
                                        ((1 *))
                                        s-%offset%))
```

```
                (cond
                  ((or (/= ctemp one) (/= stemp zero))
                   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                 ((> i n) nil)
                     (tagbody
                       (setf temp
                             (f2cl-lib:fref a-%data%
                                            (j i)
                                            ((1 lda) (1 *))
                                            a-%offset%))
                       (setf (f2cl-lib:fref a-%data%
                                            (j i)
                                            ((1 lda) (1 *))
                                            a-%offset%)
                             (- (* ctemp temp)
                                (* stemp
                                   (f2cl-lib:fref a-%data%
                                                  (1 i)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))))
                       (setf (f2cl-lib:fref a-%data%
                                            (1 i)
                                            ((1 lda) (1 *))
                                            a-%offset%)
                             (+ (* stemp temp)
                                (* ctemp
                                   (f2cl-lib:fref a-%data%
                                                  (1 i)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))))))))
               ((char-equal direct #\B)
                (f2cl-lib:fdo (j m (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                              ((> j 2) nil)
                  (tagbody
                    (setf ctemp
                          (f2cl-lib:fref c-%data%
                                         ((f2cl-lib:int-sub j 1))
                                         ((1 *))
                                         c-%offset%))
                    (setf stemp
                          (f2cl-lib:fref s-%data%
                                         ((f2cl-lib:int-sub j 1))
                                         ((1 *))
                                         s-%offset%))
                    (cond
                      ((or (/= ctemp one) (/= stemp zero))
```

```
                    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                  ((> i n) nil)
                      (tagbody
                        (setf temp
                                (f2cl-lib:fref a-%data%
                                               (j i)
                                               ((1 lda) (1 *))
                                               a-%offset%))
                        (setf (f2cl-lib:fref a-%data%
                                             (j i)
                                             ((1 lda) (1 *))
                                             a-%offset%)
                                (- (* ctemp temp)
                                   (* stemp
                                      (f2cl-lib:fref a-%data%
                                                     (1 i)
                                                     ((1 lda) (1 *))
                                                     a-%offset%))))
                        (setf (f2cl-lib:fref a-%data%
                                             (1 i)
                                             ((1 lda) (1 *))
                                             a-%offset%)
                                (+ (* stemp temp)
                                   (* ctemp
                                      (f2cl-lib:fref a-%data%
                                                     (1 i)
                                                     ((1 lda) (1 *))
                                                     a-%offset%)))))))))))
          ((char-equal pivot #\B)
           (cond
             ((char-equal direct #\F)
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
                             nil)
                (tagbody
                  (setf ctemp
                          (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
                  (setf stemp
                          (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
                  (cond
                    ((or (/= ctemp one) (/= stemp zero))
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i n) nil)
                       (tagbody
                         (setf temp
                                 (f2cl-lib:fref a-%data%
```

```
                                        (j i)
                                        ((1 lda) (1 *))
                                        a-%offset%))
                        (setf (f2cl-lib:fref a-%data%
                                             (j i)
                                             ((1 lda) (1 *))
                                             a-%offset%)
                              (+
                               (* stemp
                                  (f2cl-lib:fref a-%data%
                                                 (m i)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))
                               (* ctemp temp)))
                        (setf (f2cl-lib:fref a-%data%
                                             (m i)
                                             ((1 lda) (1 *))
                                             a-%offset%)
                              (-
                               (* ctemp
                                  (f2cl-lib:fref a-%data%
                                                 (m i)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))
                               (* stemp temp))))))))))
      ((char-equal direct #\B)
       (f2cl-lib:fdo (j (f2cl-lib:int-add m (f2cl-lib:int-sub 1))
                      (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                     ((> j 1) nil)
         (tagbody
           (setf ctemp
                 (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
           (setf stemp
                 (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
           (cond
             ((or (/= ctemp one) (/= stemp zero))
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                            ((> i n) nil)
                (tagbody
                  (setf temp
                        (f2cl-lib:fref a-%data%
                                       (j i)
                                       ((1 lda) (1 *))
                                       a-%offset%))
                  (setf (f2cl-lib:fref a-%data%
                                       (j i)
```

```
                                                ((1 lda) (1 *))
                                                a-%offset%)
                                        (+
                                         (* stemp
                                            (f2cl-lib:fref a-%data%
                                                           (m i)
                                                           ((1 lda) (1 *))
                                                           a-%offset%))
                                         (* ctemp temp)))
                                (setf (f2cl-lib:fref a-%data%
                                                     (m i)
                                                     ((1 lda) (1 *))
                                                     a-%offset%)
                                        (-
                                         (* ctemp
                                            (f2cl-lib:fref a-%data%
                                                           (m i)
                                                           ((1 lda) (1 *))
                                                           a-%offset%))
                                         (* stemp temp))))))))))))))
                ((char-equal side #\R)
                 (cond
                   ((char-equal pivot #\V)
                    (cond
                      ((char-equal direct #\F)
                       (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                                     ((> j (f2cl-lib:int-add n (f2cl-lib:int-sub 1)))
                                      nil)
                         (tagbody
                           (setf ctemp
                                   (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
                           (setf stemp
                                   (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
                           (cond
                             ((or (/= ctemp one) (/= stemp zero))
                              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                            ((> i m) nil)
                                (tagbody
                                  (setf temp
                                          (f2cl-lib:fref a-%data%
                                                         (i (f2cl-lib:int-add j 1))
                                                         ((1 lda) (1 *))
                                                         a-%offset%))
                                  (setf (f2cl-lib:fref a-%data%
                                                       (i (f2cl-lib:int-add j 1))
                                                       ((1 lda) (1 *))
```

```
                                        a-%offset%)
                                (- (* ctemp temp)
                                   (* stemp
                                      (f2cl-lib:fref a-%data%
                                                     (i j)
                                                     ((1 lda) (1 *))
                                                     a-%offset%))))
                            (setf (f2cl-lib:fref a-%data%
                                                 (i j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%)
                                  (+ (* stemp temp)
                                     (* ctemp
                                        (f2cl-lib:fref a-%data%
                                                       (i j)
                                                       ((1 lda) (1 *))
                                                       a-%offset%))))))))
          ((char-equal direct #\B)
           (f2cl-lib:fdo (j (f2cl-lib:int-add n (f2cl-lib:int-sub 1))
                          (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                         ((> j 1) nil)
             (tagbody
               (setf ctemp
                       (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
               (setf stemp
                       (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
               (cond
                 ((or (/= ctemp one) (/= stemp zero))
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i m) nil)
                    (tagbody
                      (setf temp
                              (f2cl-lib:fref a-%data%
                                             (i (f2cl-lib:int-add j 1))
                                             ((1 lda) (1 *))
                                             a-%offset%))
                      (setf (f2cl-lib:fref a-%data%
                                           (i (f2cl-lib:int-add j 1))
                                           ((1 lda) (1 *))
                                           a-%offset%)
                            (- (* ctemp temp)
                               (* stemp
                                  (f2cl-lib:fref a-%data%
                                                 (i j)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))))
```

```
                    (setf (f2cl-lib:fref a-%data%
                                         (i j)
                                         ((1 lda) (1 *))
                                         a-%offset%)
                          (+ (* stemp temp)
                             (* ctemp
                                (f2cl-lib:fref a-%data%
                                               (i j)
                                               ((1 lda) (1 *))
                                               a-%offset%))))))))))))
            ((char-equal pivot #\T)
             (cond
               ((char-equal direct #\F)
                (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                              ((> j n) nil)
                  (tagbody
                    (setf ctemp
                            (f2cl-lib:fref c-%data%
                                           ((f2cl-lib:int-sub j 1))
                                           ((1 *))
                                           c-%offset%))
                    (setf stemp
                            (f2cl-lib:fref s-%data%
                                           ((f2cl-lib:int-sub j 1))
                                           ((1 *))
                                           s-%offset%))
                    (cond
                      ((or (/= ctemp one) (/= stemp zero))
                       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                     ((> i m) nil)
                         (tagbody
                           (setf temp
                                   (f2cl-lib:fref a-%data%
                                                  (i j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))
                           (setf (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%)
                                 (- (* ctemp temp)
                                    (* stemp
                                       (f2cl-lib:fref a-%data%
                                                      (i 1)
                                                      ((1 lda) (1 *))
                                                      a-%offset%))))
```

```
                (setf (f2cl-lib:fref a-%data%
                                     (i 1)
                                     ((1 lda) (1 *))
                                     a-%offset%)
                      (+ (* stemp temp)
                         (* ctemp
                            (f2cl-lib:fref a-%data%
                                           (i 1)
                                           ((1 lda) (1 *))
                                           a-%offset%))))))))))
          ((char-equal direct #\B)
           (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                         ((> j 2) nil)
             (tagbody
               (setf ctemp
                       (f2cl-lib:fref c-%data%
                                      ((f2cl-lib:int-sub j 1))
                                      ((1 *))
                                      c-%offset%))
               (setf stemp
                       (f2cl-lib:fref s-%data%
                                      ((f2cl-lib:int-sub j 1))
                                      ((1 *))
                                      s-%offset%))
               (cond
                 ((or (/= ctemp one) (/= stemp zero))
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                ((> i m) nil)
                    (tagbody
                      (setf temp
                              (f2cl-lib:fref a-%data%
                                             (i j)
                                             ((1 lda) (1 *))
                                             a-%offset%))
                      (setf (f2cl-lib:fref a-%data%
                                           (i j)
                                           ((1 lda) (1 *))
                                           a-%offset%)
                            (- (* ctemp temp)
                               (* stemp
                                  (f2cl-lib:fref a-%data%
                                                 (i 1)
                                                 ((1 lda) (1 *))
                                                 a-%offset%))))
                      (setf (f2cl-lib:fref a-%data%
                                           (i 1)
```

```
                                        ((1 lda) (1 *))
                                        a-%offset%)
                                (+ (* stemp temp)
                                   (* ctemp
                                      (f2cl-lib:fref a-%data%
                                                     (i 1)
                                                     ((1 lda) (1 *))
                                                     a-%offset%)))))))))))
          ((char-equal pivot #\B)
           (cond
             ((char-equal direct #\F)
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j (f2cl-lib:int-add n (f2cl-lib:int-sub 1)))
                             nil)
                (tagbody
                  (setf ctemp
                          (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
                  (setf stemp
                          (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
                  (cond
                    ((or (/= ctemp one) (/= stemp zero))
                     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                   ((> i m) nil)
                       (tagbody
                         (setf temp
                                 (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%))
                         (setf (f2cl-lib:fref a-%data%
                                              (i j)
                                              ((1 lda) (1 *))
                                              a-%offset%)
                                 (+
                                  (* stemp
                                     (f2cl-lib:fref a-%data%
                                                    (i n)
                                                    ((1 lda) (1 *))
                                                    a-%offset%))
                                  (* ctemp temp)))
                         (setf (f2cl-lib:fref a-%data%
                                              (i n)
                                              ((1 lda) (1 *))
                                              a-%offset%)
                                 (-
                                  (* ctemp
```

```
                              (f2cl-lib:fref a-%data%
                                             (i n)
                                             ((1 lda) (1 *))
                                             a-%offset%))
                       (* stemp temp))))))))))
      ((char-equal direct #\B)
       (f2cl-lib:fdo (j (f2cl-lib:int-add n (f2cl-lib:int-sub 1))
                      (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                     ((> j 1) nil)
         (tagbody
           (setf ctemp
                   (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
           (setf stemp
                   (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
           (cond
             ((or (/= ctemp one) (/= stemp zero))
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                            ((> i m) nil)
                (tagbody
                  (setf temp
                          (f2cl-lib:fref a-%data%
                                         (i j)
                                         ((1 lda) (1 *))
                                         a-%offset%))
                  (setf (f2cl-lib:fref a-%data%
                                       (i j)
                                       ((1 lda) (1 *))
                                       a-%offset%)
                          (+
                           (* stemp
                              (f2cl-lib:fref a-%data%
                                             (i n)
                                             ((1 lda) (1 *))
                                             a-%offset%))
                           (* ctemp temp)))
                  (setf (f2cl-lib:fref a-%data%
                                       (i n)
                                       ((1 lda) (1 *))
                                       a-%offset%)
                          (-
                           (* ctemp
                              (f2cl-lib:fref a-%data%
                                             (i n)
                                             ((1 lda) (1 *))
                                             a-%offset%))
                           (* stemp temp)))))))))))))
```

```
   end_label
          (return (values nil nil nil nil nil nil nil nil nil))))))
```

7.74 dlasrt LAPACK

⟨*dlasrt.input*⟩≡

```
  )set break resume
  )sys rm -f dlasrt.output
  )spool dlasrt.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dlasrt.help*⟩≡

```
  ==========================================================================
  dlasrt examples
  ==========================================================================

  ==========================================================================
  Man Page Details
  ==========================================================================

  NAME
         DLASRT - number in D in increasing order (if ID = 'I') or in decreasing
         order (if ID = 'D' )

  SYNOPSIS
         SUBROUTINE DLASRT( ID, N, D, INFO )

             CHARACTER      ID

             INTEGER        INFO, N

             DOUBLE         PRECISION D( * )

  PURPOSE
         Sort the numbers in D in increasing order (if ID = 'I') or in  decreas-
         ing order (if ID = 'D' ).

         Use Quick Sort, reverting to Insertion sort on arrays of
         size <= 20. Dimension of STACK limits N to about 2**32.


  ARGUMENTS
         ID      (input) CHARACTER*1
                 = 'I': sort D in increasing order;
                 = 'D': sort D in decreasing order.

         N       (input) INTEGER
                 The length of the array D.

         D       (input/output) DOUBLE PRECISION array, dimension (N)
                 On  entry,  the array to be sorted.  On exit, D has been sorted
                 into increasing order (D(1) <= ... <= D(N) ) or into decreasing
                 order (D(1) >= ... >= D(N) ), depending on ID.

         INFO    (output) INTEGER
                 = 0:  successful exit
                 < 0:  if INFO = -i, the i-th argument had an illegal value
```

⟨*LAPACK dlasrt*⟩≡

```
(let* ((select 20))
  (declare (type (fixnum 20 20) select))
  (defun dlasrt (id n d info)
    (declare (type (simple-array double-float (*)) d)
             (type fixnum info n)
             (type character id))
    (f2cl-lib:with-multi-array-data
        ((id character id-%data% id-%offset%)
         (d double-float d-%data% d-%offset%))
      (prog ((stack (make-array 64 :element-type 'fixnum)) (d1 0.0)
             (d2 0.0) (d3 0.0) (dmnmx 0.0) (tmp 0.0) (dir 0) (endd 0) (i 0)
             (j 0) (start 0) (stkpnt 0))
        (declare (type (simple-array fixnum (64)) stack)
                 (type (double-float) d1 d2 d3 dmnmx tmp)
                 (type fixnum dir endd i j start stkpnt))
        (setf info 0)
        (setf dir -1)
        (cond
          ((char-equal id #\D)
           (setf dir 0))
          ((char-equal id #\I)
           (setf dir 1)))
        (cond
          ((= dir (f2cl-lib:int-sub 1))
           (setf info -1))
          ((< n 0)
           (setf info -2)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DLASRT" (f2cl-lib:int-sub info))
           (go end_label)))
        (if (<= n 1) (go end_label))
        (setf stkpnt 1)
        (setf (f2cl-lib:fref stack (1 1) ((1 2) (1 32))) 1)
        (setf (f2cl-lib:fref stack (2 1) ((1 2) (1 32))) n)
  label10
        (setf start (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32))))
        (setf endd (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))))
        (setf stkpnt (f2cl-lib:int-sub stkpnt 1))
        (cond
          ((and (<= (f2cl-lib:int-add endd (f2cl-lib:int-sub start)) select)
                (> (f2cl-lib:int-add endd (f2cl-lib:int-sub start)) 0))
           (cond
```

```
                ((= dir 0)
                 (f2cl-lib:fdo (i (f2cl-lib:int-add start 1)
                                  (f2cl-lib:int-add i 1))
                               ((> i endd) nil)
                   (tagbody
                     (f2cl-lib:fdo (j i (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                   ((> j (f2cl-lib:int-add start 1)) nil)
                       (tagbody
                         (cond
                           ((> (f2cl-lib:fref d (j) ((1 *)))
                               (f2cl-lib:fref d
                                              ((f2cl-lib:int-add j
                                                                 (f2cl-lib:int-sub
                                                                  1)))
                                              ((1 *))))
                            (setf dmnmx
                                    (f2cl-lib:fref d-%data%
                                                   (j)
                                                   ((1 *))
                                                   d-%offset%))
                            (setf (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
                                    (f2cl-lib:fref d-%data%
                                                   ((f2cl-lib:int-sub j 1))
                                                   ((1 *))
                                                   d-%offset%))
                            (setf (f2cl-lib:fref d-%data%
                                                 ((f2cl-lib:int-sub j 1))
                                                 ((1 *))
                                                 d-%offset%)
                                    dmnmx))
                           (t
                            (go label30)))))))
                 label30)))
                ((= dir 1)
                 (f2cl-lib:fdo (i (f2cl-lib:int-add start 1)
                                  (f2cl-lib:int-add i 1))
                               ((> i endd) nil)
                   (tagbody
                     (f2cl-lib:fdo (j i (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                   ((> j (f2cl-lib:int-add start 1)) nil)
                       (tagbody
                         (cond
                           ((< (f2cl-lib:fref d (j) ((1 *)))
                               (f2cl-lib:fref d
                                              ((f2cl-lib:int-add j
                                                                 (f2cl-lib:int-sub
```

```
                                                          1)))
                                        ((1 *))))
                          (setf dmnmx
                                  (f2cl-lib:fref d-%data%
                                                 (j)
                                                 ((1 *))
                                                 d-%offset%))
                          (setf (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
                                  (f2cl-lib:fref d-%data%
                                                 ((f2cl-lib:int-sub j 1))
                                                 ((1 *))
                                                 d-%offset%))
                          (setf (f2cl-lib:fref d-%data%
                                               ((f2cl-lib:int-sub j 1))
                                               ((1 *))
                                               d-%offset%)
                                  dmnmx))
                         (t
                          (go label50)))))
  label50)))))
           ((> (f2cl-lib:int-add endd (f2cl-lib:int-sub start)) select)
            (setf d1 (f2cl-lib:fref d-%data% (start) ((1 *)) d-%offset%))
            (setf d2 (f2cl-lib:fref d-%data% (endd) ((1 *)) d-%offset%))
            (setf i (the fixnum (truncate (+ start endd) 2)))
            (setf d3 (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
            (cond
              ((< d1 d2)
               (cond
                 ((< d3 d1)
                  (setf dmnmx d1))
                 ((< d3 d2)
                  (setf dmnmx d3))
                 (t
                  (setf dmnmx d2))))
              (t
               (cond
                 ((< d3 d2)
                  (setf dmnmx d2))
                 ((< d3 d1)
                  (setf dmnmx d3))
                 (t
                  (setf dmnmx d1)))))
            (cond
              ((= dir 0)
               (tagbody
                 (setf i (f2cl-lib:int-sub start 1))
```

```
                (setf j (f2cl-lib:int-add endd 1))
  label60
                (setf j (f2cl-lib:int-sub j 1))
                (if (< (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%) dmnmx)
                    (go label60))
  label80
                (setf i (f2cl-lib:int-add i 1))
                (if (> (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) dmnmx)
                    (go label80))
                (cond
                  ((< i j)
                   (setf tmp (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
                   (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                           (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))
                   (setf (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%) tmp)
                   (go label60)))
                (cond
                  ((> (f2cl-lib:int-add j (f2cl-lib:int-sub start))
                      (f2cl-lib:int-add endd
                                        (f2cl-lib:int-sub j)
                                        (f2cl-lib:int-sub 1)))
                   (setf stkpnt (f2cl-lib:int-add stkpnt 1))
                   (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32))) start)
                   (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))) j)
                   (setf stkpnt (f2cl-lib:int-add stkpnt 1))
                   (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32)))
                           (f2cl-lib:int-add j 1))
                   (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))) endd))
                  (t
                   (setf stkpnt (f2cl-lib:int-add stkpnt 1))
                   (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32)))
                           (f2cl-lib:int-add j 1))
                   (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))) endd)
                   (setf stkpnt (f2cl-lib:int-add stkpnt 1))
                   (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32))) start)
                   (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))) j)))))
               (t
                (tagbody
                  (setf i (f2cl-lib:int-sub start 1))
                  (setf j (f2cl-lib:int-add endd 1))
  label90
                  (setf j (f2cl-lib:int-sub j 1))
                  (if (> (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%) dmnmx)
                      (go label90))
  label110
                  (setf i (f2cl-lib:int-add i 1))
```

```
                (if (< (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) dmnmx)
                    (go label110))
                (cond
                  ((< i j)
                   (setf tmp (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
                   (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                           (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))
                   (setf (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%) tmp)
                   (go label90)))
                (cond
                  ((> (f2cl-lib:int-add j (f2cl-lib:int-sub start))
                      (f2cl-lib:int-add endd
                                        (f2cl-lib:int-sub j)
                                        (f2cl-lib:int-sub 1)))
                   (setf stkpnt (f2cl-lib:int-add stkpnt 1))
                   (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32))) start)
                   (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))) j)
                   (setf stkpnt (f2cl-lib:int-add stkpnt 1))
                   (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32)))
                           (f2cl-lib:int-add j 1))
                   (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))) endd))
                  (t
                   (setf stkpnt (f2cl-lib:int-add stkpnt 1))
                   (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32)))
                           (f2cl-lib:int-add j 1))
                   (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))) endd)
                   (setf stkpnt (f2cl-lib:int-add stkpnt 1))
                   (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32))) start)
                   (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32)))
                           j))))))
        (if (> stkpnt 0) (go label10))
  end_label
        (return (values nil nil nil info))))))
```

7.75 dlassq LAPACK

⟨*dlassq.input*⟩≡

```
)set break resume
)sys rm -f dlassq.output
)spool dlassq.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlassq.help*⟩≡

```
  ======================================================================
  dlassq examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLASSQ  -  the  values  scl and smsq such that   ( scl**2 )*smsq = x( 1
         )**2 +...+ x( n )**2 + ( scale**2 )*sumsq,

  SYNOPSIS
         SUBROUTINE DLASSQ( N, X, INCX, SCALE, SUMSQ )

             INTEGER        INCX, N

             DOUBLE         PRECISION SCALE, SUMSQ

             DOUBLE         PRECISION X( * )

  PURPOSE
         DLASSQ  returns the values  scl  and  smsq  such that

         where  x( i ) = X( 1 + ( i - 1  )*INCX  ).  The  value  of   sumsq   is
         assumed to be non-negative and  scl  returns the value

            scl = max( scale, abs( x( i ) ) ).

         scale and sumsq must be supplied in SCALE and SUMSQ and
         scl and smsq are overwritten on SCALE and SUMSQ respectively.

         The routine makes only one pass through the vector x.


  ARGUMENTS
         N       (input) INTEGER
                 The number of elements to be used from the vector X.

         X       (input) DOUBLE PRECISION array, dimension (N)
                 The vector for which a scaled sum of squares is computed.  x( i
                 )  = X( 1 + ( i - 1 )*INCX ), 1 <= i <= n.

         INCX    (input) INTEGER
                 The increment between successive values of the vector X.   INCX
```

```
        > 0.

SCALE   (input/output) DOUBLE PRECISION
        On entry, the value scale in the equation above. On exit,
        SCALE is overwritten with scl , the scaling factor for the sum
        of squares.

SUMSQ   (input/output) DOUBLE PRECISION
        On entry, the value sumsq in the equation above. On exit,
        SUMSQ is overwritten with smsq , the basic sum of squares from
        which scl has been factored out.
```

⟨*LAPACK dlassq*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dlassq (n x incx scale sumsq)
    (declare (type (double-float) sumsq scale)
             (type (simple-array double-float (*)) x)
             (type fixnum incx n))
    (f2cl-lib:with-multi-array-data
        ((x double-float x-%data% x-%offset%))
      (prog ((absxi 0.0) (ix 0))
        (declare (type (double-float) absxi) (type fixnum ix))
        (cond
          ((> n 0)
           (f2cl-lib:fdo (ix 1 (f2cl-lib:int-add ix incx))
                         ((> ix
                             (f2cl-lib:int-add 1
                                               (f2cl-lib:int-mul
                                                (f2cl-lib:int-add n
                                                 (f2cl-lib:int-sub 1))
                                                incx)))
                          nil)
             (tagbody
               (cond
                 ((/= (f2cl-lib:fref x (ix) ((1 *))) zero)
                  (setf absxi
                          (abs
                           (f2cl-lib:fref x-%data% (ix) ((1 *)) x-%offset%)))
                  (cond
                    ((< scale absxi)
                     (setf sumsq (+ 1 (* sumsq (expt (/ scale absxi) 2))))
                     (setf scale absxi))
                    (t
                     (setf sumsq (+ sumsq (expt (/ absxi scale) 2))))))))))
        (return (values nil nil nil scale sumsq))))))
```

7.76 dlasv2 LAPACK

⟨*dlasv2.input*⟩≡

```
)set break resume
)sys rm -f dlasv2.output
)spool dlasv2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasv2.help*⟩≡

```
  ======================================================================
  dlasv2 examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLASV2 - the singular value decomposition of a 2-by-2 triangular matrix
         [ F G ]  [ 0 H ]

  SYNOPSIS
         SUBROUTINE DLASV2( F, G, H, SSMIN, SSMAX, SNR, CSR, SNL, CSL )

             DOUBLE         PRECISION CSL, CSR, F, G, H, SNL, SNR, SSMAX, SSMIN

  PURPOSE
         DLASV2 computes the singular value decomposition of a 2-by-2 triangular
         matrix
            [  F   G  ]
            [  0   H  ]. On return, abs(SSMAX) is the larger singular value,
         abs(SSMIN) is the smaller singular value, and (CSL,SNL) and (CSR,SNR)
         are the left and right singular vectors for abs(SSMAX), giving the
         decomposition

            [ CSL  SNL ] [  F   G  ] [ CSR -SNR ]  =  [ SSMAX   0   ]
            [-SNL  CSL ] [  0   H  ] [ SNR  CSR ]     [  0    SSMIN ].


  ARGUMENTS
         F       (input) DOUBLE PRECISION
                 The (1,1) element of the 2-by-2 matrix.

         G       (input) DOUBLE PRECISION
                 The (1,2) element of the 2-by-2 matrix.

         H       (input) DOUBLE PRECISION
                 The (2,2) element of the 2-by-2 matrix.

         SSMIN   (output) DOUBLE PRECISION
                 abs(SSMIN) is the smaller singular value.

         SSMAX   (output) DOUBLE PRECISION
                 abs(SSMAX) is the larger singular value.
```

```
        SNL     (output) DOUBLE PRECISION
                CSL     (output) DOUBLE PRECISION The vector (CSL,  SNL)  is  a
                unit left singular vector for the singular value abs(SSMAX).

        SNR     (output) DOUBLE PRECISION
                CSR      (output)  DOUBLE  PRECISION The vector (CSR, SNR) is a
                unit right singular vector for the singular value abs(SSMAX).

    FURTHER DETAILS
        Any input parameter may be aliased with any output parameter.

        Barring over/underflow and assuming a guard digit in  subtraction,  all
        output  quantities  are correct to within a few units in the last place
        (ulps).

        In IEEE arithmetic, the code works correctly if one matrix  element  is
        infinite.

        Overflow  will not occur unless the largest singular value itself over-
        flows or is within a few ulps of overflow. (On  machines  with  partial
        overflow,  like  the  Cray,  overflow may occur if the largest singular
        value is within a factor of 2 of overflow.)

        Underflow is harmless if underflow is gradual. Otherwise,  results  may
        correspond  to  a  matrix  modified  by  perturbations of size near the
        underflow threshold.
```

⟨*LAPACK dlasv2*⟩≡

```
(let* ((zero 0.0) (half 0.5) (one 1.0) (two 2.0) (four 4.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 0.5 0.5) half)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two)
           (type (double-float 4.0 4.0) four))
  (defun dlasv2 (f g h ssmin ssmax snr csr snl csl)
    (declare (type (double-float) csl snl csr snr ssmax ssmin h g f))
    (prog ((a 0.0) (clt 0.0) (crt 0.0) (d 0.0) (fa 0.0) (ft 0.0) (ga 0.0)
           (gt 0.0) (ha 0.0) (ht 0.0) (l 0.0) (m 0.0) (mm 0.0) (r 0.0) (s 0.0)
           (slt 0.0) (srt 0.0) (t$ 0.0) (temp 0.0) (tsign 0.0) (tt 0.0)
           (pmax 0) (gasmal nil) (swap nil))
      (declare (type (double-float) a clt crt d fa ft ga gt ha ht l m mm r s
                                    slt srt t$ temp tsign tt)
               (type fixnum pmax)
               (type (member t nil) gasmal swap))
      (setf ft f)
      (setf fa (abs ft))
      (setf ht h)
      (setf ha (abs h))
      (setf pmax 1)
      (setf swap (> ha fa))
      (cond
        (swap
         (setf pmax 3)
         (setf temp ft)
         (setf ft ht)
         (setf ht temp)
         (setf temp fa)
         (setf fa ha)
         (setf ha temp)))
      (setf gt g)
      (setf ga (abs gt))
      (cond
        ((= ga zero)
         (setf ssmin ha)
         (setf ssmax fa)
         (setf clt one)
         (setf crt one)
         (setf slt zero)
         (setf srt zero))
        (t
         (setf gasmal t)
         (cond
           ((> ga fa)
```

```
    (setf pmax 2)
    (cond
      ((< (f2cl-lib:f2cl/ fa ga) (dlamch "EPS"))
       (setf gasmal nil)
       (setf ssmax ga)
       (cond
         ((> ha one)
          (setf ssmin (/ fa (/ ga ha))))
         (t
          (setf ssmin (* (/ fa ga) ha))))
       (setf clt one)
       (setf slt (/ ht gt))
       (setf srt one)
       (setf crt (/ ft gt))))))
  (cond
    (gasmal
     (setf d (- fa ha))
     (cond
       ((= d fa)
        (setf l one))
       (t
        (setf l (/ d fa))))
     (setf m (/ gt ft))
     (setf t$ (- two l))
     (setf mm (* m m))
     (setf tt (* t$ t$))
     (setf s (f2cl-lib:fsqrt (+ tt mm)))
     (cond
       ((= l zero)
        (setf r (abs m)))
       (t
        (setf r (f2cl-lib:fsqrt (+ (* l l) mm)))))
     (setf a (* half (+ s r)))
     (setf ssmin (/ ha a))
     (setf ssmax (* fa a))
     (cond
       ((= mm zero)
        (cond
          ((= l zero)
           (setf t$ (* (f2cl-lib:sign two ft) (f2cl-lib:sign one gt))))
          (t
           (setf t$ (+ (/ gt (f2cl-lib:sign d ft)) (/ m t$))))))
       (t
        (setf t$ (* (+ (/ m (+ s t$)) (/ m (+ r l))) (+ one a)))))
     (setf l (f2cl-lib:fsqrt (+ (* t$ t$) four)))
     (setf crt (/ two l))
```

```
      (setf srt (/ t$ l))
      (setf clt (/ (+ crt (* srt m)) a))
      (setf slt (/ (* (/ ht ft) srt) a))))))
(cond
  (swap
   (setf csl srt)
   (setf snl crt)
   (setf csr slt)
   (setf snr clt))
  (t
   (setf csl clt)
   (setf snl slt)
   (setf csr crt)
   (setf snr srt)))
(if (= pmax 1)
    (setf tsign
            (* (f2cl-lib:sign one csr)
               (f2cl-lib:sign one csl)
               (f2cl-lib:sign one f))))
(if (= pmax 2)
    (setf tsign
            (* (f2cl-lib:sign one snr)
               (f2cl-lib:sign one csl)
               (f2cl-lib:sign one g))))
(if (= pmax 3)
    (setf tsign
            (* (f2cl-lib:sign one snr)
               (f2cl-lib:sign one snl)
               (f2cl-lib:sign one h))))
(setf ssmax (f2cl-lib:sign ssmax tsign))
(setf ssmin
        (f2cl-lib:sign ssmin
                       (* tsign
                          (f2cl-lib:sign one f)
                          (f2cl-lib:sign one h))))
(return (values nil nil nil ssmin ssmax snr csr snl csl)))))
```

7.77 dlaswp LAPACK

⟨*dlaswp.input*⟩≡

```
)set break resume
)sys rm -f dlaswp.output
)spool dlaswp.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlaswp.help*⟩≡

```
  ======================================================================
  dlaswp examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DLASWP - a series of row interchanges on the matrix A

  SYNOPSIS
         SUBROUTINE DLASWP( N, A, LDA, K1, K2, IPIV, INCX )

             INTEGER        INCX, K1, K2, LDA, N

             INTEGER        IPIV( * )

             DOUBLE         PRECISION A( LDA, * )

  PURPOSE
         DLASWP  performs a series of row interchanges on the matrix A.  One row
         interchange is initiated for each of rows K1 through K2 of A.


  ARGUMENTS
         N       (input) INTEGER
                 The number of columns of the matrix A.

         A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
                 On entry, the matrix of column dimension N  to  which  the  row
                 interchanges will be applied.  On exit, the permuted matrix.

         LDA     (input) INTEGER
                 The leading dimension of the array A.

         K1      (input) INTEGER
                 The  first  element of IPIV for which a row interchange will be
                 done.

         K2      (input) INTEGER
                 The last element of IPIV for which a row  interchange  will  be
                 done.

         IPIV    (input) INTEGER array, dimension (K2*abs(INCX))
```

```
          The vector of pivot indices.  Only the elements in positions K1
          through K2 of IPIV are accessed.  IPIV(K) = L  implies  rows  K
          and L are to be interchanged.

  INCX    (input) INTEGER
          The  increment  between  successive values of IPIV.  If IPIV is
          negative, the pivots are applied in reverse order.
```

⟨*LAPACK dlaswp*⟩≡

```
(defun dlaswp (n a lda k1 k2 ipiv incx)
  (declare (type (simple-array fixnum (*)) ipiv)
           (type (simple-array double-float (*)) a)
           (type fixnum incx k2 k1 lda n))
  (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (ipiv fixnum ipiv-%data% ipiv-%offset%))
    (prog ((temp 0.0) (i 0) (i1 0) (i2 0) (inc 0) (ip 0) (ix 0) (ix0 0) (j 0)
           (k 0) (n32 0))
      (declare (type fixnum n32 k j ix0 ix ip inc i2 i1 i)
               (type (double-float) temp))
      (cond
        ((> incx 0)
         (setf ix0 k1)
         (setf i1 k1)
         (setf i2 k2)
         (setf inc 1))
        ((< incx 0)
         (setf ix0
                 (f2cl-lib:int-add 1
                                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 k2)
                                                     incx)))
         (setf i1 k2)
         (setf i2 k1)
         (setf inc -1))
        (t
         (go end_label)))
      (setf n32 (* (the fixnum (truncate n 32)) 32))
      (cond
        ((/= n32 0)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 32))
                       ((> j n32) nil)
           (tagbody
             (setf ix ix0)
             (f2cl-lib:fdo (i i1 (f2cl-lib:int-add i inc))
                           ((> i i2) nil)
               (tagbody
                 (setf ip
                         (f2cl-lib:fref ipiv-%data%
                                        (ix)
                                        ((1 *))
                                        ipiv-%offset%))
                 (cond
                   ((/= ip i)
                    (f2cl-lib:fdo (k j (f2cl-lib:int-add k 1))
```

```
                                    ((> k (f2cl-lib:int-add j 31)) nil)
                          (tagbody
                            (setf temp
                                    (f2cl-lib:fref a-%data%
                                                   (i k)
                                                   ((1 lda) (1 *))
                                                   a-%offset%))
                            (setf (f2cl-lib:fref a-%data%
                                                 (i k)
                                                 ((1 lda) (1 *))
                                                 a-%offset%)
                                    (f2cl-lib:fref a-%data%
                                                   (ip k)
                                                   ((1 lda) (1 *))
                                                   a-%offset%))
                            (setf (f2cl-lib:fref a-%data%
                                                 (ip k)
                                                 ((1 lda) (1 *))
                                                 a-%offset%)
                                  temp)))))
                    (setf ix (f2cl-lib:int-add ix incx))))))))
        (cond
          ((/= n32 n)
           (setf n32 (f2cl-lib:int-add n32 1))
           (setf ix ix0)
           (f2cl-lib:fdo (i i1 (f2cl-lib:int-add i inc))
                         ((> i i2) nil)
             (tagbody
               (setf ip (f2cl-lib:fref ipiv-%data% (ix) ((1 *)) ipiv-%offset%))
               (cond
                 ((/= ip i)
                  (f2cl-lib:fdo (k n32 (f2cl-lib:int-add k 1))
                                ((> k n) nil)
                    (tagbody
                      (setf temp
                              (f2cl-lib:fref a-%data%
                                             (i k)
                                             ((1 lda) (1 *))
                                             a-%offset%))
                      (setf (f2cl-lib:fref a-%data%
                                           (i k)
                                           ((1 lda) (1 *))
                                           a-%offset%)
                              (f2cl-lib:fref a-%data%
                                             (ip k)
                                             ((1 lda) (1 *))
```

```
                                          a-%offset%))
                    (setf (f2cl-lib:fref a-%data%
                                         (ip k)
                                         ((1 lda) (1 *))
                                         a-%offset%)
                          temp)))))
             (setf ix (f2cl-lib:int-add ix incx))))))
  end_label
       (return (values nil nil nil nil nil nil nil)))))
```

7.78 dlasy2 LAPACK

⟨*dlasy2.input*⟩≡

```
)set break resume
)sys rm -f dlasy2.output
)spool dlasy2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dlasy2.help*⟩≡

```
=====================================================================
dlasy2 examples
=====================================================================

=====================================================================
Man Page Details
=====================================================================

NAME
       DLASY2  -  for  the N1 by N2 matrix X, 1 <= N1,N2 <= 2, in   op(TL)*X +
       ISGN*X*op(TR) = SCALE*B,

SYNOPSIS
       SUBROUTINE DLASY2( LTRANL, LTRANR, ISGN, N1, N2, TL, LDTL, TR, LDTR, B,
                          LDB, SCALE, X, LDX, XNORM, INFO )

           LOGICAL         LTRANL, LTRANR

           INTEGER         INFO, ISGN, LDB, LDTL, LDTR, LDX, N1, N2

           DOUBLE          PRECISION SCALE, XNORM

           DOUBLE          PRECISION B( LDB, * ), TL( LDTL, * ), TR( LDTR, * ),
                           X( LDX, * )

PURPOSE
       DLASY2 solves for the N1 by N2 matrix X, 1 <= N1,N2 <= 2, in

       where TL is N1 by N1, TR is N2 by N2, B is N1 by N2, and ISGN  =  1  or
       -1.  op(T) = T or T', where T' denotes the transpose of T.


ARGUMENTS
       LTRANL  (input) LOGICAL
               On  entry, LTRANL specifies the op(TL): = .FALSE., op(TL) = TL,
               = .TRUE., op(TL) = TL'.

       LTRANR  (input) LOGICAL
               On entry, LTRANR specifies the op(TR): = .FALSE., op(TR) =  TR,
               = .TRUE., op(TR) = TR'.

       ISGN    (input) INTEGER
               On  entry, ISGN specifies the sign of the equation as described
               before. ISGN may only be 1 or -1.
```

N1 (input) INTEGER
On entry, N1 specifies the order of matrix TL. N1 may only be 0, 1 or 2.

N2 (input) INTEGER
On entry, N2 specifies the order of matrix TR. N2 may only be 0, 1 or 2.

TL (input) DOUBLE PRECISION array, dimension (LDTL,2)
On entry, TL contains an N1 by N1 matrix.

LDTL (input) INTEGER
The leading dimension of the matrix TL. LDTL >= max(1,N1).

TR (input) DOUBLE PRECISION array, dimension (LDTR,2)
On entry, TR contains an N2 by N2 matrix.

LDTR (input) INTEGER
The leading dimension of the matrix TR. LDTR >= max(1,N2).

B (input) DOUBLE PRECISION array, dimension (LDB,2)
On entry, the N1 by N2 matrix B contains the right-hand side of the equation.

LDB (input) INTEGER
The leading dimension of the matrix B. LDB >= max(1,N1).

SCALE (output) DOUBLE PRECISION
On exit, SCALE contains the scale factor. SCALE is chosen less than or equal to 1 to prevent the solution overflowing.

X (output) DOUBLE PRECISION array, dimension (LDX,2)
On exit, X contains the N1 by N2 solution.

LDX (input) INTEGER
The leading dimension of the matrix X. LDX >= max(1,N1).

XNORM (output) DOUBLE PRECISION
On exit, XNORM is the infinity-norm of the solution.

INFO (output) INTEGER
On exit, INFO is set to 0: successful exit.
1: TL and TR have too close eigenvalues, so TL or TR is perturbed to get a nonsingular equation. NOTE: In the interests of speed, this routine does not check the inputs for errors.

⟨*LAPACK dlasy2*⟩≡

```
(let* ((zero 0.0) (one 1.0) (two 2.0) (half 0.5) (eight 8.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two)
           (type (double-float 0.5 0.5) half)
           (type (double-float 8.0 8.0) eight))
  (let ((locu12
         (make-array 4
                     :element-type 'fixnum
                     :initial-contents '(3 4 1 2)))
        (locl21
         (make-array 4
                     :element-type 'fixnum
                     :initial-contents '(2 1 4 3)))
        (locu22
         (make-array 4
                     :element-type 'fixnum
                     :initial-contents '(4 3 2 1)))
        (xswpiv
         (make-array 4 :element-type 't :initial-contents '(nil nil t t)))
        (bswpiv
         (make-array 4 :element-type 't :initial-contents '(nil t nil t))))
    (declare (type (simple-array (member t nil) (4)) bswpiv xswpiv)
             (type (simple-array fixnum (4)) locu22 locl21 locu12))
    (defun dlasy2
           (ltranl ltranr isgn n1 n2 tl ldtl tr ldtr b ldb$ scale x ldx xnorm
            info)
      (declare (type (double-float) xnorm scale)
               (type (simple-array double-float (*)) x b tr tl)
               (type fixnum info ldx ldb$ ldtr ldtl n2 n1 isgn)
               (type (member t nil) ltranr ltranl))
      (f2cl-lib:with-multi-array-data
          ((tl double-float tl-%data% tl-%offset%)
           (tr double-float tr-%data% tr-%offset%)
           (b double-float b-%data% b-%offset%)
           (x double-float x-%data% x-%offset%))
        (prog ((btmp (make-array 4 :element-type 'double-float))
               (t16 (make-array 16 :element-type 'double-float))
               (tmp (make-array 4 :element-type 'double-float))
               (x2 (make-array 2 :element-type 'double-float))
               (jpiv (make-array 4 :element-type 'fixnum)) (bet 0.0)
               (eps 0.0) (gam 0.0) (l21 0.0) (sgn 0.0) (smin 0.0) (smlnum 0.0)
               (tau1 0.0) (temp 0.0) (u11 0.0) (u12 0.0) (u22 0.0) (xmax 0.0)
               (i 0) (ip 0) (ipiv 0) (ipsv 0) (j 0) (jp 0) (jpsv 0) (k 0)
               (bswap nil) (xswap nil))
```

```
          (declare (type (simple-array double-float (16)) t16)
                   (type (simple-array double-float (4)) btmp tmp)
                   (type (simple-array double-float (2)) x2)
                   (type (simple-array fixnum (4)) jpiv)
                   (type (double-float) bet eps gam l21 sgn smin smlnum tau1
                                        temp u11 u12 u22 xmax)
                   (type fixnum i ip ipiv ipsv j jp jpsv k)
                   (type (member t nil) bswap xswap))
          (setf info 0)
          (if (or (= n1 0) (= n2 0)) (go end_label))
          (setf eps (dlamch "P"))
          (setf smlnum (/ (dlamch "S") eps))
          (setf sgn (coerce (the fixnum isgn) 'double-float))
          (setf k (f2cl-lib:int-sub (f2cl-lib:int-add n1 n1 n2) 2))
          (f2cl-lib:computed-goto (label10 label20 label30 label50) k)
  label10
          (setf tau1
                  (+
                   (f2cl-lib:fref tl-%data% (1 1) ((1 ldtl) (1 *)) tl-%offset%)
                   (* sgn
                      (f2cl-lib:fref tr-%data%
                                     (1 1)
                                     ((1 ldtr) (1 *))
                                     tr-%offset%))))
          (setf bet (abs tau1))
          (cond
            ((<= bet smlnum)
             (setf tau1 smlnum)
             (setf bet smlnum)
             (setf info 1)))
          (setf scale one)
          (setf gam
                  (abs
                   (f2cl-lib:fref b-%data% (1 1) ((1 ldb$) (1 *)) b-%offset%)))
          (if (> (* smlnum gam) bet) (setf scale (/ one gam)))
          (setf (f2cl-lib:fref x-%data% (1 1) ((1 ldx) (1 *)) x-%offset%)
                  (/
                   (*
                    (f2cl-lib:fref b-%data% (1 1) ((1 ldb$) (1 *)) b-%offset%)
                    scale)
                   tau1))
          (setf xnorm
                  (abs
                   (f2cl-lib:fref x-%data% (1 1) ((1 ldx) (1 *)) x-%offset%)))
          (go end_label)
  label20
```

```
(setf smin
        (max
         (* eps
            (max
             (abs
              (f2cl-lib:fref tl-%data%
                             (1 1)
                             ((1 ldtl) (1 *))
                             tl-%offset%))
             (abs
              (f2cl-lib:fref tr-%data%
                             (1 1)
                             ((1 ldtr) (1 *))
                             tr-%offset%))
             (abs
              (f2cl-lib:fref tr-%data%
                             (1 2)
                             ((1 ldtr) (1 *))
                             tr-%offset%))
             (abs
              (f2cl-lib:fref tr-%data%
                             (2 1)
                             ((1 ldtr) (1 *))
                             tr-%offset%))
             (abs
              (f2cl-lib:fref tr-%data%
                             (2 2)
                             ((1 ldtr) (1 *))
                             tr-%offset%))))
         smlnum))
(setf (f2cl-lib:fref tmp (1) ((1 4)))
        (+
         (f2cl-lib:fref tl-%data% (1 1) ((1 ldtl) (1 *)) tl-%offset%)
         (* sgn
            (f2cl-lib:fref tr-%data%
                           (1 1)
                           ((1 ldtr) (1 *))
                           tr-%offset%))))
(setf (f2cl-lib:fref tmp (4) ((1 4)))
        (+
         (f2cl-lib:fref tl-%data% (1 1) ((1 ldtl) (1 *)) tl-%offset%)
         (* sgn
            (f2cl-lib:fref tr-%data%
                           (2 2)
                           ((1 ldtr) (1 *))
                           tr-%offset%))))
```

```
            (cond
              (ltranr
               (setf (f2cl-lib:fref tmp (2) ((1 4)))
                       (* sgn
                          (f2cl-lib:fref tr-%data%
                                         (2 1)
                                         ((1 ldtr) (1 *))
                                         tr-%offset%)))
               (setf (f2cl-lib:fref tmp (3) ((1 4)))
                       (* sgn
                          (f2cl-lib:fref tr-%data%
                                         (1 2)
                                         ((1 ldtr) (1 *))
                                         tr-%offset%))))
              (t
               (setf (f2cl-lib:fref tmp (2) ((1 4)))
                       (* sgn
                          (f2cl-lib:fref tr-%data%
                                         (1 2)
                                         ((1 ldtr) (1 *))
                                         tr-%offset%)))
               (setf (f2cl-lib:fref tmp (3) ((1 4)))
                       (* sgn
                          (f2cl-lib:fref tr-%data%
                                         (2 1)
                                         ((1 ldtr) (1 *))
                                         tr-%offset%)))))
            (setf (f2cl-lib:fref btmp (1) ((1 4)))
                    (f2cl-lib:fref b-%data% (1 1) ((1 ldb$) (1 *)) b-%offset%))
            (setf (f2cl-lib:fref btmp (2) ((1 4)))
                    (f2cl-lib:fref b-%data% (1 2) ((1 ldb$) (1 *)) b-%offset%))
            (go label40)
  label30
            (setf smin
                    (max
                     (* eps
                        (max
                         (abs
                          (f2cl-lib:fref tr-%data%
                                         (1 1)
                                         ((1 ldtr) (1 *))
                                         tr-%offset%))
                         (abs
                          (f2cl-lib:fref tl-%data%
                                         (1 1)
                                         ((1 ldtl) (1 *))
```

```
                                tl-%offset%))
                    (abs
                     (f2cl-lib:fref tl-%data%
                                    (1 2)
                                    ((1 ldtl) (1 *))
                                    tl-%offset%))
                    (abs
                     (f2cl-lib:fref tl-%data%
                                    (2 1)
                                    ((1 ldtl) (1 *))
                                    tl-%offset%))
                    (abs
                     (f2cl-lib:fref tl-%data%
                                    (2 2)
                                    ((1 ldtl) (1 *))
                                    tl-%offset%))))
              smlnum))
(setf (f2cl-lib:fref tmp (1) ((1 4)))
        (+
         (f2cl-lib:fref tl-%data% (1 1) ((1 ldtl) (1 *)) tl-%offset%)
         (* sgn
            (f2cl-lib:fref tr-%data%
                           (1 1)
                           ((1 ldtr) (1 *))
                           tr-%offset%))))
(setf (f2cl-lib:fref tmp (4) ((1 4)))
        (+
         (f2cl-lib:fref tl-%data% (2 2) ((1 ldtl) (1 *)) tl-%offset%)
         (* sgn
            (f2cl-lib:fref tr-%data%
                           (1 1)
                           ((1 ldtr) (1 *))
                           tr-%offset%))))
(cond
  (ltranl
   (setf (f2cl-lib:fref tmp (2) ((1 4)))
           (f2cl-lib:fref tl-%data%
                          (1 2)
                          ((1 ldtl) (1 *))
                          tl-%offset%))
   (setf (f2cl-lib:fref tmp (3) ((1 4)))
           (f2cl-lib:fref tl-%data%
                          (2 1)
                          ((1 ldtl) (1 *))
                          tl-%offset%)))
  (t
```

```
              (setf (f2cl-lib:fref tmp (2) ((1 4)))
                    (f2cl-lib:fref tl-%data%
                                   (2 1)
                                   ((1 ldtl) (1 *))
                                   tl-%offset%))
              (setf (f2cl-lib:fref tmp (3) ((1 4)))
                    (f2cl-lib:fref tl-%data%
                                   (1 2)
                                   ((1 ldtl) (1 *))
                                   tl-%offset%))))
            (setf (f2cl-lib:fref btmp (1) ((1 4)))
                  (f2cl-lib:fref b-%data% (1 1) ((1 ldb$) (1 *)) b-%offset%))
            (setf (f2cl-lib:fref btmp (2) ((1 4)))
                  (f2cl-lib:fref b-%data% (2 1) ((1 ldb$) (1 *)) b-%offset%))
  label40
            (setf ipiv (idamax 4 tmp 1))
            (setf u11 (f2cl-lib:fref tmp (ipiv) ((1 4))))
            (cond
              ((<= (abs u11) smin)
               (setf info 1)
               (setf u11 smin)))
            (setf u12
                  (f2cl-lib:fref tmp
                                 ((f2cl-lib:fref locu12 (ipiv) ((1 4))))
                                 ((1 4))))
            (setf l21
                  (/
                   (f2cl-lib:fref tmp
                                  ((f2cl-lib:fref locl21 (ipiv) ((1 4))))
                                  ((1 4)))
                   u11))
            (setf u22
                  (-
                   (f2cl-lib:fref tmp
                                  ((f2cl-lib:fref locu22 (ipiv) ((1 4))))
                                  ((1 4)))
                   (* u12 l21)))
            (setf xswap (f2cl-lib:fref xswpiv (ipiv) ((1 4))))
            (setf bswap (f2cl-lib:fref bswpiv (ipiv) ((1 4))))
            (cond
              ((<= (abs u22) smin)
               (setf info 1)
               (setf u22 smin)))
            (cond
              (bswap
               (setf temp (f2cl-lib:fref btmp (2) ((1 4))))
```

```
   (setf (f2cl-lib:fref btmp (2) ((1 4)))
           (- (f2cl-lib:fref btmp (1) ((1 4))) (* l21 temp)))
   (setf (f2cl-lib:fref btmp (1) ((1 4))) temp))
  (t
   (setf (f2cl-lib:fref btmp (2) ((1 4)))
           (- (f2cl-lib:fref btmp (2) ((1 4)))
              (* l21 (f2cl-lib:fref btmp (1) ((1 4))))))))
(setf scale one)
(cond
  ((or
    (> (* two smlnum (abs (f2cl-lib:fref btmp (2) ((1 4)))))
       (abs u22))
    (> (* two smlnum (abs (f2cl-lib:fref btmp (1) ((1 4)))))
       (abs u11)))
   (setf scale
           (/ half
              (max (abs (f2cl-lib:fref btmp (1) ((1 4))))
                   (abs (f2cl-lib:fref btmp (2) ((1 4))))))
   (setf (f2cl-lib:fref btmp (1) ((1 4)))
           (* (f2cl-lib:fref btmp (1) ((1 4))) scale))
   (setf (f2cl-lib:fref btmp (2) ((1 4)))
           (* (f2cl-lib:fref btmp (2) ((1 4))) scale))))
(setf (f2cl-lib:fref x2 (2) ((1 2)))
        (/ (f2cl-lib:fref btmp (2) ((1 4))) u22))
(setf (f2cl-lib:fref x2 (1) ((1 2)))
        (- (/ (f2cl-lib:fref btmp (1) ((1 4))) u11)
           (* (/ u12 u11) (f2cl-lib:fref x2 (2) ((1 2))))))
(cond
  (xswap
   (setf temp (f2cl-lib:fref x2 (2) ((1 2))))
   (setf (f2cl-lib:fref x2 (2) ((1 2)))
           (f2cl-lib:fref x2 (1) ((1 2))))
   (setf (f2cl-lib:fref x2 (1) ((1 2))) temp)))
(setf (f2cl-lib:fref x-%data% (1 1) ((1 ldx) (1 *)) x-%offset%)
        (f2cl-lib:fref x2 (1) ((1 2))))
(cond
  ((= n1 1)
   (setf (f2cl-lib:fref x-%data% (1 2) ((1 ldx) (1 *)) x-%offset%)
           (f2cl-lib:fref x2 (2) ((1 2))))
   (setf xnorm
           (+
            (abs
             (f2cl-lib:fref x-%data%
                            (1 1)
                            ((1 ldx) (1 *))
                            x-%offset%))
```

```
                        (abs
                         (f2cl-lib:fref x-%data%
                                        (1 2)
                                        ((1 ldx) (1 *))
                                        x-%offset%)))))
              (t
               (setf (f2cl-lib:fref x-%data% (2 1) ((1 ldx) (1 *)) x-%offset%)
                       (f2cl-lib:fref x2 (2) ((1 2))))
               (setf xnorm
                       (max
                        (abs
                         (f2cl-lib:fref x-%data%
                                        (1 1)
                                        ((1 ldx) (1 *))
                                        x-%offset%))
                        (abs
                         (f2cl-lib:fref x-%data%
                                        (2 1)
                                        ((1 ldx) (1 *))
                                        x-%offset%))))))
             (go end_label)
   label50
             (setf smin
                     (max
                      (abs
                       (f2cl-lib:fref tr-%data%
                                      (1 1)
                                      ((1 ldtr) (1 *))
                                      tr-%offset%))
                      (abs
                       (f2cl-lib:fref tr-%data%
                                      (1 2)
                                      ((1 ldtr) (1 *))
                                      tr-%offset%))
                      (abs
                       (f2cl-lib:fref tr-%data%
                                      (2 1)
                                      ((1 ldtr) (1 *))
                                      tr-%offset%))
                      (abs
                       (f2cl-lib:fref tr-%data%
                                      (2 2)
                                      ((1 ldtr) (1 *))
                                      tr-%offset%))))
             (setf smin
                     (max smin
```

```
                    (abs
                     (f2cl-lib:fref tl-%data%
                                    (1 1)
                                    ((1 ldtl) (1 *))
                                    tl-%offset%))
                    (abs
                     (f2cl-lib:fref tl-%data%
                                    (1 2)
                                    ((1 ldtl) (1 *))
                                    tl-%offset%))
                    (abs
                     (f2cl-lib:fref tl-%data%
                                    (2 1)
                                    ((1 ldtl) (1 *))
                                    tl-%offset%))
                    (abs
                     (f2cl-lib:fref tl-%data%
                                    (2 2)
                                    ((1 ldtl) (1 *))
                                    tl-%offset%))))
(setf smin (max (* eps smin) smlnum))
(setf (f2cl-lib:fref btmp (1) ((1 4))) zero)
(dcopy 16 btmp 0 t16 1)
(setf (f2cl-lib:fref t16 (1 1) ((1 4) (1 4)))
        (+
         (f2cl-lib:fref tl-%data% (1 1) ((1 ldtl) (1 *)) tl-%offset%)
         (* sgn
            (f2cl-lib:fref tr-%data%
                           (1 1)
                           ((1 ldtr) (1 *))
                           tr-%offset%))))
(setf (f2cl-lib:fref t16 (2 2) ((1 4) (1 4)))
        (+
         (f2cl-lib:fref tl-%data% (2 2) ((1 ldtl) (1 *)) tl-%offset%)
         (* sgn
            (f2cl-lib:fref tr-%data%
                           (1 1)
                           ((1 ldtr) (1 *))
                           tr-%offset%))))
(setf (f2cl-lib:fref t16 (3 3) ((1 4) (1 4)))
        (+
         (f2cl-lib:fref tl-%data% (1 1) ((1 ldtl) (1 *)) tl-%offset%)
         (* sgn
            (f2cl-lib:fref tr-%data%
                           (2 2)
                           ((1 ldtr) (1 *))
```

```
                              tr-%offset%))))
          (setf (f2cl-lib:fref t16 (4 4) ((1 4) (1 4)))
                (+
                 (f2cl-lib:fref tl-%data% (2 2) ((1 ldtl) (1 *)) tl-%offset%)
                 (* sgn
                    (f2cl-lib:fref tr-%data%
                                   (2 2)
                                   ((1 ldtr) (1 *))
                                   tr-%offset%))))
          (cond
            (ltranl
             (setf (f2cl-lib:fref t16 (1 2) ((1 4) (1 4)))
                     (f2cl-lib:fref tl-%data%
                                    (2 1)
                                    ((1 ldtl) (1 *))
                                    tl-%offset%))
             (setf (f2cl-lib:fref t16 (2 1) ((1 4) (1 4)))
                     (f2cl-lib:fref tl-%data%
                                    (1 2)
                                    ((1 ldtl) (1 *))
                                    tl-%offset%))
             (setf (f2cl-lib:fref t16 (3 4) ((1 4) (1 4)))
                     (f2cl-lib:fref tl-%data%
                                    (2 1)
                                    ((1 ldtl) (1 *))
                                    tl-%offset%))
             (setf (f2cl-lib:fref t16 (4 3) ((1 4) (1 4)))
                     (f2cl-lib:fref tl-%data%
                                    (1 2)
                                    ((1 ldtl) (1 *))
                                    tl-%offset%)))
            (t
             (setf (f2cl-lib:fref t16 (1 2) ((1 4) (1 4)))
                     (f2cl-lib:fref tl-%data%
                                    (1 2)
                                    ((1 ldtl) (1 *))
                                    tl-%offset%))
             (setf (f2cl-lib:fref t16 (2 1) ((1 4) (1 4)))
                     (f2cl-lib:fref tl-%data%
                                    (2 1)
                                    ((1 ldtl) (1 *))
                                    tl-%offset%))
             (setf (f2cl-lib:fref t16 (3 4) ((1 4) (1 4)))
                     (f2cl-lib:fref tl-%data%
                                    (1 2)
                                    ((1 ldtl) (1 *))
```

```
                              tl-%offset%))
      (setf (f2cl-lib:fref t16 (4 3) ((1 4) (1 4)))
            (f2cl-lib:fref tl-%data%
                           (2 1)
                           ((1 ldtl) (1 *))
                           tl-%offset%))))
    (cond
      (ltranr
       (setf (f2cl-lib:fref t16 (1 3) ((1 4) (1 4)))
             (* sgn
                (f2cl-lib:fref tr-%data%
                               (1 2)
                               ((1 ldtr) (1 *))
                               tr-%offset%)))
       (setf (f2cl-lib:fref t16 (2 4) ((1 4) (1 4)))
             (* sgn
                (f2cl-lib:fref tr-%data%
                               (1 2)
                               ((1 ldtr) (1 *))
                               tr-%offset%)))
       (setf (f2cl-lib:fref t16 (3 1) ((1 4) (1 4)))
             (* sgn
                (f2cl-lib:fref tr-%data%
                               (2 1)
                               ((1 ldtr) (1 *))
                               tr-%offset%)))
       (setf (f2cl-lib:fref t16 (4 2) ((1 4) (1 4)))
             (* sgn
                (f2cl-lib:fref tr-%data%
                               (2 1)
                               ((1 ldtr) (1 *))
                               tr-%offset%))))
      (t
       (setf (f2cl-lib:fref t16 (1 3) ((1 4) (1 4)))
             (* sgn
                (f2cl-lib:fref tr-%data%
                               (2 1)
                               ((1 ldtr) (1 *))
                               tr-%offset%)))
       (setf (f2cl-lib:fref t16 (2 4) ((1 4) (1 4)))
             (* sgn
                (f2cl-lib:fref tr-%data%
                               (2 1)
                               ((1 ldtr) (1 *))
                               tr-%offset%)))
       (setf (f2cl-lib:fref t16 (3 1) ((1 4) (1 4)))
```

```
            (* sgn
               (f2cl-lib:fref tr-%data%
                              (1 2)
                              ((1 ldtr) (1 *))
                              tr-%offset%)))
   (setf (f2cl-lib:fref t16 (4 2) ((1 4) (1 4)))
            (* sgn
               (f2cl-lib:fref tr-%data%
                              (1 2)
                              ((1 ldtr) (1 *))
                              tr-%offset%)))))
(setf (f2cl-lib:fref btmp (1) ((1 4)))
        (f2cl-lib:fref b-%data% (1 1) ((1 ldb$) (1 *)) b-%offset%))
(setf (f2cl-lib:fref btmp (2) ((1 4)))
        (f2cl-lib:fref b-%data% (2 1) ((1 ldb$) (1 *)) b-%offset%))
(setf (f2cl-lib:fref btmp (3) ((1 4)))
        (f2cl-lib:fref b-%data% (1 2) ((1 ldb$) (1 *)) b-%offset%))
(setf (f2cl-lib:fref btmp (4) ((1 4)))
        (f2cl-lib:fref b-%data% (2 2) ((1 ldb$) (1 *)) b-%offset%))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i 3) nil)
  (tagbody
    (setf xmax zero)
    (f2cl-lib:fdo (ip i (f2cl-lib:int-add ip 1))
                  ((> ip 4) nil)
      (tagbody
        (f2cl-lib:fdo (jp i (f2cl-lib:int-add jp 1))
                      ((> jp 4) nil)
          (tagbody
            (cond
              ((>= (abs (f2cl-lib:fref t16 (ip jp) ((1 4) (1 4))))
                   xmax)
               (setf xmax
                       (abs
                        (f2cl-lib:fref t16 (ip jp) ((1 4) (1 4)))))
               (setf ipsv ip)
               (setf jpsv jp)))))))
    (cond
      ((/= ipsv i)
       (dswap 4
        (f2cl-lib:array-slice t16
                              double-float
                              (ipsv 1)
                              ((1 4) (1 4)))
        4 (f2cl-lib:array-slice t16 double-float (i 1) ((1 4) (1 4)))
        4)
```

```
            (setf temp (f2cl-lib:fref btmp (i) ((1 4))))
            (setf (f2cl-lib:fref btmp (i) ((1 4)))
                    (f2cl-lib:fref btmp (ipsv) ((1 4))))
            (setf (f2cl-lib:fref btmp (ipsv) ((1 4))) temp)))
        (if (/= jpsv i)
            (dswap 4
             (f2cl-lib:array-slice t16
                                   double-float
                                   (1 jpsv)
                                   ((1 4) (1 4)))
             1
             (f2cl-lib:array-slice t16 double-float (1 i) ((1 4) (1 4)))
             1))
        (setf (f2cl-lib:fref jpiv (i) ((1 4))) jpsv)
        (cond
          ((< (abs (f2cl-lib:fref t16 (i i) ((1 4) (1 4)))) smin)
           (setf info 1)
           (setf (f2cl-lib:fref t16 (i i) ((1 4) (1 4))) smin)))
        (f2cl-lib:fdo (j (f2cl-lib:int-add i 1) (f2cl-lib:int-add j 1))
                      ((> j 4) nil)
          (tagbody
            (setf (f2cl-lib:fref t16 (j i) ((1 4) (1 4)))
                    (/ (f2cl-lib:fref t16 (j i) ((1 4) (1 4)))
                       (f2cl-lib:fref t16 (i i) ((1 4) (1 4)))))
            (setf (f2cl-lib:fref btmp (j) ((1 4)))
                    (- (f2cl-lib:fref btmp (j) ((1 4)))
                       (* (f2cl-lib:fref t16 (j i) ((1 4) (1 4)))
                          (f2cl-lib:fref btmp (i) ((1 4))))))
            (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
                             (f2cl-lib:int-add k 1))
                          ((> k 4) nil)
              (tagbody
                (setf (f2cl-lib:fref t16 (j k) ((1 4) (1 4)))
                   (- (f2cl-lib:fref t16 (j k) ((1 4) (1 4)))
                     (* (f2cl-lib:fref t16 (j i) ((1 4) (1 4)))
                       (f2cl-lib:fref t16 (i k) ((1 4) (1 4))))))))))))
    (if (< (abs (f2cl-lib:fref t16 (4 4) ((1 4) (1 4)))) smin)
        (setf (f2cl-lib:fref t16 (4 4) ((1 4) (1 4))) smin))
    (setf scale one)
    (cond
      ((or
        (> (* eight smlnum (abs (f2cl-lib:fref btmp (1) ((1 4)))))
           (abs (f2cl-lib:fref t16 (1 1) ((1 4) (1 4)))))
        (> (* eight smlnum (abs (f2cl-lib:fref btmp (2) ((1 4)))))
           (abs (f2cl-lib:fref t16 (2 2) ((1 4) (1 4)))))
        (> (* eight smlnum (abs (f2cl-lib:fref btmp (3) ((1 4)))))
```

```
          (abs (f2cl-lib:fref t16 (3 3) ((1 4) (1 4)))))
         (> (* eight smlnum (abs (f2cl-lib:fref btmp (4) ((1 4)))))
          (abs (f2cl-lib:fref t16 (4 4) ((1 4) (1 4))))))
        (setf scale
              (/ (/ one eight)
                 (max (abs (f2cl-lib:fref btmp (1) ((1 4))))
                      (abs (f2cl-lib:fref btmp (2) ((1 4))))
                      (abs (f2cl-lib:fref btmp (3) ((1 4))))
                      (abs (f2cl-lib:fref btmp (4) ((1 4)))))))
        (setf (f2cl-lib:fref btmp (1) ((1 4)))
              (* (f2cl-lib:fref btmp (1) ((1 4))) scale))
        (setf (f2cl-lib:fref btmp (2) ((1 4)))
              (* (f2cl-lib:fref btmp (2) ((1 4))) scale))
        (setf (f2cl-lib:fref btmp (3) ((1 4)))
              (* (f2cl-lib:fref btmp (3) ((1 4))) scale))
        (setf (f2cl-lib:fref btmp (4) ((1 4)))
              (* (f2cl-lib:fref btmp (4) ((1 4))) scale))))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i 4) nil)
        (tagbody
          (setf k (f2cl-lib:int-sub 5 i))
          (setf temp (/ one (f2cl-lib:fref t16 (k k) ((1 4) (1 4)))))
          (setf (f2cl-lib:fref tmp (k) ((1 4)))
                (* (f2cl-lib:fref btmp (k) ((1 4))) temp))
          (f2cl-lib:fdo (j (f2cl-lib:int-add k 1) (f2cl-lib:int-add j 1))
                        ((> j 4) nil)
            (tagbody
              (setf (f2cl-lib:fref tmp (k) ((1 4)))
                    (- (f2cl-lib:fref tmp (k) ((1 4)))
                       (* temp
                          (f2cl-lib:fref t16 (k j) ((1 4) (1 4)))
                          (f2cl-lib:fref tmp (j) ((1 4))))))))))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i 3) nil)
        (tagbody
          (cond
            ((/=
              (f2cl-lib:fref jpiv
                             ((f2cl-lib:int-add 4 (f2cl-lib:int-sub i)))
                             ((1 4)))
              (f2cl-lib:int-add 4 (f2cl-lib:int-sub i)))
             (setf temp
                   (f2cl-lib:fref tmp ((f2cl-lib:int-sub 4 i)) ((1 4))))
             (setf (f2cl-lib:fref tmp ((f2cl-lib:int-sub 4 i)) ((1 4)))
                   (f2cl-lib:fref tmp
                                  ((f2cl-lib:fref jpiv
```

```
                                                        ((f2cl-lib:int-sub 4
                                                                           i))
                                                        ((1 4))))
                                          ((1 4))))
                                (setf (f2cl-lib:fref tmp
                                                     ((f2cl-lib:fref jpiv
                                                                     ((f2cl-lib:int-sub 4 i))
                                                                     ((1 4))))
                                                     ((1 4)))
                                      temp)))))
                        (setf (f2cl-lib:fref x-%data% (1 1) ((1 ldx) (1 *)) x-%offset%)
                                (f2cl-lib:fref tmp (1) ((1 4))))
                        (setf (f2cl-lib:fref x-%data% (2 1) ((1 ldx) (1 *)) x-%offset%)
                                (f2cl-lib:fref tmp (2) ((1 4))))
                        (setf (f2cl-lib:fref x-%data% (1 2) ((1 ldx) (1 *)) x-%offset%)
                                (f2cl-lib:fref tmp (3) ((1 4))))
                        (setf (f2cl-lib:fref x-%data% (2 2) ((1 ldx) (1 *)) x-%offset%)
                                (f2cl-lib:fref tmp (4) ((1 4))))
                        (setf xnorm
                                (max
                                 (+ (abs (f2cl-lib:fref tmp (1) ((1 4))))
                                    (abs (f2cl-lib:fref tmp (3) ((1 4)))))
                                 (+ (abs (f2cl-lib:fref tmp (2) ((1 4))))
                                    (abs (f2cl-lib:fref tmp (4) ((1 4)))))))))
  end_label
                        (return
                         (values nil
                                 nil
                                 nil
                                 nil
                                 nil
                                 nil
                                 nil
                                 nil
                                 nil
                                 nil
                                 nil
                                 scale
                                 nil
                                 nil
                                 xnorm
                                 info)))))))
```

7.79 dorg2r LAPACK

⟨*dorg2r.input*⟩≡

```
)set break resume
)sys rm -f dorg2r.output
)spool dorg2r.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dorg2r.help*⟩≡

```
=========================================================================
dorg2r examples
=========================================================================

=========================================================================
Man Page Details
=========================================================================

NAME
       DORG2R - an m by n real matrix Q with orthonormal columns,

SYNOPSIS
       SUBROUTINE DORG2R( M, N, K, A, LDA, TAU, WORK, INFO )

           INTEGER        INFO, K, LDA, M, N

           DOUBLE         PRECISION A( LDA, * ), TAU( * ), WORK( * )

PURPOSE
       DORG2R  generates  an  m  by  n real matrix Q with orthonormal columns,
       which is defined as the first n columns of a product  of  k  elementary
       reflectors of order m

             Q  =  H(1) H(2) . . . H(k)

       as returned by DGEQRF.


ARGUMENTS
       M       (input) INTEGER
               The number of rows of the matrix Q. M >= 0.

       N       (input) INTEGER
               The number of columns of the matrix Q. M >= N >= 0.

       K       (input) INTEGER
               The  number  of elementary reflectors whose product defines the
               matrix Q. N >= K >= 0.

       A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
               On entry, the i-th column must contain the vector which defines
               the  elementary  reflector H(i), for i = 1,2,...,k, as returned
               by DGEQRF in the first k columns of its array argument  A.   On
               exit, the m-by-n matrix Q.
```

```
LDA     (input) INTEGER
        The first dimension of the array A. LDA >= max(1,M).

TAU     (input) DOUBLE PRECISION array, dimension (K)
        TAU(i) must contain the scalar factor of the elementary reflec-
        tor H(i), as returned by DGEQRF.

WORK    (workspace) DOUBLE PRECISION array, dimension (N)

INFO    (output) INTEGER
        = 0: successful exit
        < 0: if INFO = -i, the i-th argument has an illegal value
```

⟨*LAPACK dorg2r*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dorg2r (m n k a lda tau work info)
    (declare (type (simple-array double-float (*)) work tau a)
             (type fixnum info lda k n m))
    (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((i 0) (j 0) (l 0))
        (declare (type fixnum i j l))
        (setf info 0)
        (cond
          ((< m 0)
           (setf info -1))
          ((or (< n 0) (> n m))
           (setf info -2))
          ((or (< k 0) (> k n))
           (setf info -3))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info -5)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DORG2R" (f2cl-lib:int-sub info))
           (go end_label)))
        (if (<= n 0) (go end_label))
        (f2cl-lib:fdo (j (f2cl-lib:int-add k 1) (f2cl-lib:int-add j 1))
                      ((> j n) nil)
          (tagbody
            (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                          ((> l m) nil)
              (tagbody
                (setf (f2cl-lib:fref a-%data% (l j) ((1 lda) (1 *)) a-%offset%)
                      zero)))
            (setf (f2cl-lib:fref a-%data% (j j) ((1 lda) (1 *)) a-%offset%)
                  one)))
        (f2cl-lib:fdo (i k (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                      ((> i 1) nil)
          (tagbody
            (cond
              ((< i n)
               (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
```

```
                    one)
                (dlarf "Left" (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
                 (f2cl-lib:int-sub n i)
                 (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) 1
                 (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
                 (f2cl-lib:array-slice a
                                       double-float
                                       (i (f2cl-lib:int-add i 1))
                                       ((1 lda) (1 *)))
                 lda work)))
            (if (< i m)
                (dscal (f2cl-lib:int-sub m i)
                 (- (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
                 (f2cl-lib:array-slice a
                                       double-float
                                       ((+ i 1) i)
                                       ((1 lda) (1 *)))
                 1))
            (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                    (- one
                       (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)))
            (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                          ((> l (f2cl-lib:int-add i (f2cl-lib:int-sub 1))) nil)
              (tagbody
                (setf (f2cl-lib:fref a-%data% (l i) ((1 lda) (1 *)) a-%offset%)
                        zero)))))
  end_label
        (return (values nil nil nil nil nil nil nil info))))))
```

7.80 dorgbr LAPACK

⟨*dorgbr.input*⟩≡

```
)set break resume
)sys rm -f dorgbr.output
)spool dorgbr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dorgbr.help*⟩≡

```
======================================================================
dorgbr examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DORGBR  -  one  of the real orthogonal matrices Q or P**T determined by
       DGEBRD when reducing a real matrix A to bidiagonal form

SYNOPSIS
       SUBROUTINE DORGBR( VECT, M, N, K, A, LDA, TAU, WORK, LWORK, INFO )

           CHARACTER      VECT

           INTEGER        INFO, K, LDA, LWORK, M, N

           DOUBLE         PRECISION A( LDA, * ), TAU( * ), WORK( * )

PURPOSE
       DORGBR generates one of the real orthogonal matrices Q or  P**T  deter-
       mined by DGEBRD when reducing a real matrix A to bidiagonal form: A = Q
       * B * P**T.  Q and P**T are defined as products of  elementary  reflec-
       tors H(i) or G(i) respectively.

       If  VECT = 'Q', A is assumed to have been an M-by-K matrix, and Q is of
       order M:
       if m >= k, Q = H(1) H(2) . . . H(k) and  DORGBR  returns  the  first  n
       columns of Q, where m >= n >= k;
       if  m < k, Q = H(1) H(2) . . . H(m-1) and DORGBR returns Q as an M-by-M
       matrix.

       If VECT = 'P', A is assumed to have been a K-by-N matrix, and  P**T  is
       of order N:
       if  k  <  n, P**T = G(k) . . . G(2) G(1) and DORGBR returns the first m
       rows of P**T, where n >= m >= k;
       if k >= n, P**T = G(n-1) . . . G(2) G(1) and DORGBR returns P**T as  an
       N-by-N matrix.


ARGUMENTS
       VECT    (input) CHARACTER*1
               Specifies  whether the matrix Q or the matrix P**T is required,
```

```
          as defined in the transformation applied by DGEBRD:
          = 'Q':  generate Q;
          = 'P':  generate P**T.

M         (input) INTEGER
          The number of rows of the matrix Q or P**T to be  returned.   M
          >= 0.

N         (input) INTEGER
          The  number  of columns of the matrix Q or P**T to be returned.
          N >= 0.  If VECT = 'Q', M >= N >= min(M,K); if VECT = 'P', N >=
          M >= min(N,K).

K         (input) INTEGER
          If  VECT  =  'Q',  the number of columns in the original M-by-K
          matrix reduced by DGEBRD.  If VECT = 'P', the number of rows in
          the original K-by-N matrix reduced by DGEBRD.  K >= 0.

A         (input/output) DOUBLE PRECISION array, dimension (LDA,N)
          On  entry,  the vectors which define the elementary reflectors,
          as returned by DGEBRD.  On exit, the M-by-N matrix Q or P**T.

LDA       (input) INTEGER
          The leading dimension of the array A. LDA >= max(1,M).

TAU       (input) DOUBLE PRECISION array, dimension
          (min(M,K)) if VECT = 'Q' (min(N,K)) if VECT = 'P'  TAU(i)  must
          contain  the  scalar factor of the elementary reflector H(i) or
          G(i), which determines Q or P**T, as returned by DGEBRD in  its
          array argument TAUQ or TAUP.

WORK      (workspace/output)  DOUBLE  PRECISION  array,  dimension
(MAX(1,LWORK))
          On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK     (input) INTEGER
          The dimension of the array WORK. LWORK >= max(1,min(M,N)).  For
          optimum performance LWORK >= min(M,N)*NB, where NB is the opti-
          mal blocksize.

          If  LWORK  = -1, then a workspace query is assumed; the routine
          only calculates the optimal size of  the  WORK  array,  returns
          this  value  as the first entry of the WORK array, and no error
          message related to LWORK is issued by XERBLA.

INFO      (output) INTEGER
```

```
= 0:  successful exit
< 0:  if INFO = -i, the i-th argument had an illegal value
```

⟨*LAPACK dorgbr*⟩≡

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dorgbr (vect m n k a lda tau work lwork info)
    (declare (type (simple-array double-float (*)) work tau a)
             (type fixnum info lwork lda k n m)
             (type character vect))
    (f2cl-lib:with-multi-array-data
        ((vect character vect-%data% vect-%offset%)
         (a double-float a-%data% a-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((i 0) (iinfo 0) (j 0) (lwkopt 0) (mn 0) (nb 0) (lquery nil)
             (wantq nil))
        (declare (type fixnum i iinfo j lwkopt mn nb)
                 (type (member t nil) lquery wantq))
        (setf info 0)
        (setf wantq (char-equal vect #\Q))
        (setf mn (min (the fixnum m) (the fixnum n)))
        (setf lquery (coerce (= lwork -1) '(member t nil)))
        (cond
          ((and (not wantq) (not (char-equal vect #\P)))
           (setf info -1))
          ((< m 0)
           (setf info -2))
          ((or (< n 0)
               (and wantq
                    (or (> n m)
                        (< n
                           (min (the fixnum m)
                                (the fixnum k)))))
               (and (not wantq)
                    (or (> m n)
                        (< m
                           (min (the fixnum n)
                                (the fixnum k))))))
           (setf info -3))
          ((< k 0)
           (setf info -4))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info -6))
          ((and
            (< lwork
               (max (the fixnum 1) (the fixnum mn)))
            (not lquery))
```

```
           (setf info -9)))
         (cond
           ((= info 0)
            (cond
              (wantq
               (setf nb (ilaenv 1 "DORGQR" " " m n k -1)))
              (t
               (setf nb (ilaenv 1 "DORGLQ" " " m n k -1))))
            (setf lwkopt
                    (f2cl-lib:int-mul
                     (max (the fixnum 1) (the fixnum mn))
                     nb))
            (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                    (coerce (the fixnum lwkopt) 'double-float))))
p          (cond
            ((/= info 0)
             (error
              " ** On entry to ~a parameter number ~a had an illegal value~%"
              "DORGBR" (f2cl-lib:int-sub info))
             (go end_label))
            (lquery
             (go end_label)))
         (cond
           ((or (= m 0) (= n 0))
            (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                    (coerce (the fixnum 1) 'double-float))
            (go end_label)))
         (cond
           (wantq
            (cond
              ((>= m k)
               (multiple-value-bind
                     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
                   (dorgqr m n k a lda tau work lwork iinfo)
                 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                  var-7))
                 (setf iinfo var-8)))
              (t
               (f2cl-lib:fdo (j m (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                             ((> j 2) nil)
                 (tagbody
                   (setf (f2cl-lib:fref a-%data%
                                        (1 j)
                                        ((1 lda) (1 *))
                                        a-%offset%)
                           zero)
```

```
            (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                            (f2cl-lib:int-add i 1))
                          ((> i m) nil)
              (tagbody
                (setf (f2cl-lib:fref a-%data%
                                     (i j)
                                     ((1 lda) (1 *))
                                     a-%offset%)
                      (f2cl-lib:fref a-%data%
                                     (i (f2cl-lib:int-sub j 1))
                                     ((1 lda) (1 *))
                                     a-%offset%))))))
          (setf (f2cl-lib:fref a-%data% (1 1) ((1 lda) (1 *)) a-%offset%)
                  one)
          (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref a-%data%
                                   (i 1)
                                   ((1 lda) (1 *))
                                   a-%offset%)
                    zero)))
          (cond
            ((> m 1)
             (multiple-value-bind
                   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
                 (dorgqr (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1)
                  (f2cl-lib:int-sub m 1)
                  (f2cl-lib:array-slice a
                                        double-float
                                        (2 2)
                                        ((1 lda) (1 *)))
                  lda tau work lwork iinfo)
               (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                var-7))
               (setf iinfo var-8)))))))
      (t
       (cond
         ((< k n)
          (multiple-value-bind
                (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
              (dorglq m n k a lda tau work lwork iinfo)
            (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                             var-7))
            (setf iinfo var-8)))
         (t
```

```
          (setf (f2cl-lib:fref a-%data% (1 1) ((1 lda) (1 *)) a-%offset%)
                  one)
          (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                        ((> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref a-%data%
                                   (i 1)
                                   ((1 lda) (1 *))
                                   a-%offset%)
                    zero)))
          (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                        ((> j n) nil)
            (tagbody
              (f2cl-lib:fdo (i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))
                               (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                            ((> i 2) nil)
                (tagbody
                  (setf (f2cl-lib:fref a-%data%
                                       (i j)
                                       ((1 lda) (1 *))
                                       a-%offset%)
                        (f2cl-lib:fref a-%data%
                                       ((f2cl-lib:int-sub i 1) j)
                                       ((1 lda) (1 *))
                                       a-%offset%))))
              (setf (f2cl-lib:fref a-%data%
                                   (1 j)
                                   ((1 lda) (1 *))
                                   a-%offset%)
                    zero)))
          (cond
            ((> n 1)
             (multiple-value-bind
                   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
                 (dorglq (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1)
                  (f2cl-lib:int-sub n 1)
                  (f2cl-lib:array-slice a
                                        double-float
                                        (2 2)
                                        ((1 lda) (1 *)))
                  lda tau work lwork iinfo)
                (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                 var-7))
                (setf iinfo var-8))))))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
          (coerce (the fixnum lwkopt) 'double-float))
```

```
   end_label
          (return (values nil nil nil nil nil nil nil nil nil info))))))
```

7.81 dorghr LAPACK

⟨*dorghr.input*⟩≡

```
  )set break resume
  )sys rm -f dorghr.output
  )spool dorghr.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dorghr.help*⟩≡

```
=====================================================================
dorghr examples
=====================================================================

=====================================================================
Man Page Details
=====================================================================

NAME
       DORGHR  - a real orthogonal matrix Q which is defined as the product of
       IHI-ILO elementary reflectors of order N, as returned by DGEHRD

SYNOPSIS
       SUBROUTINE DORGHR( N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO )

           INTEGER        IHI, ILO, INFO, LDA, LWORK, N

           DOUBLE         PRECISION A( LDA, * ), TAU( * ), WORK( * )

PURPOSE
       DORGHR generates a real orthogonal matrix Q which  is  defined  as  the
       product  of  IHI-ILO  elementary  reflectors of order N, as returned by
       DGEHRD:

       Q = H(ilo) H(ilo+1) . . . H(ihi-1).


ARGUMENTS
       N       (input) INTEGER
               The order of the matrix Q. N >= 0.

       ILO     (input) INTEGER
               IHI     (input) INTEGER ILO and IHI must have the  same  values
               as  in  the  previous  call  of  DGEHRD. Q is equal to the unit
               matrix except in the submatrix  Q(ilo+1:ihi,ilo+1:ihi).   1  <=
               ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

       A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
               On  entry,  the vectors which define the elementary reflectors,
               as returned by DGEHRD.  On exit, the N-by-N  orthogonal  matrix
               Q.

       LDA     (input) INTEGER
               The leading dimension of the array A. LDA >= max(1,N).
```

TAU (input) DOUBLE PRECISION array, dimension (N-1)
TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGEHRD.

WORK (workspace/output) DOUBLE PRECISION array, dimension (MAX(1,LWORK))
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK >= IHI-ILO. For optimum performance LWORK >= (IHI-ILO)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value

⟨*LAPACK dorghr*⟩≡

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dorghr (n ilo ihi a lda tau work lwork info)
    (declare (type (simple-array double-float (*)) work tau a)
             (type fixnum info lwork lda ihi ilo n))
    (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((i 0) (iinfo 0) (j 0) (lwkopt 0) (nb 0) (nh 0) (lquery nil))
        (declare (type fixnum i iinfo j lwkopt nb nh)
                 (type (member t nil) lquery))
        (setf info 0)
        (setf nh (f2cl-lib:int-sub ihi ilo))
        (setf lquery (coerce (= lwork -1) '(member t nil)))
        (cond
          ((< n 0)
           (setf info -1))
          ((or (< ilo 1)
               (> ilo
                  (max (the fixnum 1) (the fixnum n))))
           (setf info -2))
          ((or
            (< ihi (min (the fixnum ilo) (the fixnum n)))
            (> ihi n))
           (setf info -3))
          ((< lda (max (the fixnum 1) (the fixnum n)))
           (setf info -5))
          ((and
            (< lwork
               (max (the fixnum 1) (the fixnum nh)))
            (not lquery))
           (setf info -8)))
        (cond
          ((= info 0)
           (setf nb (ilaenv 1 "DORGQR" " " nh nh nh -1))
           (setf lwkopt
                   (f2cl-lib:int-mul
                    (max (the fixnum 1) (the fixnum nh))
                    nb))
           (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                   (coerce (the fixnum lwkopt) 'double-float))))
        (cond
          ((/= info 0)
```

```
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DORGHR" (f2cl-lib:int-sub info))
   (go end_label))
  (lquery
   (go end_label)))
(cond
  ((= n 0)
   (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
           (coerce (the fixnum 1) 'double-float))
   (go end_label)))
(f2cl-lib:fdo (j ihi (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
              ((> j (f2cl-lib:int-add ilo 1)) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))) nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *)) a-%offset%)
                zero)))
    (f2cl-lib:fdo (i (f2cl-lib:int-add j 1) (f2cl-lib:int-add i 1))
                  ((> i ihi) nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *)) a-%offset%)
                (f2cl-lib:fref a-%data%
                               (i (f2cl-lib:int-sub j 1))
                               ((1 lda) (1 *))
                               a-%offset%))))
    (f2cl-lib:fdo (i (f2cl-lib:int-add ihi 1) (f2cl-lib:int-add i 1))
                  ((> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *)) a-%offset%)
                zero)))))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j ilo) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *)) a-%offset%)
                zero)))
    (setf (f2cl-lib:fref a-%data% (j j) ((1 lda) (1 *)) a-%offset%)
            one)))
(f2cl-lib:fdo (j (f2cl-lib:int-add ihi 1) (f2cl-lib:int-add j 1))
              ((> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
```

```
                        ((> i n) nil)
              (tagbody
                (setf (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *)) a-%offset%)
                      zero)))
            (setf (f2cl-lib:fref a-%data% (j j) ((1 lda) (1 *)) a-%offset%)
                  one)))
        (cond
          ((> nh 0)
           (multiple-value-bind
                 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
               (dorgqr nh nh nh
                (f2cl-lib:array-slice a
                                      double-float
                                      ((+ ilo 1) (f2cl-lib:int-add ilo 1))
                                      ((1 lda) (1 *)))
                lda (f2cl-lib:array-slice tau double-float (ilo) ((1 *))) work
                lwork iinfo)
             (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7))
             (setf iinfo var-8))))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                (coerce (the fixnum lwkopt) 'double-float))
  end_label
        (return (values nil nil nil nil nil nil nil nil info))))))
```

7.82 dorgl2 LAPACK

⟨*dorgl2.input*⟩≡

```
)set break resume
)sys rm -f dorgl2.output
)spool dorgl2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dorgl2.help*⟩≡

```
  ========================================================================
  dorgl2 examples
  ========================================================================

  ========================================================================
  Man Page Details
  ========================================================================

  NAME
         DORGL2 - an m by n real matrix Q with orthonormal rows,

  SYNOPSIS
         SUBROUTINE DORGL2( M, N, K, A, LDA, TAU, WORK, INFO )

             INTEGER        INFO, K, LDA, M, N

             DOUBLE         PRECISION A( LDA, * ), TAU( * ), WORK( * )

  PURPOSE
         DORGL2  generates  an m by n real matrix Q with orthonormal rows, which
         is defined as the first m rows of a product of k elementary  reflectors
         of order n

               Q  =  H(k) . . . H(2) H(1)

         as returned by DGELQF.


  ARGUMENTS
         M       (input) INTEGER
                 The number of rows of the matrix Q. M >= 0.

         N       (input) INTEGER
                 The number of columns of the matrix Q. N >= M.

         K       (input) INTEGER
                 The  number  of elementary reflectors whose product defines the
                 matrix Q. M >= K >= 0.

         A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
                 On entry, the i-th row must contain the  vector  which  defines
                 the  elementary  reflector H(i), for i = 1,2,...,k, as returned
                 by DGELQF in the first k rows of  its  array  argument  A.   On
                 exit, the m-by-n matrix Q.
```

```
LDA     (input) INTEGER
        The first dimension of the array A. LDA >= max(1,M).

TAU     (input) DOUBLE PRECISION array, dimension (K)
        TAU(i) must contain the scalar factor of the elementary reflec-
        tor H(i), as returned by DGELQF.

WORK    (workspace) DOUBLE PRECISION array, dimension (M)

INFO    (output) INTEGER
        = 0: successful exit
        < 0: if INFO = -i, the i-th argument has an illegal value
```

⟨*LAPACK dorgl2*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dorgl2 (m n k a lda tau work info)
    (declare (type (simple-array double-float (*)) work tau a)
             (type fixnum info lda k n m))
    (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((i 0) (j 0) (l 0))
        (declare (type fixnum i j l))
        (setf info 0)
        (cond
          ((< m 0)
           (setf info -1))
          ((< n m)
           (setf info -2))
          ((or (< k 0) (> k m))
           (setf info -3))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info -5)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DORGL2" (f2cl-lib:int-sub info))
           (go end_label)))
        (if (<= m 0) (go end_label))
        (cond
          ((< k m)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (f2cl-lib:fdo (l (f2cl-lib:int-add k 1) (f2cl-lib:int-add l 1))
                             ((> l m) nil)
                 (tagbody
                   (setf (f2cl-lib:fref a-%data%
                                        (l j)
                                        ((1 lda) (1 *))
                                        a-%offset%)
                           zero)))
               (if (and (> j k) (<= j m))
                   (setf (f2cl-lib:fref a-%data%
                                        (j j)
```

```
                                        ((1 lda) (1 *))
                                        a-%offset%)
                              one))))))
          (f2cl-lib:fdo (i k (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                        ((> i 1) nil)
            (tagbody
              (cond
                ((< i n)
                 (cond
                   ((< i m)
                    (setf (f2cl-lib:fref a-%data%
                                         (i i)
                                         ((1 lda) (1 *))
                                         a-%offset%)
                          one)
                    (dlarf "Right" (f2cl-lib:int-sub m i)
                     (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
                     (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
                     lda (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
                     (f2cl-lib:array-slice a
                                           double-float
                                           ((+ i 1) i)
                                           ((1 lda) (1 *)))
                     lda work)))
                 (dscal (f2cl-lib:int-sub n i)
                  (- (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
                  (f2cl-lib:array-slice a
                                        double-float
                                        (i (f2cl-lib:int-add i 1))
                                        ((1 lda) (1 *)))
                  lda)))
              (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                    (- one
                       (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)))
              (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                            ((> l (f2cl-lib:int-add i (f2cl-lib:int-sub 1))) nil)
                (tagbody
                  (setf (f2cl-lib:fref a-%data% (i l) ((1 lda) (1 *)) a-%offset%)
                        zero)))))
        end_label
          (return (values nil nil nil nil nil nil nil info))))))
```

7.83 dorglq LAPACK

⟨*dorglq.input*⟩≡

```
)set break resume
)sys rm -f dorglq.output
)spool dorglq.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dorglq.help*⟩≡

```
======================================================================
dorglq examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DORGLQ - an M-by-N real matrix Q with orthonormal rows,

SYNOPSIS
       SUBROUTINE DORGLQ( M, N, K, A, LDA, TAU, WORK, LWORK, INFO )

           INTEGER        INFO, K, LDA, LWORK, M, N

           DOUBLE         PRECISION A( LDA, * ), TAU( * ), WORK( * )

PURPOSE
       DORGLQ  generates  an M-by-N real matrix Q with orthonormal rows, which
       is defined as the first M rows of a product of K elementary  reflectors
       of order N

             Q  =  H(k) . . . H(2) H(1)

       as returned by DGELQF.


ARGUMENTS
       M       (input) INTEGER
               The number of rows of the matrix Q. M >= 0.

       N       (input) INTEGER
               The number of columns of the matrix Q. N >= M.

       K       (input) INTEGER
               The  number  of elementary reflectors whose product defines the
               matrix Q. M >= K >= 0.

       A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
               On entry, the i-th row must contain the  vector  which  defines
               the  elementary  reflector H(i), for i = 1,2,...,k, as returned
               by DGELQF in the first k rows of  its  array  argument  A.   On
               exit, the M-by-N matrix Q.
```

```
LDA     (input) INTEGER
        The first dimension of the array A. LDA >= max(1,M).

TAU     (input) DOUBLE PRECISION array, dimension (K)
        TAU(i) must contain the scalar factor of the elementary reflec-
        tor H(i), as returned by DGELQF.

WORK      (workspace/output)  DOUBLE  PRECISION  array,  dimension
(MAX(1,LWORK))
        On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK   (input) INTEGER
        The dimension of the array WORK. LWORK >= max(1,M).  For  opti-
        mum  performance  LWORK >= M*NB, where NB is the optimal block-
        size.

        If LWORK = -1, then a workspace query is assumed;  the  routine
        only  calculates  the  optimal  size of the WORK array, returns
        this value as the first entry of the WORK array, and  no  error
        message related to LWORK is issued by XERBLA.

INFO    (output) INTEGER
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument has an illegal value
```

⟨*LAPACK dorglq*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dorglq (m n k a lda tau work lwork info)
    (declare (type (simple-array double-float (*)) work tau a)
             (type fixnum info lwork lda k n m))
    (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((i 0) (ib 0) (iinfo 0) (iws 0) (j 0) (ki 0) (kk 0) (l 0)
             (ldwork 0) (lwkopt 0) (nb 0) (nbmin 0) (nx 0) (lquery nil))
        (declare (type fixnum i ib iinfo iws j ki kk l ldwork
                                          lwkopt nb nbmin nx)
                 (type (member t nil) lquery))
        (setf info 0)
        (setf nb (ilaenv 1 "DORGLQ" " " m n k -1))
        (setf lwkopt
                (f2cl-lib:int-mul
                 (max (the fixnum 1) (the fixnum m))
                 nb))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                (coerce (the fixnum lwkopt) 'double-float))
        (setf lquery (coerce (= lwork -1) '(member t nil)))
        (cond
          ((< m 0)
           (setf info -1))
          ((< n m)
           (setf info -2))
          ((or (< k 0) (> k m))
           (setf info -3))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info -5))
          ((and
            (< lwork (max (the fixnum 1) (the fixnum m)))
            (not lquery))
           (setf info -8)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DORGLQ" (f2cl-lib:int-sub info))
           (go end_label))
          (lquery
           (go end_label)))
        (cond
```

```
  ((<= m 0)
   (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
           (coerce (the fixnum 1) 'double-float))
   (go end_label)))
(setf nbmin 2)
(setf nx 0)
(setf iws m)
(cond
  ((and (> nb 1) (< nb k))
   (setf nx
           (max (the fixnum 0)
                (the fixnum
                     (ilaenv 3 "DORGLQ" " " m n k -1))))
   (cond
     ((< nx k)
      (setf ldwork m)
      (setf iws (f2cl-lib:int-mul ldwork nb))
      (cond
        ((< lwork iws)
         (setf nb (the fixnum (truncate lwork ldwork)))
         (setf nbmin
                 (max (the fixnum 2)
                      (the fixnum
                           (ilaenv 2 "DORGLQ" " " m n k -1)))))))))
(cond
  ((and (>= nb nbmin) (< nb k) (< nx k))
   (setf ki (* (the fixnum (truncate (- k nx 1) nb)) nb))
   (setf kk
           (min (the fixnum k)
                (the fixnum (f2cl-lib:int-add ki nb))))
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                 ((> j kk) nil)
     (tagbody
       (f2cl-lib:fdo (i (f2cl-lib:int-add kk 1) (f2cl-lib:int-add i 1))
                     ((> i m) nil)
         (tagbody
           (setf (f2cl-lib:fref a-%data%
                                (i j)
                                ((1 lda) (1 *))
                                a-%offset%)
                   zero))))))
  (t
   (setf kk 0)))
(if (< kk m)
    (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
```

```
          (dorgl2 (f2cl-lib:int-sub m kk) (f2cl-lib:int-sub n kk)
           (f2cl-lib:int-sub k kk)
           (f2cl-lib:array-slice a
                                 double-float
                                 ((+ kk 1) (f2cl-lib:int-add kk 1))
                                 ((1 lda) (1 *)))
           lda (f2cl-lib:array-slice tau double-float ((+ kk 1)) ((1 *)))
           work iinfo)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
          (setf iinfo var-7)))
    (cond
      ((> kk 0)
       (f2cl-lib:fdo (i (f2cl-lib:int-add ki 1)
                        (f2cl-lib:int-add i (f2cl-lib:int-sub nb)))
                     ((> i 1) nil)
         (tagbody
           (setf ib
                   (min (the fixnum nb)
                        (the fixnum
                             (f2cl-lib:int-add (f2cl-lib:int-sub k i) 1))))
           (cond
             ((<= (f2cl-lib:int-add i ib) m)
              (dlarft "Forward" "Rowwise"
               (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) ib
               (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
               lda (f2cl-lib:array-slice tau double-float (i) ((1 *))) work
               ldwork)
              (dlarfb "Right" "Transpose" "Forward" "Rowwise"
               (f2cl-lib:int-add (f2cl-lib:int-sub m i ib) 1)
               (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) ib
               (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
               lda work ldwork
               (f2cl-lib:array-slice a
                                     double-float
                                     ((+ i ib) i)
                                     ((1 lda) (1 *)))
               lda
               (f2cl-lib:array-slice work double-float ((+ ib 1)) ((1 *)))
               ldwork)))
           (multiple-value-bind
                 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
               (dorgl2 ib (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) ib
                (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
                lda (f2cl-lib:array-slice tau double-float (i) ((1 *)))
                work iinfo)
             (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
```

```
                (setf iinfo var-7))
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                            ((> j (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                             nil)
                (tagbody
                  (f2cl-lib:fdo (l i (f2cl-lib:int-add l 1))
                                ((> l
                                    (f2cl-lib:int-add i
                                                      ib
                                                      (f2cl-lib:int-sub 1)))
                                 nil)
                    (tagbody
                      (setf (f2cl-lib:fref a-%data%
                                           (l j)
                                           ((1 lda) (1 *))
                                           a-%offset%)
                            zero)))))))))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                (coerce (the fixnum iws) 'double-float))
  end_label
        (return (values nil nil nil nil nil nil nil nil info))))))
```

7.84 dorgqr LAPACK

⟨*dorgqr.input*⟩≡

```
)set break resume
)sys rm -f dorgqr.output
)spool dorgqr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dorgqr.help*⟩≡

```
=====================================================================
dorgqr examples
=====================================================================

=====================================================================
Man Page Details
=====================================================================

NAME
       DORGQR - an M-by-N real matrix Q with orthonormal columns,

SYNOPSIS
       SUBROUTINE DORGQR( M, N, K, A, LDA, TAU, WORK, LWORK, INFO )

           INTEGER        INFO, K, LDA, LWORK, M, N

           DOUBLE         PRECISION A( LDA, * ), TAU( * ), WORK( * )

PURPOSE
       DORGQR  generates  an  M-by-N  real  matrix Q with orthonormal columns,
       which is defined as the first N columns of a product  of  K  elementary
       reflectors of order M

             Q  =  H(1) H(2) . . . H(k)

       as returned by DGEQRF.


ARGUMENTS
       M       (input) INTEGER
               The number of rows of the matrix Q. M >= 0.

       N       (input) INTEGER
               The number of columns of the matrix Q. M >= N >= 0.

       K       (input) INTEGER
               The  number  of elementary reflectors whose product defines the
               matrix Q. N >= K >= 0.

       A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
               On entry, the i-th column must contain the vector which defines
               the  elementary  reflector H(i), for i = 1,2,...,k, as returned
               by DGEQRF in the first k columns of its array argument  A.   On
               exit, the M-by-N matrix Q.
```

```
LDA     (input) INTEGER
        The first dimension of the array A. LDA >= max(1,M).

TAU     (input) DOUBLE PRECISION array, dimension (K)
        TAU(i) must contain the scalar factor of the elementary reflec-
        tor H(i), as returned by DGEQRF.

WORK      (workspace/output)  DOUBLE  PRECISION  array,  dimension
(MAX(1,LWORK))
        On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK   (input) INTEGER
        The dimension of the array WORK. LWORK >= max(1,N).  For  opti-
        mum  performance  LWORK >= N*NB, where NB is the optimal block-
        size.

        If LWORK = -1, then a workspace query is assumed;  the  routine
        only  calculates  the  optimal  size of the WORK array, returns
        this value as the first entry of the WORK array, and  no  error
        message related to LWORK is issued by XERBLA.

INFO    (output) INTEGER
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument has an illegal value
```

⟨*LAPACK dorgqr*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dorgqr (m n k a lda tau work lwork info)
    (declare (type (simple-array double-float (*)) work tau a)
             (type fixnum info lwork lda k n m))
    (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((i 0) (ib 0) (iinfo 0) (iws 0) (j 0) (ki 0) (kk 0) (l 0)
             (ldwork 0) (lwkopt 0) (nb 0) (nbmin 0) (nx 0) (lquery nil))
        (declare (type fixnum i ib iinfo iws j ki kk l ldwork
                                          lwkopt nb nbmin nx)
                 (type (member t nil) lquery))
        (setf info 0)
        (setf nb (ilaenv 1 "DORGQR" " " m n k -1))
        (setf lwkopt
                (f2cl-lib:int-mul
                 (max (the fixnum 1) (the fixnum n))
                 nb))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                (coerce (the fixnum lwkopt) 'double-float))
        (setf lquery (coerce (= lwork -1) '(member t nil)))
        (cond
          ((< m 0)
           (setf info -1))
          ((or (< n 0) (> n m))
           (setf info -2))
          ((or (< k 0) (> k n))
           (setf info -3))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info -5))
          ((and
            (< lwork (max (the fixnum 1) (the fixnum n)))
            (not lquery))
           (setf info -8)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DORGQR" (f2cl-lib:int-sub info))
           (go end_label))
          (lquery
           (go end_label)))
        (cond
```

```
  ((<= n 0)
   (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
           (coerce (the fixnum 1) 'double-float))
   (go end_label)))
(setf nbmin 2)
(setf nx 0)
(setf iws n)
(cond
  ((and (> nb 1) (< nb k))
   (setf nx
           (max (the fixnum 0)
                (the fixnum
                     (ilaenv 3 "DORGQR" " " m n k -1))))
   (cond
     ((< nx k)
      (setf ldwork n)
      (setf iws (f2cl-lib:int-mul ldwork nb))
      (cond
        ((< lwork iws)
         (setf nb (the fixnum (truncate lwork ldwork)))
         (setf nbmin
                 (max (the fixnum 2)
                      (the fixnum
                           (ilaenv 2 "DORGQR" " " m n k -1)))))))))
(cond
  ((and (>= nb nbmin) (< nb k) (< nx k))
   (setf ki (* (the fixnum (truncate (- k nx 1) nb)) nb))
   (setf kk
           (min (the fixnum k)
                (the fixnum (f2cl-lib:int-add ki nb))))
   (f2cl-lib:fdo (j (f2cl-lib:int-add kk 1) (f2cl-lib:int-add j 1))
                 ((> j n) nil)
     (tagbody
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i kk) nil)
         (tagbody
           (setf (f2cl-lib:fref a-%data%
                                (i j)
                                ((1 lda) (1 *))
                                a-%offset%)
                   zero))))))
  (t
   (setf kk 0)))
(if (< kk n)
    (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
```

```
          (dorg2r (f2cl-lib:int-sub m kk) (f2cl-lib:int-sub n kk)
           (f2cl-lib:int-sub k kk)
           (f2cl-lib:array-slice a
                                 double-float
                                 ((+ kk 1) (f2cl-lib:int-add kk 1))
                                 ((1 lda) (1 *)))
           lda (f2cl-lib:array-slice tau double-float ((+ kk 1)) ((1 *)))
           work iinfo)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
          (setf iinfo var-7)))
    (cond
      ((> kk 0)
       (f2cl-lib:fdo (i (f2cl-lib:int-add ki 1)
                        (f2cl-lib:int-add i (f2cl-lib:int-sub nb)))
                     ((> i 1) nil)
         (tagbody
           (setf ib
                   (min (the fixnum nb)
                        (the fixnum
                             (f2cl-lib:int-add (f2cl-lib:int-sub k i) 1))))
           (cond
             ((<= (f2cl-lib:int-add i ib) n)
              (dlarft "Forward" "Columnwise"
               (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1) ib
               (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
               lda (f2cl-lib:array-slice tau double-float (i) ((1 *))) work
               ldwork)
              (dlarfb "Left" "No transpose" "Forward" "Columnwise"
               (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
               (f2cl-lib:int-add (f2cl-lib:int-sub n i ib) 1) ib
               (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
               lda work ldwork
               (f2cl-lib:array-slice a
                                     double-float
                                     (i (f2cl-lib:int-add i ib))
                                     ((1 lda) (1 *)))
               lda
               (f2cl-lib:array-slice work double-float ((+ ib 1)) ((1 *)))
               ldwork)))
           (multiple-value-bind
                 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
               (dorg2r (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1) ib ib
                (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
                lda (f2cl-lib:array-slice tau double-float (i) ((1 *)))
                work iinfo)
             (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
```

```
                  (setf iinfo var-7))
                (f2cl-lib:fdo (j i (f2cl-lib:int-add j 1))
                              ((> j
                                  (f2cl-lib:int-add i ib (f2cl-lib:int-sub 1)))
                               nil)
                  (tagbody
                    (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                                  ((> l
                                      (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                                   nil)
                      (tagbody
                        (setf (f2cl-lib:fref a-%data%
                                             (l j)
                                             ((1 lda) (1 *))
                                             a-%offset%)
                              zero)))))))))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                (coerce (the fixnum iws) 'double-float))
  end_label
        (return (values nil nil nil nil nil nil nil nil info))))))
```

7.85 dorm2r LAPACK

⟨*dorm2r.input*⟩≡

```
  )set break resume
  )sys rm -f dorm2r.output
  )spool dorm2r.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dorm2r.help*⟩≡

```
======================================================================
dorm2r examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DORM2R  -  the  general real m by n matrix C with   Q * C if SIDE = 'L'
       and TRANS = 'N', or   Q'* C if SIDE = 'L' and TRANS = 'T', or   C  *  Q
       if  SIDE  =  'R' and TRANS = 'N', or   C * Q' if SIDE = 'R' and TRANS =
       'T',

SYNOPSIS
       SUBROUTINE DORM2R( SIDE, TRANS, M, N, K, A, LDA,  TAU,  C,  LDC,  WORK,
                          INFO )

           CHARACTER      SIDE, TRANS

           INTEGER        INFO, K, LDA, LDC, M, N

           DOUBLE         PRECISION  A( LDA, * ), C( LDC, * ), TAU( * ), WORK(
                          * )

PURPOSE
       DORM2R overwrites the general real m by n matrix C with

       where Q is a real orthogonal matrix defined as the product of k elemen-
       tary reflectors

             Q = H(1) H(2) . . . H(k)

       as  returned by DGEQRF. Q is of order m if SIDE = 'L' and of order n if
       SIDE = 'R'.


ARGUMENTS
       SIDE    (input) CHARACTER*1
               = 'L': apply Q or Q' from the Left
               = 'R': apply Q or Q' from the Right

       TRANS   (input) CHARACTER*1
               = 'N': apply Q  (No transpose)
               = 'T': apply Q' (Transpose)
```

```
M       (input) INTEGER
        The number of rows of the matrix C. M >= 0.

N       (input) INTEGER
        The number of columns of the matrix C. N >= 0.

K       (input) INTEGER
        The number of elementary reflectors whose product  defines  the
        matrix Q.  If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >=
        0.

A       (input) DOUBLE PRECISION array, dimension (LDA,K)
        The i-th column must contain the vector which defines the  ele-
        mentary  reflector H(i), for i = 1,2,...,k, as returned by DGE-
        QRF in the first k columns of its array argument A.  A is modi-
        fied by the routine but restored on exit.

LDA     (input) INTEGER
        The  leading  dimension  of the array A.  If SIDE = 'L', LDA >=
        max(1,M); if SIDE = 'R', LDA >= max(1,N).

TAU     (input) DOUBLE PRECISION array, dimension (K)
        TAU(i) must contain the scalar factor of the elementary reflec-
        tor H(i), as returned by DGEQRF.

C       (input/output) DOUBLE PRECISION array, dimension (LDC,N)
        On  entry,  the  m by n matrix C.  On exit, C is overwritten by
        Q*C or Q'*C or C*Q' or C*Q.

LDC     (input) INTEGER
        The leading dimension of the array C. LDC >= max(1,M).

WORK    (workspace) DOUBLE PRECISION array, dimension
        (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO    (output) INTEGER
        = 0: successful exit
        < 0: if INFO = -i, the i-th argument had an illegal value
```

⟨*LAPACK dorm2r*⟩≡

```
(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dorm2r (side trans m n k a lda tau c ldc work info)
    (declare (type (simple-array double-float (*)) work c tau a)
             (type fixnum info ldc lda k n m)
             (type character trans side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (trans character trans-%data% trans-%offset%)
         (a double-float a-%data% a-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (c double-float c-%data% c-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((aii 0.0) (i 0) (i1 0) (i2 0) (i3 0) (ic 0) (jc 0) (mi 0) (ni 0)
             (nq 0) (left nil) (notran nil))
        (declare (type (double-float) aii)
                 (type fixnum i i1 i2 i3 ic jc mi ni nq)
                 (type (member t nil) left notran))
        (setf info 0)
        (setf left (char-equal side #\L))
        (setf notran (char-equal trans #\N))
        (cond
          (left
           (setf nq m))
          (t
           (setf nq n)))
        (cond
          ((and (not left) (not (char-equal side #\R)))
           (setf info -1))
          ((and (not notran) (not (char-equal trans #\T)))
           (setf info -2))
          ((< m 0)
           (setf info -3))
          ((< n 0)
           (setf info -4))
          ((or (< k 0) (> k nq))
           (setf info -5))
          ((< lda (max (the fixnum 1) (the fixnum nq)))
           (setf info -7))
          ((< ldc (max (the fixnum 1) (the fixnum m)))
           (setf info -10)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
```

```
             "DORM2R" (f2cl-lib:int-sub info))
            (go end_label)))
         (if (or (= m 0) (= n 0) (= k 0)) (go end_label))
         (cond
           ((or (and left (not notran)) (and (not left) notran))
            (setf i1 1)
            (setf i2 k)
            (setf i3 1))
           (t
            (setf i1 k)
            (setf i2 1)
            (setf i3 -1)))
         (cond
           (left
            (setf ni n)
            (setf jc 1))
           (t
            (setf mi m)
            (setf ic 1)))
         (f2cl-lib:fdo (i i1 (f2cl-lib:int-add i i3))
                       ((> i i2) nil)
           (tagbody
             (cond
               (left
                (setf mi (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1))
                (setf ic i))
               (t
                (setf ni (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1))
                (setf jc i)))
             (setf aii
                     (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%))
             (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                     one)
             (dlarf side mi ni
              (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) 1
              (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
              (f2cl-lib:array-slice c double-float (ic jc) ((1 ldc) (1 *))) ldc
              work)
             (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                     aii)))
       end_label
         (return (values nil nil nil nil nil nil nil nil nil nil nil info))))))
```

7.86 dormbr LAPACK

⟨*dormbr.input*⟩≡

```
)set break resume
)sys rm -f dormbr.output
)spool dormbr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dormbr.help*⟩≡

```
  ======================================================================
  dormbr examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DORMBR - = 'Q', DORMBR overwrites the general real M-by-N matrix C with
         SIDE = 'L' SIDE = 'R' TRANS = 'N'

  SYNOPSIS
         SUBROUTINE DORMBR( VECT, SIDE, TRANS, M, N, K, A,  LDA,  TAU,  C,  LDC,
                            WORK, LWORK, INFO )

             CHARACTER      SIDE, TRANS, VECT

             INTEGER        INFO, K, LDA, LDC, LWORK, M, N

             DOUBLE         PRECISION  A( LDA, * ), C( LDC, * ), TAU( * ), WORK(
                            * )

  PURPOSE
         If VECT = 'Q', DORMBR overwrites the general real M-by-N matrix C with
                         SIDE = 'L'     SIDE = 'R'  TRANS  =  'N':       Q  *  C
         C * Q TRANS = 'T':      Q**T * C       C * Q**T

         If VECT = 'P', DORMBR overwrites the general real M-by-N matrix C with
                        SIDE = 'L'     SIDE = 'R'
         TRANS = 'N':      P * C          C * P
         TRANS = 'T':      P**T * C       C * P**T

         Here  Q  and P**T are the orthogonal matrices determined by DGEBRD when
         reducing a real matrix A to bidiagonal form: A = Q * B *  P**T.  Q  and
         P**T  are  defined  as  products of elementary reflectors H(i) and G(i)
         respectively.

         Let nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Thus nq is the order
         of the orthogonal matrix Q or P**T that is applied.

         If VECT = 'Q', A is assumed to have been an NQ-by-K matrix: if nq >= k,
         Q = H(1) H(2) . . . H(k);
         if nq < k, Q = H(1) H(2) . . . H(nq-1).
```

```
       If VECT = 'P', A is assumed to have been a K-by-NQ matrix: if k < nq, P
       = G(1) G(2) . . . G(k);
       if k >= nq, P = G(1) G(2) . . . G(nq-1).


 ARGUMENTS
       VECT    (input) CHARACTER*1
               = 'Q': apply Q or Q**T;
               = 'P': apply P or P**T.

       SIDE    (input) CHARACTER*1
               = 'L': apply Q, Q**T, P or P**T from the Left;
               = 'R': apply Q, Q**T, P or P**T from the Right.

       TRANS   (input) CHARACTER*1
               = 'N':  No transpose, apply Q  or P;
               = 'T':  Transpose, apply Q**T or P**T.

       M       (input) INTEGER
               The number of rows of the matrix C. M >= 0.

       N       (input) INTEGER
               The number of columns of the matrix C. N >= 0.

       K       (input) INTEGER
               If  VECT  =  'Q',  the number of columns in the original matrix
               reduced by DGEBRD.  If VECT = 'P', the number of  rows  in  the
               original matrix reduced by DGEBRD.  K >= 0.

       A       (input) DOUBLE PRECISION array, dimension
               (LDA,min(nq,K)) if VECT = 'Q' (LDA,nq)        if VECT = 'P' The
               vectors which define the elementary reflectors H(i)  and  G(i),
               whose  products  determine the matrices Q and P, as returned by
               DGEBRD.

       LDA     (input) INTEGER
               The leading dimension of the array A.  If VECT =  'Q',  LDA  >=
               max(1,nq); if VECT = 'P', LDA >= max(1,min(nq,K)).

       TAU     (input) DOUBLE PRECISION array, dimension (min(nq,K))
               TAU(i) must contain the scalar factor of the elementary reflec-
               tor H(i) or G(i) which determines Q or P, as returned by DGEBRD
               in the array argument TAUQ or TAUP.

       C       (input/output) DOUBLE PRECISION array, dimension (LDC,N)
               On  entry,  the  M-by-N matrix C.  On exit, C is overwritten by
```

Q*C or Q**T*C or C*Q**T or C*Q or P*C or P**T*C or C*P or C*P**T.

LDC (input) INTEGER
The leading dimension of the array C. LDC >= max(1,M).

WORK (workspace/output) DOUBLE PRECISION array, dimension (MAX(1,LWORK))
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1,N); if SIDE = 'R', LWORK >= max(1,M). For optimum performance LWORK >= N*NB if SIDE = 'L', and LWORK >= M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value

⟨*LAPACK dormbr*⟩≡

```
(defun dormbr (vect side trans m n k a lda tau c ldc work lwork info)
  (declare (type (simple-array double-float (*)) work c tau a)
           (type fixnum info lwork ldc lda k n m)
           (type character trans side vect))
  (f2cl-lib:with-multi-array-data
      ((vect character vect-%data% vect-%offset%)
       (side character side-%data% side-%offset%)
       (trans character trans-%data% trans-%offset%)
       (a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (c double-float c-%data% c-%offset%)
       (work double-float work-%data% work-%offset%))
    (prog ((i1 0) (i2 0) (iinfo 0) (lwkopt 0) (mi 0) (nb 0) (ni 0) (nq 0)
           (nw 0)
           (transt
            (make-array '(1) :element-type 'character :initial-element #\ ))
           (applyq nil) (left nil) (lquery nil) (notran nil))
      (declare (type (member t nil) notran lquery left applyq)
               (type (simple-array character (1)) transt)
               (type fixnum nw nq ni nb mi lwkopt iinfo i2 i1))
      (setf info 0)
      (setf applyq (char-equal vect #\Q))
      (setf left (char-equal side #\L))
      (setf notran (char-equal trans #\N))
      (setf lquery (coerce (= lwork -1) '(member t nil)))
      (cond
        (left
         (setf nq m)
         (setf nw n))
        (t
         (setf nq n)
         (setf nw m)))
      (cond
        ((and (not applyq) (not (char-equal vect #\P)))
         (setf info -1))
        ((and (not left) (not (char-equal side #\R)))
         (setf info -2))
        ((and (not notran) (not (char-equal trans #\T)))
         (setf info -3))
        ((< m 0)
         (setf info -4))
        ((< n 0)
         (setf info -5))
        ((< k 0)
         (setf info -6))
```

```
          ((or
            (and applyq
                 (< lda
                    (max (the fixnum 1) (the fixnum nq))))
            (and (not applyq)
                 (< lda
                    (max (the fixnum 1)
                         (the fixnum
                              (min (the fixnum nq)
                                   (the fixnum k)))))))
           (setf info -8))
          ((< ldc (max (the fixnum 1) (the fixnum m)))
           (setf info -11))
          ((and
            (< lwork (max (the fixnum 1) (the fixnum nw)))
            (not lquery))
           (setf info -13)))
        (cond
          ((= info 0)
           (cond
             (applyq
              (cond
                (left
                 (setf nb
                         (ilaenv 1 "DORMQR" (f2cl-lib:f2cl-// side trans)
                          (f2cl-lib:int-sub m 1) n (f2cl-lib:int-sub m 1) -1)))
                (t
                 (setf nb
                         (ilaenv 1 "DORMQR" (f2cl-lib:f2cl-// side trans) m
                          (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1) -1)))))
             (t
              (cond
                (left
                 (setf nb
                         (ilaenv 1 "DORMLQ" (f2cl-lib:f2cl-// side trans)
                          (f2cl-lib:int-sub m 1) n (f2cl-lib:int-sub m 1) -1)))
                (t
                 (setf nb
                         (ilaenv 1 "DORMLQ" (f2cl-lib:f2cl-// side trans) m
                          (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1) -1))))))
           (setf lwkopt
                   (f2cl-lib:int-mul
                    (max (the fixnum 1) (the fixnum nw))
                    nb))
           (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                   (coerce (the fixnum lwkopt) 'double-float))))
```

```
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DORMBR" (f2cl-lib:int-sub info))
   (go end_label))
  (lquery
   (go end_label)))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
        (coerce (the fixnum 1) 'double-float))
(if (or (= m 0) (= n 0)) (go end_label))
(cond
  (applyq
   (cond
     ((>= nq k)
      (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
             var-10 var-11 var-12)
          (dormqr side trans m n k a lda tau c ldc work lwork iinfo)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                         var-8 var-9 var-10 var-11))
        (setf iinfo var-12)))
     ((> nq 1)
      (cond
        (left
         (setf mi (f2cl-lib:int-sub m 1))
         (setf ni n)
         (setf i1 2)
         (setf i2 1))
        (t
         (setf mi m)
         (setf ni (f2cl-lib:int-sub n 1))
         (setf i1 1)
         (setf i2 2)))
      (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
             var-10 var-11 var-12)
          (dormqr side trans mi ni (f2cl-lib:int-sub nq 1)
           (f2cl-lib:array-slice a double-float (2 1) ((1 lda) (1 *)))
           lda tau
           (f2cl-lib:array-slice c double-float (i1 i2) ((1 ldc) (1 *)))
           ldc work lwork iinfo)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                         var-8 var-9 var-10 var-11))
        (setf iinfo var-12)))))
  (t
```

```
          (cond
            (notran
             (f2cl-lib:f2cl-set-string transt "T" (string 1)))
            (t
             (f2cl-lib:f2cl-set-string transt "N" (string 1))))
          (cond
            ((> nq k)
             (multiple-value-bind
                   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
                    var-10 var-11 var-12)
                 (dormlq side transt m n k a lda tau c ldc work lwork iinfo)
               (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                                var-8 var-9 var-10 var-11))
               (setf iinfo var-12)))
            ((> nq 1)
             (cond
               (left
                (setf mi (f2cl-lib:int-sub m 1))
                (setf ni n)
                (setf i1 2)
                (setf i2 1))
               (t
                (setf mi m)
                (setf ni (f2cl-lib:int-sub n 1))
                (setf i1 1)
                (setf i2 2)))
             (multiple-value-bind
                   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
                    var-10 var-11 var-12)
                 (dormlq side transt mi ni (f2cl-lib:int-sub nq 1)
                  (f2cl-lib:array-slice a double-float (1 2) ((1 lda) (1 *)))
                  lda tau
                  (f2cl-lib:array-slice c double-float (i1 i2) ((1 ldc) (1 *)))
                  ldc work lwork iinfo)
               (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                                var-8 var-9 var-10 var-11))
               (setf iinfo var-12))))))
      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
            (coerce (the fixnum lwkopt) 'double-float))
  end_label
      (return
       (values nil nil nil nil nil nil nil nil nil nil nil nil nil info)))))
```

7.87 dorml2 LAPACK

⟨*dorml2.input*⟩≡

```
)set break resume
)sys rm -f dorml2.output
)spool dorml2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dorml2.help*⟩≡

```
  ======================================================================
  dorml2 examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         DORML2  -  the  general real m by n matrix C with   Q * C if SIDE = 'L'
         and TRANS = 'N', or   Q'* C if SIDE = 'L' and TRANS = 'T', or   C  *  Q
         if  SIDE  =  'R' and TRANS = 'N', or   C * Q' if SIDE = 'R' and TRANS =
         'T',

  SYNOPSIS
         SUBROUTINE DORML2( SIDE, TRANS, M, N, K, A, LDA,  TAU,  C,  LDC,  WORK,
                            INFO )

             CHARACTER      SIDE, TRANS

             INTEGER        INFO, K, LDA, LDC, M, N

             DOUBLE         PRECISION  A( LDA, * ), C( LDC, * ), TAU( * ), WORK(
                            * )

  PURPOSE
         DORML2 overwrites the general real m by n matrix C with

         where Q is a real orthogonal matrix defined as the product of k elemen-
         tary reflectors

               Q = H(k) . . . H(2) H(1)

         as  returned by DGELQF. Q is of order m if SIDE = 'L' and of order n if
         SIDE = 'R'.


  ARGUMENTS
         SIDE    (input) CHARACTER*1
                 = 'L': apply Q or Q' from the Left
                 = 'R': apply Q or Q' from the Right

         TRANS   (input) CHARACTER*1
                 = 'N': apply Q  (No transpose)
                 = 'T': apply Q' (Transpose)
```

```
M       (input) INTEGER
        The number of rows of the matrix C. M >= 0.

N       (input) INTEGER
        The number of columns of the matrix C. N >= 0.

K       (input) INTEGER
        The number of elementary reflectors whose product  defines  the
        matrix Q.  If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >=
        0.

A       (input) DOUBLE PRECISION array, dimension
        (LDA,M) if SIDE = 'L', (LDA,N) if SIDE = 'R' The i-th row  must
        contain the vector which defines the elementary reflector H(i),
        for i = 1,2,...,k, as returned by DGELQF in the first k rows of
        its  array  argument  A.   A  is  modified  by  the routine but
        restored on exit.

LDA     (input) INTEGER
        The leading dimension of the array A. LDA >= max(1,K).

TAU     (input) DOUBLE PRECISION array, dimension (K)
        TAU(i) must contain the scalar factor of the elementary reflec-
        tor H(i), as returned by DGELQF.

C       (input/output) DOUBLE PRECISION array, dimension (LDC,N)
        On  entry,  the  m by n matrix C.  On exit, C is overwritten by
        Q*C or Q'*C or C*Q' or C*Q.

LDC     (input) INTEGER
        The leading dimension of the array C. LDC >= max(1,M).

WORK    (workspace) DOUBLE PRECISION array, dimension
        (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO    (output) INTEGER
        = 0: successful exit
        < 0: if INFO = -i, the i-th argument had an illegal value
```

⟨*LAPACK dorml2*⟩≡

```
(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dorml2 (side trans m n k a lda tau c ldc work info)
    (declare (type (simple-array double-float (*)) work c tau a)
             (type fixnum info ldc lda k n m)
             (type character trans side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (trans character trans-%data% trans-%offset%)
         (a double-float a-%data% a-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (c double-float c-%data% c-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((aii 0.0) (i 0) (i1 0) (i2 0) (i3 0) (ic 0) (jc 0) (mi 0) (ni 0)
             (nq 0) (left nil) (notran nil))
        (declare (type (double-float) aii)
                 (type fixnum i i1 i2 i3 ic jc mi ni nq)
                 (type (member t nil) left notran))
        (setf info 0)
        (setf left (char-equal side #\L))
        (setf notran (char-equal trans #\N))
        (cond
          (left
           (setf nq m))
          (t
           (setf nq n)))
        (cond
          ((and (not left) (not (char-equal side #\R)))
           (setf info -1))
          ((and (not notran) (not (char-equal trans #\T)))
           (setf info -2))
          ((< m 0)
           (setf info -3))
          ((< n 0)
           (setf info -4))
          ((or (< k 0) (> k nq))
           (setf info -5))
          ((< lda (max (the fixnum 1) (the fixnum k)))
           (setf info -7))
          ((< ldc (max (the fixnum 1) (the fixnum m)))
           (setf info -10)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
```

```
             "DORML2" (f2cl-lib:int-sub info))
            (go end_label)))
         (if (or (= m 0) (= n 0) (= k 0)) (go end_label))
         (cond
           ((or (and left notran) (and (not left) (not notran)))
            (setf i1 1)
            (setf i2 k)
            (setf i3 1))
           (t
            (setf i1 k)
            (setf i2 1)
            (setf i3 -1)))
         (cond
           (left
            (setf ni n)
            (setf jc 1))
           (t
            (setf mi m)
            (setf ic 1)))
         (f2cl-lib:fdo (i i1 (f2cl-lib:int-add i i3))
                       ((> i i2) nil)
           (tagbody
             (cond
               (left
                (setf mi (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1))
                (setf ic i))
               (t
                (setf ni (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1))
                (setf jc i)))
             (setf aii
                     (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%))
             (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                     one)
             (dlarf side mi ni
              (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
              (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
              (f2cl-lib:array-slice c double-float (ic jc) ((1 ldc) (1 *))) ldc
              work)
             (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                     aii)))
       end_label
         (return (values nil nil nil nil nil nil nil nil nil nil nil info))))))
```

7.88 dormlq LAPACK

⟨*dormlq.input*⟩≡

```
)set break resume
)sys rm -f dormlq.output
)spool dormlq.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dormlq.help*⟩≡

```
=====================================================================
dormlq examples
=====================================================================

=====================================================================
Man Page Details
=====================================================================

NAME
       DORMLQ  - the general real M-by-N matrix C with   SIDE = 'L' SIDE = 'R'
       TRANS = 'N'

SYNOPSIS
       SUBROUTINE DORMLQ( SIDE, TRANS, M, N, K, A, LDA,  TAU,  C,  LDC,  WORK,
                          LWORK, INFO )

           CHARACTER      SIDE, TRANS

           INTEGER        INFO, K, LDA, LDC, LWORK, M, N

           DOUBLE         PRECISION  A( LDA, * ), C( LDC, * ), TAU( * ), WORK(
                          * )

PURPOSE
       DORMLQ overwrites the general real M-by-N matrix C with  TRANS  =  'T':
       Q**T * C       C * Q**T

       where Q is a real orthogonal matrix defined as the product of k elemen-
       tary reflectors

             Q = H(k) . . . H(2) H(1)

       as returned by DGELQF. Q is of order M if SIDE = 'L' and of order N  if
       SIDE = 'R'.


ARGUMENTS
       SIDE    (input) CHARACTER*1
               = 'L': apply Q or Q**T from the Left;
               = 'R': apply Q or Q**T from the Right.

       TRANS   (input) CHARACTER*1
               = 'N':  No transpose, apply Q;
               = 'T':  Transpose, apply Q**T.
```

```
M       (input) INTEGER
        The number of rows of the matrix C. M >= 0.

N       (input) INTEGER
        The number of columns of the matrix C. N >= 0.

K       (input) INTEGER
        The  number  of elementary reflectors whose product defines the
        matrix Q.  If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >=
        0.

A       (input) DOUBLE PRECISION array, dimension
        (LDA,M)  if SIDE = 'L', (LDA,N) if SIDE = 'R' The i-th row must
        contain the vector which defines the elementary reflector H(i),
        for i = 1,2,...,k, as returned by DGELQF in the first k rows of
        its array argument  A.  A  is  modified  by  the  routine  but
        restored on exit.

LDA     (input) INTEGER
        The leading dimension of the array A. LDA >= max(1,K).

TAU     (input) DOUBLE PRECISION array, dimension (K)
        TAU(i) must contain the scalar factor of the elementary reflec-
        tor H(i), as returned by DGELQF.

C       (input/output) DOUBLE PRECISION array, dimension (LDC,N)
        On entry, the M-by-N matrix C.  On exit, C  is  overwritten  by
        Q*C or Q**T*C or C*Q**T or C*Q.

LDC     (input) INTEGER
        The leading dimension of the array C. LDC >= max(1,M).

WORK       (workspace/output)  DOUBLE  PRECISION  array,  dimension
(MAX(1,LWORK))
        On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK   (input) INTEGER
        The  dimension  of  the  array  WORK.   If SIDE = 'L', LWORK >=
        max(1,N); if SIDE = 'R', LWORK >= max(1,M).  For  optimum  per-
        formance LWORK >= N*NB if SIDE = 'L', and LWORK >= M*NB if SIDE
        = 'R', where NB is the optimal blocksize.

        If LWORK = -1, then a workspace query is assumed;  the  routine
        only  calculates  the  optimal  size of the WORK array, returns
        this value as the first entry of the WORK array, and  no  error
        message related to LWORK is issued by XERBLA.
```

```
INFO    (output) INTEGER
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value
```

⟨*LAPACK dormlq*⟩≡

```
(let* ((nbmax 64) (ldt (+ nbmax 1)))
  (declare (type (fixnum 64 64) nbmax)
           (type fixnum ldt))
  (defun dormlq (side trans m n k a lda tau c ldc work lwork info)
    (declare (type (simple-array double-float (*)) work c tau a)
             (type fixnum info lwork ldc lda k n m)
             (type character trans side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (trans character trans-%data% trans-%offset%)
         (a double-float a-%data% a-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (c double-float c-%data% c-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((i 0) (i1 0) (i2 0) (i3 0) (ib 0) (ic 0) (iinfo 0) (iws 0) (jc 0)
             (ldwork 0) (lwkopt 0) (mi 0) (nb 0) (nbmin 0) (ni 0) (nq 0) (nw 0)
             (transt
              (make-array '(1) :element-type 'character :initial-element #\ ))
             (left nil) (lquery nil) (notran nil)
             (t$
              (make-array (the fixnum (reduce #'* (list ldt nbmax)))
                          :element-type 'double-float)))
        (declare (type (simple-array double-float (*)) t$)
                 (type fixnum i i1 i2 i3 ib ic iinfo iws jc ldwork
                                        lwkopt mi nb nbmin ni nq nw)
                 (type (simple-array character (1)) transt)
                 (type (member t nil) left lquery notran))
        (setf info 0)
        (setf left (char-equal side #\L))
        (setf notran (char-equal trans #\N))
        (setf lquery (coerce (= lwork -1) '(member t nil)))
        (cond
          (left
           (setf nq m)
           (setf nw n))
          (t
           (setf nq n)
           (setf nw m)))
        (cond
          ((and (not left) (not (char-equal side #\R)))
           (setf info -1))
          ((and (not notran) (not (char-equal trans #\T)))
           (setf info -2))
          ((< m 0)
           (setf info -3))
```

```
  ((< n 0)
   (setf info -4))
  ((or (< k 0) (> k nq))
   (setf info -5))
  ((< lda (max (the fixnum 1) (the fixnum k)))
   (setf info -7))
  ((< ldc (max (the fixnum 1) (the fixnum m)))
   (setf info -10))
  ((and
    (< lwork
       (max (the fixnum 1) (the fixnum nw)))
    (not lquery))
   (setf info -12)))
(cond
  ((= info 0)
   (setf nb
           (min (the fixnum nbmax)
                (the fixnum
                     (ilaenv 1 "DORMLQ" (f2cl-lib:f2cl-// side trans) m
                      n k -1))))
   (setf lwkopt
           (f2cl-lib:int-mul
            (max (the fixnum 1) (the fixnum nw))
            nb))
   (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
           (coerce (the fixnum lwkopt) 'double-float))))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DORMLQ" (f2cl-lib:int-sub info))
   (go end_label))
  (lquery
   (go end_label)))
(cond
  ((or (= m 0) (= n 0) (= k 0))
   (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
           (coerce (the fixnum 1) 'double-float))
   (go end_label)))
(setf nbmin 2)
(setf ldwork nw)
(cond
  ((and (> nb 1) (< nb k))
   (setf iws (f2cl-lib:int-mul nw nb))
   (cond
     ((< lwork iws)
```

```
            (setf nb (the fixnum (truncate lwork ldwork)))
            (setf nbmin
                    (max (the fixnum 2)
                         (the fixnum
                              (ilaenv 2 "DORMLQ"
                               (f2cl-lib:f2cl-// side trans) m n k -1)))))))
          (t
           (setf iws nw)))
        (cond
          ((or (< nb nbmin) (>= nb k))
           (multiple-value-bind
                 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
                  var-10 var-11)
               (dorml2 side trans m n k a lda tau c ldc work iinfo)
             (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                              var-8 var-9 var-10))
             (setf iinfo var-11)))
          (t
           (cond
             ((or (and left notran) (and (not left) (not notran)))
              (setf i1 1)
              (setf i2 k)
              (setf i3 nb))
             (t
              (setf i1
                      (+ (* (the fixnum (truncate (- k 1) nb)) nb)
                         1))
              (setf i2 1)
              (setf i3 (f2cl-lib:int-sub nb))))
           (cond
             (left
              (setf ni n)
              (setf jc 1))
             (t
              (setf mi m)
              (setf ic 1)))
           (cond
             (notran
              (f2cl-lib:f2cl-set-string transt "T" (string 1)))
             (t
              (f2cl-lib:f2cl-set-string transt "N" (string 1))))
           (f2cl-lib:fdo (i i1 (f2cl-lib:int-add i i3))
                         ((> i i2) nil)
             (tagbody
               (setf ib
                       (min (the fixnum nb)
```

```
                              (the fixnum
                                   (f2cl-lib:int-add (f2cl-lib:int-sub k i) 1))))
                 (dlarft "Forward" "Rowwise"
                  (f2cl-lib:int-add (f2cl-lib:int-sub nq i) 1) ib
                  (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
                  (f2cl-lib:array-slice tau double-float (i) ((1 *))) t$ ldt)
                 (cond
                   (left
                    (setf mi (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1))
                    (setf ic i))
                   (t
                    (setf ni (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1))
                    (setf jc i)))
                 (dlarfb side transt "Forward" "Rowwise" mi ni ib
                  (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
                  t$ ldt
                  (f2cl-lib:array-slice c double-float (ic jc) ((1 ldc) (1 *)))
                  ldc work ldwork)))))
          (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                  (coerce (the fixnum lwkopt) 'double-float))
   end_label
          (return
           (values nil nil nil nil nil nil nil nil nil nil nil nil info))))))
```

7.89 dormqr LAPACK

⟨*dormqr.input*⟩≡

```
)set break resume
)sys rm -f dormqr.output
)spool dormqr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dormqr.help*⟩≡

```
  ==========================================================================
  dormqr examples
  ==========================================================================

  ==========================================================================
  Man Page Details
  ==========================================================================

  NAME
         DORMQR  - the general real M-by-N matrix C with   SIDE = 'L' SIDE = 'R'
         TRANS = 'N'

  SYNOPSIS
         SUBROUTINE DORMQR( SIDE, TRANS, M, N, K, A, LDA,  TAU,  C,  LDC,  WORK,
                            LWORK, INFO )

             CHARACTER      SIDE, TRANS

             INTEGER        INFO, K, LDA, LDC, LWORK, M, N

             DOUBLE         PRECISION  A( LDA, * ), C( LDC, * ), TAU( * ), WORK(
                            * )

  PURPOSE
         DORMQR overwrites the general real M-by-N matrix C with  TRANS  =  'T':
         Q**T * C       C * Q**T

         where Q is a real orthogonal matrix defined as the product of k elemen-
         tary reflectors

               Q = H(1) H(2) . . . H(k)

         as returned by DGEQRF. Q is of order M if SIDE = 'L' and of order N  if
         SIDE = 'R'.


  ARGUMENTS
         SIDE    (input) CHARACTER*1
                 = 'L': apply Q or Q**T from the Left;
                 = 'R': apply Q or Q**T from the Right.

         TRANS   (input) CHARACTER*1
                 = 'N':  No transpose, apply Q;
                 = 'T':  Transpose, apply Q**T.
```

```
M       (input) INTEGER
        The number of rows of the matrix C. M >= 0.

N       (input) INTEGER
        The number of columns of the matrix C. N >= 0.

K       (input) INTEGER
        The  number  of elementary reflectors whose product defines the
        matrix Q.  If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >=
        0.

A       (input) DOUBLE PRECISION array, dimension (LDA,K)
        The  i-th column must contain the vector which defines the ele-
        mentary reflector H(i), for i = 1,2,...,k, as returned by  DGE-
        QRF in the first k columns of its array argument A.  A is modi-
        fied by the routine but restored on exit.

LDA     (input) INTEGER
        The leading dimension of the array A.  If SIDE =  'L',  LDA  >=
        max(1,M); if SIDE = 'R', LDA >= max(1,N).

TAU     (input) DOUBLE PRECISION array, dimension (K)
        TAU(i) must contain the scalar factor of the elementary reflec-
        tor H(i), as returned by DGEQRF.

C       (input/output) DOUBLE PRECISION array, dimension (LDC,N)
        On entry, the M-by-N matrix C.  On exit, C  is  overwritten  by
        Q*C or Q**T*C or C*Q**T or C*Q.

LDC     (input) INTEGER
        The leading dimension of the array C. LDC >= max(1,M).

WORK       (workspace/output)  DOUBLE  PRECISION  array,  dimension
(MAX(1,LWORK))
        On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK   (input) INTEGER
        The  dimension  of  the  array  WORK.   If SIDE = 'L', LWORK >=
        max(1,N); if SIDE = 'R', LWORK >= max(1,M).  For  optimum  per-
        formance LWORK >= N*NB if SIDE = 'L', and LWORK >= M*NB if SIDE
        = 'R', where NB is the optimal blocksize.

        If LWORK = -1, then a workspace query is assumed;  the  routine
        only  calculates  the  optimal  size of the WORK array, returns
        this value as the first entry of the WORK array, and  no  error
        message related to LWORK is issued by XERBLA.
```

```
INFO    (output) INTEGER
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value
```

⟨*LAPACK dormqr*⟩≡

```
(let* ((nbmax 64) (ldt (+ nbmax 1)))
  (declare (type (fixnum 64 64) nbmax)
           (type fixnum ldt))
  (defun dormqr (side trans m n k a lda tau c ldc work lwork info)
    (declare (type (simple-array double-float (*)) work c tau a)
             (type fixnum info lwork ldc lda k n m)
             (type character trans side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (trans character trans-%data% trans-%offset%)
         (a double-float a-%data% a-%offset%)
         (tau double-float tau-%data% tau-%offset%)
         (c double-float c-%data% c-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((i 0) (i1 0) (i2 0) (i3 0) (ib 0) (ic 0) (iinfo 0) (iws 0) (jc 0)
             (ldwork 0) (lwkopt 0) (mi 0) (nb 0) (nbmin 0) (ni 0) (nq 0) (nw 0)
             (left nil) (lquery nil) (notran nil)
             (t$
              (make-array (the fixnum (reduce #'* (list ldt nbmax)))
                          :element-type 'double-float)))
        (declare (type (simple-array double-float (*)) t$)
                 (type fixnum i i1 i2 i3 ib ic iinfo iws jc ldwork
                                          lwkopt mi nb nbmin ni nq nw)
                 (type (member t nil) left lquery notran))
        (setf info 0)
        (setf left (char-equal side #\L))
        (setf notran (char-equal trans #\N))
        (setf lquery (coerce (= lwork -1) '(member t nil)))
        (cond
          (left
           (setf nq m)
           (setf nw n))
          (t
           (setf nq n)
           (setf nw m)))
        (cond
          ((and (not left) (not (char-equal side #\R)))
           (setf info -1))
          ((and (not notran) (not (char-equal trans #\T)))
           (setf info -2))
          ((< m 0)
           (setf info -3))
          ((< n 0)
           (setf info -4))
          ((or (< k 0) (> k nq))
```

```
    (setf info -5))
   ((< lda (max (the fixnum 1) (the fixnum nq)))
    (setf info -7))
   ((< ldc (max (the fixnum 1) (the fixnum m)))
    (setf info -10))
   ((and
     (< lwork
        (max (the fixnum 1) (the fixnum nw)))
     (not lquery))
    (setf info -12)))
 (cond
   ((= info 0)
    (setf nb
          (min (the fixnum nbmax)
               (the fixnum
                    (ilaenv 1 "DORMQR" (f2cl-lib:f2cl-// side trans) m
                     n k -1))))
    (setf lwkopt
          (f2cl-lib:int-mul
           (max (the fixnum 1) (the fixnum nw))
           nb))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
          (coerce (the fixnum lwkopt) 'double-float))))
 (cond
   ((/= info 0)
    (error
     " ** On entry to ~a parameter number ~a had an illegal value~%"
     "DORMQR" (f2cl-lib:int-sub info))
    (go end_label))
   (lquery
    (go end_label)))
 (cond
   ((or (= m 0) (= n 0) (= k 0))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
          (coerce (the fixnum 1) 'double-float))
    (go end_label)))
 (setf nbmin 2)
 (setf ldwork nw)
 (cond
   ((and (> nb 1) (< nb k))
    (setf iws (f2cl-lib:int-mul nw nb))
    (cond
      ((< lwork iws)
       (setf nb (the fixnum (truncate lwork ldwork)))
       (setf nbmin
             (max (the fixnum 2)
```

```
                    (the fixnum
                         (ilaenv 2 "DORMQR"
                          (f2cl-lib:f2cl-// side trans) m n k -1)))))))
      (t
       (setf iws nw)))
    (cond
      ((or (< nb nbmin) (>= nb k))
       (multiple-value-bind
             (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
              var-10 var-11)
           (dorm2r side trans m n k a lda tau c ldc work iinfo)
         (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                          var-8 var-9 var-10))
         (setf iinfo var-11)))
      (t
       (cond
         ((or (and left (not notran)) (and (not left) notran))
          (setf i1 1)
          (setf i2 k)
          (setf i3 nb))
         (t
          (setf i1
                  (+ (* (the fixnum (truncate (- k 1) nb)) nb)
                     1))
          (setf i2 1)
          (setf i3 (f2cl-lib:int-sub nb))))
       (cond
         (left
          (setf ni n)
          (setf jc 1))
         (t
          (setf mi m)
          (setf ic 1)))
       (f2cl-lib:fdo (i i1 (f2cl-lib:int-add i i3))
                     ((> i i2) nil)
         (tagbody
           (setf ib
                   (min (the fixnum nb)
                        (the fixnum
                             (f2cl-lib:int-add (f2cl-lib:int-sub k i) 1))))
           (dlarft "Forward" "Columnwise"
            (f2cl-lib:int-add (f2cl-lib:int-sub nq i) 1) ib
            (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
            (f2cl-lib:array-slice tau double-float (i) ((1 *))) t$ ldt)
           (cond
             (left
```

```
                  (setf mi (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1))
                  (setf ic i))
                 (t
                  (setf ni (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1))
                  (setf jc i)))
                (dlarfb side trans "Forward" "Columnwise" mi ni ib
                 (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
                 t$ ldt
                 (f2cl-lib:array-slice c double-float (ic jc) ((1 ldc) (1 *)))
                 ldc work ldwork)))))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                (coerce (the fixnum lwkopt) 'double-float))
  end_label
        (return
         (values nil nil nil nil nil nil nil nil nil nil nil nil info))))))
```

7.90 dtrevc LAPACK

⟨*dtrevc.input*⟩≡

```
)set break resume
)sys rm -f dtrevc.output
)spool dtrevc.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dtrevc.help*⟩≡

```
======================================================================
dtrevc examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DTREVC  -  some  or all of the right and/or left eigenvectors of a real
       upper quasi-triangular matrix T

SYNOPSIS
       SUBROUTINE DTREVC( SIDE, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR,
                          MM, M, WORK, INFO )

           CHARACTER      HOWMNY, SIDE

           INTEGER        INFO, LDT, LDVL, LDVR, M, MM, N

           LOGICAL        SELECT( * )

           DOUBLE         PRECISION T( LDT, * ), VL( LDVL, * ), VR( LDVR, * ),
                          WORK( * )

PURPOSE
       DTREVC computes some or all of the right and/or left eigenvectors of  a
       real  upper  quasi-triangular matrix T.  Matrices of this type are pro-
       duced by the Schur  factorization  of  a  real  general  matrix:  A  =
       Q*T*Q**T, as computed by DHSEQR.

       The  right  eigenvector x and the left eigenvector y of T corresponding
       to an eigenvalue w are defined by:

          T*x = w*x,     (y**H)*T = w*(y**H)

       where y**H denotes the conjugate transpose of y.
       The eigenvalues are not input to this routine, but  are  read  directly
       from the diagonal blocks of T.

       This  routine  returns the matrices X and/or Y of right and left eigen-
       vectors of T, or the products Q*X and/or  Q*Y,  where  Q  is  an  input
       matrix.  If Q is the orthogonal factor that reduces a matrix A to Schur
       form T, then Q*X and Q*Y are the matrices of right and  left  eigenvec-
       tors of A.
```

```
ARGUMENTS
       SIDE    (input) CHARACTER*1
               = 'R':  compute right eigenvectors only;
               = 'L':  compute left eigenvectors only;
               = 'B':  compute both right and left eigenvectors.

       HOWMNY  (input) CHARACTER*1
               = 'A':  compute all right and/or left eigenvectors;
               =  'B':  compute all right and/or left eigenvectors, backtrans-
               formed by the matrices in VR and/or VL; = 'S':  compute
               selected right and/or left eigenvectors, as indicated by the
               logical array SELECT.

       SELECT  (input/output) LOGICAL array, dimension (N)
               If HOWMNY = 'S', SELECT specifies the eigenvectors to be com-
               puted.  If w(j) is a real eigenvalue, the corresponding real
               eigenvector is computed if SELECT(j) is .TRUE..  If w(j) and
               w(j+1) are the real and imaginary parts of a complex eigen-
               value, the corresponding complex eigenvector is computed if
               either SELECT(j) or SELECT(j+1) is .TRUE., and on exit
               SELECT(j) is set to .TRUE. and SELECT(j+1) is set to Not refer-
               enced if HOWMNY = 'A' or 'B'.

       N       (input) INTEGER
               The order of the matrix T. N >= 0.

       T       (input) DOUBLE PRECISION array, dimension (LDT,N)
               The upper quasi-triangular matrix T in Schur canonical form.

       LDT     (input) INTEGER
               The leading dimension of the array T. LDT >= max(1,N).

       VL      (input/output) DOUBLE PRECISION array, dimension (LDVL,MM)
               On entry, if SIDE = 'L' or 'B' and HOWMNY = 'B', VL must con-
               tain an N-by-N matrix Q (usually the orthogonal matrix Q of
               Schur vectors returned by DHSEQR).  On exit, if SIDE = 'L' or
               'B', VL contains: if HOWMNY = 'A', the matrix Y of left eigen-
               vectors of T; if HOWMNY = 'B', the matrix Q*Y; if HOWMNY = 'S',
               the left eigenvectors of T specified by SELECT, stored consecu-
               tively in the columns of VL, in the same order as their eigen-
               values. A complex eigenvector corresponding to a complex
               eigenvalue is stored in two consecutive columns, the first
               holding the real part, and the second the imaginary part.  Not
               referenced if SIDE = 'R'.
```

```
        LDVL    (input) INTEGER
                The  leading dimension of the array VL.  LDVL >= 1, and if SIDE
                = 'L' or 'B', LDVL >= N.

        VR      (input/output) DOUBLE PRECISION array, dimension (LDVR,MM)
                On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR  must  con-
                tain  an  N-by-N  matrix  Q (usually the orthogonal matrix Q of
                Schur vectors returned by DHSEQR).  On exit, if SIDE =  'R'  or
                'B', VR contains: if HOWMNY = 'A', the matrix X of right eigen-
                vectors of T; if HOWMNY = 'B', the matrix Q*X; if HOWMNY = 'S',
                the right eigenvectors of T specified by SELECT, stored consec-
                utively in the columns of VR, in the same order as their eigen-
                values.  A  complex  eigenvector  corresponding  to  a complex
                eigenvalue is stored in  two  consecutive  columns,  the  first
                holding  the  real part and the second the imaginary part.  Not
                referenced if SIDE = 'L'.

        LDVR    (input) INTEGER
                The leading dimension of the array VR.  LDVR >= 1, and if  SIDE
                = 'R' or 'B', LDVR >= N.

        MM      (input) INTEGER
                The number of columns in the arrays VL and/or VR. MM >= M.

        M       (output) INTEGER
                The  number of columns in the arrays VL and/or VR actually used
                to store the eigenvectors.  If HOWMNY = 'A' or 'B', M is set to
                N.  Each selected real eigenvector occupies one column and each
                selected complex eigenvector occupies two columns.

        WORK    (workspace) DOUBLE PRECISION array, dimension (3*N)

        INFO    (output) INTEGER
                = 0:  successful exit
                < 0:  if INFO = -i, the i-th argument had an illegal value

    FURTHER DETAILS
        The algorithm used in this program is basically backward (forward) sub-
        stitution,  with  scaling  to make the the code robust against possible
        overflow.

        Each eigenvector is normalized so that the element of largest magnitude
        has  magnitude 1; here the magnitude of a complex number (x,y) is taken
        to be |x| + |y|.
```

⟨*LAPACK dtrevc*⟩≡

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dtrevc (side howmny select n t$ ldt vl ldvl vr ldvr mm m work info)
    (declare (type (simple-array double-float (*)) work vr vl t$)
             (type fixnum info m mm ldvr ldvl ldt n)
             (type (simple-array (member t nil) (*)) select)
             (type character howmny side))
    (f2cl-lib:with-multi-array-data
        ((side character side-%data% side-%offset%)
         (howmny character howmny-%data% howmny-%offset%)
         (select (member t nil) select-%data% select-%offset%)
         (t$ double-float t$-%data% t$-%offset%)
         (vl double-float vl-%data% vl-%offset%)
         (vr double-float vr-%data% vr-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((x (make-array 4 :element-type 'double-float)) (beta 0.0)
             (bignum 0.0) (emax 0.0) (ovfl 0.0) (rec 0.0) (remax 0.0)
             (scale 0.0) (smin 0.0) (smlnum 0.0) (ulp 0.0) (unfl 0.0)
             (vcrit 0.0) (vmax 0.0) (wi 0.0) (wr 0.0) (xnorm 0.0) (i 0)
             (ierr 0) (ii 0) (ip 0) (is 0) (j 0) (j1 0) (j2 0) (jnxt 0) (k 0)
             (ki 0) (n2 0) (allv nil) (bothv nil) (leftv nil) (over nil)
             (pair nil) (rightv nil) (somev nil) (sqrt$ 0.0f0))
        (declare (type (single-float) sqrt$)
                 (type (simple-array double-float (4)) x)
                 (type (double-float) beta bignum emax ovfl rec remax scale
                                      smin smlnum ulp unfl vcrit vmax wi wr
                                      xnorm)
                 (type fixnum i ierr ii ip is j j1 j2 jnxt k ki
                                    n2)
                 (type (member t nil) allv bothv leftv over pair rightv
                                      somev))
        (setf bothv (char-equal side #\B))
        (setf rightv (or (char-equal side #\R) bothv))
        (setf leftv (or (char-equal side #\L) bothv))
        (setf allv (char-equal howmny #\A))
        (setf over (char-equal howmny #\B))
        (setf somev (char-equal howmny #\S))
        (setf info 0)
        (cond
          ((and (not rightv) (not leftv))
           (setf info -1))
          ((and (not allv) (not over) (not somev))
           (setf info -2))
          ((< n 0)
```

```
 (setf info -4))
((< ldt (max (the fixnum 1) (the fixnum n)))
 (setf info -6))
((or (< ldvl 1) (and leftv (< ldvl n)))
 (setf info -8))
((or (< ldvr 1) (and rightv (< ldvr n)))
 (setf info -10))
(t
 (cond
   (somev
    (setf m 0)
    (setf pair nil)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  ((> j n) nil)
      (tagbody
        (cond
          (pair
           (setf pair nil)
           (setf (f2cl-lib:fref select-%data%
                                (j)
                                ((1 *))
                                select-%offset%)
                  nil))
          (t
           (cond
             ((< j n)
              (cond
                ((=
                  (f2cl-lib:fref t$
                                 ((f2cl-lib:int-add j 1) j)
                                 ((1 ldt) (1 *)))
                  zero)
                 (if
                  (f2cl-lib:fref select-%data%
                                 (j)
                                 ((1 *))
                                 select-%offset%)
                  (setf m (f2cl-lib:int-add m 1))))
                (t
                 (setf pair t)
                 (cond
                   ((or (f2cl-lib:fref select (j) ((1 *)))
                        (f2cl-lib:fref select
                                       ((f2cl-lib:int-add j 1))
                                       ((1 *))))
                    (setf (f2cl-lib:fref select-%data%
```

```
                                                (j)
                                                ((1 *))
                                                select-%offset%)
                                        t)
                                (setf m (f2cl-lib:int-add m 2)))))))
                        (t
                         (if
                          (f2cl-lib:fref select-%data%
                                         (n)
                                         ((1 *))
                                         select-%offset%)
                          (setf m (f2cl-lib:int-add m 1)))))))))))
             (t
              (setf m n)))
           (cond
             ((< mm m)
              (setf info -11)))))
      (cond
        ((/= info 0)
         (error
          " ** On entry to ~a parameter number ~a had an illegal value~%"
          "DTREVC" (f2cl-lib:int-sub info))
         (go end_label)))
      (if (= n 0) (go end_label))
      (setf unfl (dlamch "Safe minimum"))
      (setf ovfl (/ one unfl))
      (multiple-value-bind (var-0 var-1)
          (dlabad unfl ovfl)
        (declare (ignore))
        (setf unfl var-0)
        (setf ovfl var-1))
      (setf ulp (dlamch "Precision"))
      (setf smlnum (* unfl (/ n ulp)))
      (setf bignum (/ (- one ulp) smlnum))
      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) zero)
      (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%) zero)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))) nil)
            (tagbody
              (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
                    (+
                     (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
                     (abs
```

```
                          (f2cl-lib:fref t$-%data%
                                        (i j)
                                        ((1 ldt) (1 *))
                                        t$-%offset%))))))))
        (setf n2 (f2cl-lib:int-mul 2 n))
        (cond
          (rightv
           (setf ip 0)
           (setf is m)
           (f2cl-lib:fdo (ki n (f2cl-lib:int-add ki (f2cl-lib:int-sub 1)))
                         ((> ki 1) nil)
             (tagbody
               (if (= ip 1) (go label130))
               (if (= ki 1) (go label40))
               (if
                (=
                 (f2cl-lib:fref t$-%data%
                                (ki (f2cl-lib:int-sub ki 1))
                                ((1 ldt) (1 *))
                                t$-%offset%)
                 zero)
                (go label40))
               (setf ip -1)
  label40
               (cond
                 (somev
                  (cond
                    ((= ip 0)
                     (if
                      (not
                       (f2cl-lib:fref select-%data%
                                      (ki)
                                      ((1 *))
                                      select-%offset%))
                      (go label130)))
                    (t
                     (if
                      (not
                       (f2cl-lib:fref select-%data%
                                      ((f2cl-lib:int-sub ki 1))
                                      ((1 *))
                                      select-%offset%))
                      (go label130))))))
               (setf wr
                       (f2cl-lib:fref t$-%data%
                                      (ki ki)
```

```
                        ((1 ldt) (1 *))
                        t$-%offset%))
(setf wi zero)
(if (/= ip 0)
    (setf wi
            (*
             (f2cl-lib:fsqrt
              (abs
               (f2cl-lib:fref t$-%data%
                              (ki (f2cl-lib:int-sub ki 1))
                              ((1 ldt) (1 *))
                              t$-%offset%)))
             (f2cl-lib:fsqrt
              (abs
               (f2cl-lib:fref t$-%data%
                              ((f2cl-lib:int-sub ki 1) ki)
                              ((1 ldt) (1 *))
                              t$-%offset%))))))
(setf smin (max (* ulp (+ (abs wr) (abs wi))) smlnum))
(cond
  ((= ip 0)
   (setf (f2cl-lib:fref work-%data%
                        ((f2cl-lib:int-add ki n))
                        ((1 *))
                        work-%offset%)
         one)
   (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                 ((> k
                     (f2cl-lib:int-add ki (f2cl-lib:int-sub 1)))
                  nil)
     (tagbody
       (setf (f2cl-lib:fref work-%data%
                            ((f2cl-lib:int-add k n))
                            ((1 *))
                            work-%offset%)
             (-
              (f2cl-lib:fref t$-%data%
                             (k ki)
                             ((1 ldt) (1 *))
                             t$-%offset%)))))
   (setf jnxt (f2cl-lib:int-sub ki 1))
   (f2cl-lib:fdo (j (f2cl-lib:int-add ki (f2cl-lib:int-sub 1))
                  (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                 ((> j 1) nil)
     (tagbody
       (if (> j jnxt) (go label60))
```

```
(setf j1 j)
(setf j2 j)
(setf jnxt (f2cl-lib:int-sub j 1))
(cond
  ((> j 1)
   (cond
     ((/=
       (f2cl-lib:fref t$
                      (j
                       (f2cl-lib:int-add j
                                         (f2cl-lib:int-sub
                                          1)))
                      ((1 ldt) (1 *)))
       zero)
      (setf j1 (f2cl-lib:int-sub j 1))
      (setf jnxt (f2cl-lib:int-sub j 2))))))
(cond
  ((= j1 j2)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
          var-8 var-9 var-10 var-11 var-12 var-13 var-14
          var-15 var-16 var-17)
       (dlaln2 nil 1 1 smin one
        (f2cl-lib:array-slice t$
                              double-float
                              (j j)
                              ((1 ldt) (1 *)))
        ldt one one
        (f2cl-lib:array-slice work
                              double-float
                              ((+ j n))
                              ((1 *)))
        n wr zero x 2 scale xnorm ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                      var-6 var-7 var-8 var-9 var-10
                      var-11 var-12 var-13 var-14))
     (setf scale var-15)
     (setf xnorm var-16)
     (setf ierr var-17))
   (cond
     ((> xnorm one)
      (cond
        ((> (f2cl-lib:fref work (j) ((1 *)))
            (f2cl-lib:f2cl/ bignum xnorm))
         (setf (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
                 (/ (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
```

```
                    xnorm))
       (setf scale (/ scale xnorm))))))
 (if (/= scale one)
     (dscal ki scale
      (f2cl-lib:array-slice work
                            double-float
                            ((+ 1 n))
                            ((1 *)))
      1))
 (setf (f2cl-lib:fref work-%data%
                      ((f2cl-lib:int-add j n))
                      ((1 *))
                      work-%offset%)
         (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
 (daxpy (f2cl-lib:int-sub j 1)
  (- (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
  (f2cl-lib:array-slice t$
                        double-float
                        (1 j)
                        ((1 ldt) (1 *)))
  1
  (f2cl-lib:array-slice work
                        double-float
                        ((+ 1 n))
                        ((1 *)))
  1))
(t
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10 var-11 var-12 var-13 var-14
        var-15 var-16 var-17)
     (dlaln2 nil 2 1 smin one
      (f2cl-lib:array-slice t$
                            double-float
                            ((+ j (f2cl-lib:int-sub 1))
                             (f2cl-lib:int-sub j 1))
                            ((1 ldt) (1 *)))
      ldt one one
      (f2cl-lib:array-slice work
                            double-float
                            ((+ j
                                (f2cl-lib:int-sub 1)
                                n))
                            ((1 *)))
      n wr zero x 2 scale xnorm ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
```

```
                  var-6 var-7 var-8 var-9 var-10
                  var-11 var-12 var-13 var-14))
  (setf scale var-15)
  (setf xnorm var-16)
  (setf ierr var-17))
(cond
  ((> xnorm one)
   (setf beta
         (max
          (f2cl-lib:fref work-%data%
                         ((f2cl-lib:int-sub j 1))
                         ((1 *))
                         work-%offset%)
          (f2cl-lib:fref work-%data%
                         (j)
                         ((1 *))
                         work-%offset%)))
   (cond
     ((> beta (f2cl-lib:f2cl/ bignum xnorm))
      (setf (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
            (/ (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
               xnorm))
      (setf (f2cl-lib:fref x (2 1) ((1 2) (1 2)))
            (/ (f2cl-lib:fref x (2 1) ((1 2) (1 2)))
               xnorm))
      (setf scale (/ scale xnorm))))))
(if (/= scale one)
    (dscal ki scale
     (f2cl-lib:array-slice work
                           double-float
                           ((+ 1 n))
                           ((1 *)))
     1))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add
                       (f2cl-lib:int-sub j 1)
                       n))
                     ((1 *))
                     work-%offset%)
       (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j n))
                     ((1 *))
                     work-%offset%)
       (f2cl-lib:fref x (2 1) ((1 2) (1 2))))
(daxpy (f2cl-lib:int-sub j 2)
```

```
                        (- (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
                        (f2cl-lib:array-slice t$
                                              double-float
                                              (1 (f2cl-lib:int-sub j 1))
                                              ((1 ldt) (1 *)))
                        1
                        (f2cl-lib:array-slice work
                                              double-float
                                              ((+ 1 n))
                                              ((1 *)))
                        1)
                       (daxpy (f2cl-lib:int-sub j 2)
                        (- (f2cl-lib:fref x (2 1) ((1 2) (1 2))))
                        (f2cl-lib:array-slice t$
                                              double-float
                                              (1 j)
                                              ((1 ldt) (1 *)))
                        1
                        (f2cl-lib:array-slice work
                                              double-float
                                              ((+ 1 n))
                                              ((1 *)))
                        1)))
  label60))
                  (cond
                    ((not over)
                     (dcopy ki
                      (f2cl-lib:array-slice work
                                            double-float
                                            ((+ 1 n))
                                            ((1 *)))
                      1
                      (f2cl-lib:array-slice vr
                                            double-float
                                            (1 is)
                                            ((1 ldvr) (1 *)))
                      1)
                     (setf ii
                             (idamax ki
                              (f2cl-lib:array-slice vr
                                                    double-float
                                                    (1 is)
                                                    ((1 ldvr) (1 *)))
                              1))
                     (setf remax
                             (/ one
```

```
            (abs
             (f2cl-lib:fref vr-%data%
                            (ii is)
                            ((1 ldvr) (1 *))
                            vr-%offset%))))
 (dscal ki remax
  (f2cl-lib:array-slice vr
                        double-float
                        (1 is)
                        ((1 ldvr) (1 *)))
  1)
 (f2cl-lib:fdo (k (f2cl-lib:int-add ki 1)
                  (f2cl-lib:int-add k 1))
               ((> k n) nil)
   (tagbody
     (setf (f2cl-lib:fref vr-%data%
                          (k is)
                          ((1 ldvr) (1 *))
                          vr-%offset%)
             zero))))
(t
 (if (> ki 1)
     (dgemv "N" n (f2cl-lib:int-sub ki 1) one vr ldvr
      (f2cl-lib:array-slice work
                            double-float
                            ((+ 1 n))
                            ((1 *)))
      1
      (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add ki n))
                     ((1 *))
                     work-%offset%)
      (f2cl-lib:array-slice vr
                            double-float
                            (1 ki)
                            ((1 ldvr) (1 *)))
      1))
 (setf ii
         (idamax n
          (f2cl-lib:array-slice vr
                                double-float
                                (1 ki)
                                ((1 ldvr) (1 *)))
          1))
 (setf remax
         (/ one
```

```
                        (abs
                         (f2cl-lib:fref vr-%data%
                                        (ii ki)
                                        ((1 ldvr) (1 *))
                                        vr-%offset%))))
                    (dscal n remax
                     (f2cl-lib:array-slice vr
                                           double-float
                                           (1 ki)
                                           ((1 ldvr) (1 *)))
                     1))))
                (t
                 (cond
                   ((>=
                     (abs
                      (f2cl-lib:fref t$
                                     ((f2cl-lib:int-add ki
                                                        (f2cl-lib:int-sub 1))
                                      ki)
                                     ((1 ldt) (1 *))))
                     (abs
                      (f2cl-lib:fref t$
                                     (ki
                                      (f2cl-lib:int-add ki
                                                        (f2cl-lib:int-sub 1)))
                                     ((1 ldt) (1 *)))))
                    (setf (f2cl-lib:fref work-%data%
                                         ((f2cl-lib:int-add
                                           (f2cl-lib:int-sub ki 1)
                                           n))
                                         ((1 *))
                                         work-%offset%)
                            one)
                    (setf (f2cl-lib:fref work-%data%
                                         ((f2cl-lib:int-add ki n2))
                                         ((1 *))
                                         work-%offset%)
                            (/ wi
                               (f2cl-lib:fref t$-%data%
                                              ((f2cl-lib:int-sub ki 1) ki)
                                              ((1 ldt) (1 *))
                                              t$-%offset%))))
                   (t
                    (setf (f2cl-lib:fref work-%data%
                                         ((f2cl-lib:int-add
                                           (f2cl-lib:int-sub ki 1)
```

```
                                n))
                               ((1 *))
                               work-%offset%)
                    (/ (- wi)
                       (f2cl-lib:fref t$-%data%
                                      (ki (f2cl-lib:int-sub ki 1))
                                      ((1 ldt) (1 *))
                                      t$-%offset%)))
           (setf (f2cl-lib:fref work-%data%
                                ((f2cl-lib:int-add ki n2))
                                ((1 *))
                                work-%offset%)
                 one)))
        (setf (f2cl-lib:fref work-%data%
                             ((f2cl-lib:int-add ki n))
                             ((1 *))
                             work-%offset%)
              zero)
        (setf (f2cl-lib:fref work-%data%
                             ((f2cl-lib:int-add
                               (f2cl-lib:int-sub ki 1)
                               n2))
                             ((1 *))
                             work-%offset%)
              zero)
        (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                      ((> k
                          (f2cl-lib:int-add ki (f2cl-lib:int-sub 2)))
                       nil)
          (tagbody
            (setf (f2cl-lib:fref work-%data%
                                 ((f2cl-lib:int-add k n))
                                 ((1 *))
                                 work-%offset%)
                  (*
                   (-
                    (f2cl-lib:fref work-%data%
                                   ((f2cl-lib:int-add
                                     (f2cl-lib:int-sub ki 1)
                                     n))
                                   ((1 *))
                                   work-%offset%))
                    (f2cl-lib:fref t$-%data%
                                   (k (f2cl-lib:int-sub ki 1))
                                   ((1 ldt) (1 *))
                                   t$-%offset%)))
```

```
                    (setf (f2cl-lib:fref work-%data%
                                         ((f2cl-lib:int-add k n2))
                                         ((1 *))
                                         work-%offset%)
                          (*
                           (-
                            (f2cl-lib:fref work-%data%
                                           ((f2cl-lib:int-add ki n2))
                                           ((1 *))
                                           work-%offset%))
                            (f2cl-lib:fref t$-%data%
                                           (k ki)
                                           ((1 ldt) (1 *))
                                           t$-%offset%)))))
                  (setf jnxt (f2cl-lib:int-sub ki 2))
                  (f2cl-lib:fdo (j (f2cl-lib:int-add ki (f2cl-lib:int-sub 2))
                                 (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                                ((> j 1) nil)
                    (tagbody
                      (if (> j jnxt) (go label90))
                      (setf j1 j)
                      (setf j2 j)
                      (setf jnxt (f2cl-lib:int-sub j 1))
                      (cond
                        ((> j 1)
                         (cond
                           ((/=
                             (f2cl-lib:fref t$
                                            (j
                                             (f2cl-lib:int-add j
                                                               (f2cl-lib:int-sub
                                                                1)))
                                            ((1 ldt) (1 *)))
                             zero)
                            (setf j1 (f2cl-lib:int-sub j 1))
                            (setf jnxt (f2cl-lib:int-sub j 2))))))
                      (cond
                        ((= j1 j2)
                         (multiple-value-bind
                               (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                                var-8 var-9 var-10 var-11 var-12 var-13 var-14
                                var-15 var-16 var-17)
                             (dlaln2 nil 1 2 smin one
                              (f2cl-lib:array-slice t$
                                                    double-float
                                                    (j j)
```

```
                              ((1 ldt) (1 *)))
     ldt one one
     (f2cl-lib:array-slice work
                           double-float
                           ((+ j n))
                           ((1 *)))
     n wr wi x 2 scale xnorm ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10
                   var-11 var-12 var-13 var-14))
  (setf scale var-15)
  (setf xnorm var-16)
  (setf ierr var-17))
(cond
  ((> xnorm one)
   (cond
     ((> (f2cl-lib:fref work (j) ((1 *)))
         (f2cl-lib:f2cl/ bignum xnorm))
      (setf (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
            (/ (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
               xnorm))
      (setf (f2cl-lib:fref x (1 2) ((1 2) (1 2)))
            (/ (f2cl-lib:fref x (1 2) ((1 2) (1 2)))
               xnorm))
      (setf scale (/ scale xnorm))))))
(cond
  ((/= scale one)
   (dscal ki scale
    (f2cl-lib:array-slice work
                          double-float
                          ((+ 1 n))
                          ((1 *)))
    1)
   (dscal ki scale
    (f2cl-lib:array-slice work
                          double-float
                          ((+ 1 n2))
                          ((1 *)))
    1)))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j n))
                     ((1 *))
                     work-%offset%)
       (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j n2))
```

```
                        ((1 *))
                        work-%offset%)
               (f2cl-lib:fref x (1 2) ((1 2) (1 2))))
 (daxpy (f2cl-lib:int-sub j 1)
  (- (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
  (f2cl-lib:array-slice t$
                        double-float
                        (1 j)
                        ((1 ldt) (1 *)))
  1
  (f2cl-lib:array-slice work
                        double-float
                        ((+ 1 n))
                        ((1 *)))
  1)
 (daxpy (f2cl-lib:int-sub j 1)
  (- (f2cl-lib:fref x (1 2) ((1 2) (1 2))))
  (f2cl-lib:array-slice t$
                        double-float
                        (1 j)
                        ((1 ldt) (1 *)))
  1
  (f2cl-lib:array-slice work
                        double-float
                        ((+ 1 n2))
                        ((1 *)))
  1))
(t
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10 var-11 var-12 var-13 var-14
        var-15 var-16 var-17)
     (dlaln2 nil 2 2 smin one
      (f2cl-lib:array-slice t$
                            double-float
                            ((+ j (f2cl-lib:int-sub 1))
                             (f2cl-lib:int-sub j 1))
                            ((1 ldt) (1 *)))
      ldt one one
      (f2cl-lib:array-slice work
                            double-float
                            ((+ j
                                (f2cl-lib:int-sub 1)
                                n))
                            ((1 *)))
      n wr wi x 2 scale xnorm ierr)
```

```
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10
                   var-11 var-12 var-13 var-14))
  (setf scale var-15)
  (setf xnorm var-16)
  (setf ierr var-17))
(cond
  ((> xnorm one)
   (setf beta
           (max
            (f2cl-lib:fref work-%data%
                           ((f2cl-lib:int-sub j 1))
                           ((1 *))
                           work-%offset%)
            (f2cl-lib:fref work-%data%
                           (j)
                           ((1 *))
                           work-%offset%)))
   (cond
     ((> beta (f2cl-lib:f2cl/ bignum xnorm))
      (setf rec (/ one xnorm))
      (setf (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
              (* (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
                 rec))
      (setf (f2cl-lib:fref x (1 2) ((1 2) (1 2)))
              (* (f2cl-lib:fref x (1 2) ((1 2) (1 2)))
                 rec))
      (setf (f2cl-lib:fref x (2 1) ((1 2) (1 2)))
              (* (f2cl-lib:fref x (2 1) ((1 2) (1 2)))
                 rec))
      (setf (f2cl-lib:fref x (2 2) ((1 2) (1 2)))
              (* (f2cl-lib:fref x (2 2) ((1 2) (1 2)))
                 rec))
      (setf scale (* scale rec))))))
(cond
  ((/= scale one)
   (dscal ki scale
    (f2cl-lib:array-slice work
                          double-float
                          ((+ 1 n))
                          ((1 *)))
    1)
   (dscal ki scale
    (f2cl-lib:array-slice work
                          double-float
                          ((+ 1 n2))
```

```
                          ((1 *)))
    1)))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add
                       (f2cl-lib:int-sub j 1)
                       n))
                     ((1 *))
                     work-%offset%)
        (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j n))
                     ((1 *))
                     work-%offset%)
        (f2cl-lib:fref x (2 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add
                       (f2cl-lib:int-sub j 1)
                       n2))
                     ((1 *))
                     work-%offset%)
        (f2cl-lib:fref x (1 2) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j n2))
                     ((1 *))
                     work-%offset%)
        (f2cl-lib:fref x (2 2) ((1 2) (1 2))))
(daxpy (f2cl-lib:int-sub j 2)
 (- (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
 (f2cl-lib:array-slice t$
                       double-float
                       (1 (f2cl-lib:int-sub j 1))
                       ((1 ldt) (1 *)))
 1
 (f2cl-lib:array-slice work
                       double-float
                       ((+ 1 n))
                       ((1 *)))
 1)
(daxpy (f2cl-lib:int-sub j 2)
 (- (f2cl-lib:fref x (2 1) ((1 2) (1 2))))
 (f2cl-lib:array-slice t$
                       double-float
                       (1 j)
                       ((1 ldt) (1 *)))
 1
 (f2cl-lib:array-slice work
```

```
                                                double-float
                                                ((+ 1 n))
                                                ((1 *)))
                                    1)
                                   (daxpy (f2cl-lib:int-sub j 2)
                                    (- (f2cl-lib:fref x (1 2) ((1 2) (1 2))))
                                    (f2cl-lib:array-slice t$
                                                double-float
                                                (1 (f2cl-lib:int-sub j 1))
                                                ((1 ldt) (1 *)))
                                    1
                                    (f2cl-lib:array-slice work
                                                double-float
                                                ((+ 1 n2))
                                                ((1 *)))
                                    1)
                                   (daxpy (f2cl-lib:int-sub j 2)
                                    (- (f2cl-lib:fref x (2 2) ((1 2) (1 2))))
                                    (f2cl-lib:array-slice t$
                                                double-float
                                                (1 j)
                                                ((1 ldt) (1 *)))
                                    1
                                    (f2cl-lib:array-slice work
                                                double-float
                                                ((+ 1 n2))
                                                ((1 *)))
                                    1)))
  label90))
                        (cond
                          ((not over)
                           (dcopy ki
                            (f2cl-lib:array-slice work
                                                  double-float
                                                  ((+ 1 n))
                                                  ((1 *)))
                            1
                            (f2cl-lib:array-slice vr
                                                  double-float
                                                  (1 (f2cl-lib:int-sub is 1))
                                                  ((1 ldvr) (1 *)))
                            1)
                           (dcopy ki
                            (f2cl-lib:array-slice work
                                                  double-float
                                                  ((+ 1 n2))
```

```
                        ((1 *)))
 1
 (f2cl-lib:array-slice vr
                        double-float
                        (1 is)
                        ((1 ldvr) (1 *)))
 1)
(setf emax zero)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k ki) nil)
  (tagbody
    (setf emax
            (max emax
                 (+
                  (abs
                   (f2cl-lib:fref vr-%data%
                                  (k
                                   (f2cl-lib:int-sub is
                                                     1))
                                  ((1 ldvr) (1 *))
                                  vr-%offset%))
                  (abs
                   (f2cl-lib:fref vr-%data%
                                  (k is)
                                  ((1 ldvr) (1 *))
                                  vr-%offset%)))))))
(setf remax (/ one emax))
(dscal ki remax
 (f2cl-lib:array-slice vr
                        double-float
                        (1 (f2cl-lib:int-sub is 1))
                        ((1 ldvr) (1 *)))
 1)
(dscal ki remax
 (f2cl-lib:array-slice vr
                        double-float
                        (1 is)
                        ((1 ldvr) (1 *)))
 1)
(f2cl-lib:fdo (k (f2cl-lib:int-add ki 1)
                 (f2cl-lib:int-add k 1))
              ((> k n) nil)
  (tagbody
    (setf (f2cl-lib:fref vr-%data%
                         (k (f2cl-lib:int-sub is 1))
                         ((1 ldvr) (1 *))
```

```
                                vr-%offset%)
                        zero)
             (setf (f2cl-lib:fref vr-%data%
                                  (k is)
                                  ((1 ldvr) (1 *))
                                  vr-%offset%)
                        zero))))
        (t
         (cond
           ((> ki 2)
            (dgemv "N" n (f2cl-lib:int-sub ki 2) one vr ldvr
             (f2cl-lib:array-slice work
                                   double-float
                                   ((+ 1 n))
                                   ((1 *)))
             1
             (f2cl-lib:fref work-%data%
                            ((f2cl-lib:int-add
                              (f2cl-lib:int-sub ki 1)
                              n))
                            ((1 *))
                            work-%offset%)
             (f2cl-lib:array-slice vr
                                   double-float
                                   (1 (f2cl-lib:int-sub ki 1))
                                   ((1 ldvr) (1 *)))
             1)
            (dgemv "N" n (f2cl-lib:int-sub ki 2) one vr ldvr
             (f2cl-lib:array-slice work
                                   double-float
                                   ((+ 1 n2))
                                   ((1 *)))
             1
             (f2cl-lib:fref work-%data%
                            ((f2cl-lib:int-add ki n2))
                            ((1 *))
                            work-%offset%)
             (f2cl-lib:array-slice vr
                                   double-float
                                   (1 ki)
                                   ((1 ldvr) (1 *)))
             1))
           (t
            (dscal n
             (f2cl-lib:fref work-%data%
                            ((f2cl-lib:int-add
```

```
                        (f2cl-lib:int-sub ki 1)
                        n))
                      ((1 *))
                      work-%offset%)
    (f2cl-lib:array-slice vr
                          double-float
                          (1 (f2cl-lib:int-sub ki 1))
                          ((1 ldvr) (1 *)))
    1)
   (dscal n
    (f2cl-lib:fref work-%data%
                   ((f2cl-lib:int-add ki n2))
                   ((1 *))
                   work-%offset%)
    (f2cl-lib:array-slice vr
                          double-float
                          (1 ki)
                          ((1 ldvr) (1 *)))
    1)))
(setf emax zero)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k n) nil)
  (tagbody
    (setf emax
            (max emax
                 (+
                  (abs
                   (f2cl-lib:fref vr-%data%
                                  (k
                                   (f2cl-lib:int-sub ki
                                                     1))
                                  ((1 ldvr) (1 *))
                                  vr-%offset%))
                  (abs
                   (f2cl-lib:fref vr-%data%
                                  (k ki)
                                  ((1 ldvr) (1 *))
                                  vr-%offset%)))))))
(setf remax (/ one emax))
(dscal n remax
 (f2cl-lib:array-slice vr
                       double-float
                       (1 (f2cl-lib:int-sub ki 1))
                       ((1 ldvr) (1 *)))
 1)
(dscal n remax
```

```
                      (f2cl-lib:array-slice vr
                                            double-float
                                            (1 ki)
                                            ((1 ldvr) (1 *)))
                      1)))))
               (setf is (f2cl-lib:int-sub is 1))
               (if (/= ip 0) (setf is (f2cl-lib:int-sub is 1)))
  label130
               (if (= ip 1) (setf ip 0))
               (if (= ip -1) (setf ip 1))))))
       (cond
         (leftv
          (setf ip 0)
          (setf is 1)
          (f2cl-lib:fdo (ki 1 (f2cl-lib:int-add ki 1))
                        ((> ki n) nil)
            (tagbody
              (if (= ip -1) (go label250))
              (if (= ki n) (go label150))
              (if
               (=
                (f2cl-lib:fref t$-%data%
                               ((f2cl-lib:int-add ki 1) ki)
                               ((1 ldt) (1 *))
                               t$-%offset%)
                zero)
               (go label150))
              (setf ip 1)
  label150
              (cond
                (somev
                 (if
                  (not
                   (f2cl-lib:fref select-%data% (ki) ((1 *)) select-%offset%))
                  (go label250))))
              (setf wr
                      (f2cl-lib:fref t$-%data%
                                     (ki ki)
                                     ((1 ldt) (1 *))
                                     t$-%offset%))
              (setf wi zero)
              (if (/= ip 0)
                  (setf wi
                          (*
                           (f2cl-lib:fsqrt
                            (abs
```

```
                    (f2cl-lib:fref t$-%data%
                                  (ki (f2cl-lib:int-add ki 1))
                                  ((1 ldt) (1 *))
                                  t$-%offset%)))
                  (f2cl-lib:fsqrt
                   (abs
                    (f2cl-lib:fref t$-%data%
                                  ((f2cl-lib:int-add ki 1) ki)
                                  ((1 ldt) (1 *))
                                  t$-%offset%))))))
          (setf smin (max (* ulp (+ (abs wr) (abs wi))) smlnum))
          (cond
            ((= ip 0)
             (setf (f2cl-lib:fref work-%data%
                                  ((f2cl-lib:int-add ki n))
                                  ((1 *))
                                  work-%offset%)
                   one)
             (f2cl-lib:fdo (k (f2cl-lib:int-add ki 1)
                              (f2cl-lib:int-add k 1))
                           ((> k n) nil)
               (tagbody
                 (setf (f2cl-lib:fref work-%data%
                                      ((f2cl-lib:int-add k n))
                                      ((1 *))
                                      work-%offset%)
                       (-
                        (f2cl-lib:fref t$-%data%
                                       (ki k)
                                       ((1 ldt) (1 *))
                                       t$-%offset%)))))
             (setf vmax one)
             (setf vcrit bignum)
             (setf jnxt (f2cl-lib:int-add ki 1))
             (f2cl-lib:fdo (j (f2cl-lib:int-add ki 1)
                              (f2cl-lib:int-add j 1))
                           ((> j n) nil)
               (tagbody
                 (if (< j jnxt) (go label170))
                 (setf j1 j)
                 (setf j2 j)
                 (setf jnxt (f2cl-lib:int-add j 1))
                 (cond
                   ((< j n)
                    (cond
                      ((/=
```

```
        (f2cl-lib:fref t$
                       ((f2cl-lib:int-add j 1) j)
                       ((1 ldt) (1 *)))
        zero)
       (setf j2 (f2cl-lib:int-add j 1))
       (setf jnxt (f2cl-lib:int-add j 2))))))
(cond
  ((= j1 j2)
   (cond
     ((> (f2cl-lib:fref work (j) ((1 *))) vcrit)
      (setf rec (/ one vmax))
      (dscal (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
       rec
       (f2cl-lib:array-slice work
                             double-float
                             ((+ ki n))
                             ((1 *)))
       1)
      (setf vmax one)
      (setf vcrit bignum)))
   (setf (f2cl-lib:fref work-%data%
                        ((f2cl-lib:int-add j n))
                        ((1 *))
                        work-%offset%)
         (-
          (f2cl-lib:fref work-%data%
                         ((f2cl-lib:int-add j n))
                         ((1 *))
                         work-%offset%)
          (ddot (f2cl-lib:int-sub j ki 1)
           (f2cl-lib:array-slice t$
                                 double-float
                                 ((+ ki 1) j)
                                 ((1 ldt) (1 *)))
           1
           (f2cl-lib:array-slice work
                                 double-float
                                 ((+ ki 1 n))
                                 ((1 *)))
           1)))
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
          var-8 var-9 var-10 var-11 var-12 var-13 var-14
          var-15 var-16 var-17)
       (dlaln2 nil 1 1 smin one
        (f2cl-lib:array-slice t$
```

```
                                double-float
                                (j j)
                                ((1 ldt) (1 *)))
          ldt one one
          (f2cl-lib:array-slice work
                                double-float
                                ((+ j n))
                                ((1 *)))
          n wr zero x 2 scale xnorm ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8 var-9 var-10
                         var-11 var-12 var-13 var-14))
        (setf scale var-15)
        (setf xnorm var-16)
        (setf ierr var-17))
       (if (/= scale one)
           (dscal
            (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
            scale
            (f2cl-lib:array-slice work
                                  double-float
                                  ((+ ki n))
                                  ((1 *)))
            1))
       (setf (f2cl-lib:fref work-%data%
                            ((f2cl-lib:int-add j n))
                            ((1 *))
                            work-%offset%)
             (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
       (setf vmax
             (max
              (abs
               (f2cl-lib:fref work-%data%
                              ((f2cl-lib:int-add j n))
                              ((1 *))
                              work-%offset%))
              vmax))
       (setf vcrit (/ bignum vmax)))
      (t
       (setf beta
             (max
              (f2cl-lib:fref work-%data%
                             (j)
                             ((1 *))
                             work-%offset%)
              (f2cl-lib:fref work-%data%
```

```
                        ((f2cl-lib:int-add j 1))
                        ((1 *))
                        work-%offset%)))
(cond
  ((> beta vcrit)
   (setf rec (/ one vmax))
   (dscal (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
    rec
    (f2cl-lib:array-slice work
                          double-float
                          ((+ ki n))
                          ((1 *)))
    1)
   (setf vmax one)
   (setf vcrit bignum)))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j n))
                     ((1 *))
                     work-%offset%)
        (-
         (f2cl-lib:fref work-%data%
                        ((f2cl-lib:int-add j n))
                        ((1 *))
                        work-%offset%)
         (ddot (f2cl-lib:int-sub j ki 1)
          (f2cl-lib:array-slice t$
                                double-float
                                ((+ ki 1) j)
                                ((1 ldt) (1 *)))
          1
          (f2cl-lib:array-slice work
                                double-float
                                ((+ ki 1 n))
                                ((1 *)))
          1)))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j 1 n))
                     ((1 *))
                     work-%offset%)
        (-
         (f2cl-lib:fref work-%data%
                        ((f2cl-lib:int-add j 1 n))
                        ((1 *))
                        work-%offset%)
         (ddot (f2cl-lib:int-sub j ki 1)
          (f2cl-lib:array-slice t$
```

```
                                    double-float
                                    ((+ ki 1)
                                     (f2cl-lib:int-add j
                                                       1))
                                    ((1 ldt) (1 *)))
              1
              (f2cl-lib:array-slice work
                                    double-float
                                    ((+ ki 1 n))
                                    ((1 *)))
              1)))
        (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
               var-8 var-9 var-10 var-11 var-12 var-13 var-14
               var-15 var-16 var-17)
            (dlaln2 t 2 1 smin one
             (f2cl-lib:array-slice t$
                                   double-float
                                   (j j)
                                   ((1 ldt) (1 *)))
             ldt one one
             (f2cl-lib:array-slice work
                                   double-float
                                   ((+ j n))
                                   ((1 *)))
             n wr zero x 2 scale xnorm ierr)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                           var-6 var-7 var-8 var-9 var-10
                           var-11 var-12 var-13 var-14))
          (setf scale var-15)
          (setf xnorm var-16)
          (setf ierr var-17))
        (if (/= scale one)
            (dscal
             (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
             scale
             (f2cl-lib:array-slice work
                                   double-float
                                   ((+ ki n))
                                   ((1 *)))
             1))
        (setf (f2cl-lib:fref work-%data%
                             ((f2cl-lib:int-add j n))
                             ((1 *))
                             work-%offset%)
                (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
```

```
                        (setf (f2cl-lib:fref work-%data%
                                             ((f2cl-lib:int-add j 1 n))
                                             ((1 *))
                                             work-%offset%)
                              (f2cl-lib:fref x (2 1) ((1 2) (1 2))))
                        (setf vmax
                              (max
                               (abs
                                (f2cl-lib:fref work-%data%
                                               ((f2cl-lib:int-add j n))
                                               ((1 *))
                                               work-%offset%))
                               (abs
                                (f2cl-lib:fref work-%data%
                                               ((f2cl-lib:int-add j 1 n))
                                               ((1 *))
                                               work-%offset%))
                               vmax))
                        (setf vcrit (/ bignum vmax))))
  label170))
                  (cond
                    ((not over)
                     (dcopy (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
                      (f2cl-lib:array-slice work
                                            double-float
                                            ((+ ki n))
                                            ((1 *)))
                      1
                      (f2cl-lib:array-slice vl
                                            double-float
                                            (ki is)
                                            ((1 ldvl) (1 *)))
                      1)
                     (setf ii
                           (f2cl-lib:int-sub
                            (f2cl-lib:int-add
                             (idamax
                              (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
                              (f2cl-lib:array-slice vl
                                                    double-float
                                                    (ki is)
                                                    ((1 ldvl) (1 *)))
                              1)
                             ki)
                            1))
                     (setf remax
```

```
                (/ one
                   (abs
                    (f2cl-lib:fref vl-%data%
                                   (ii is)
                                   ((1 ldvl) (1 *))
                                   vl-%offset%))))
               (dscal (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1) remax
                (f2cl-lib:array-slice vl
                                      double-float
                                      (ki is)
                                      ((1 ldvl) (1 *)))
                1)
               (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                             ((> k
                                 (f2cl-lib:int-add ki
                                                   (f2cl-lib:int-sub 1)))
                              nil)
                 (tagbody
                   (setf (f2cl-lib:fref vl-%data%
                                        (k is)
                                        ((1 ldvl) (1 *))
                                        vl-%offset%)
                         zero))))
              (t
               (if (< ki n)
                   (dgemv "N" n (f2cl-lib:int-sub n ki) one
                    (f2cl-lib:array-slice vl
                                          double-float
                                          (1 (f2cl-lib:int-add ki 1))
                                          ((1 ldvl) (1 *)))
                    ldvl
                    (f2cl-lib:array-slice work
                                          double-float
                                          ((+ ki 1 n))
                                          ((1 *)))
                    1
                    (f2cl-lib:fref work-%data%
                                   ((f2cl-lib:int-add ki n))
                                   ((1 *))
                                   work-%offset%)
                    (f2cl-lib:array-slice vl
                                          double-float
                                          (1 ki)
                                          ((1 ldvl) (1 *)))
                    1))
               (setf ii
```

```
                (idamax n
                 (f2cl-lib:array-slice vl
                                       double-float
                                       (1 ki)
                                       ((1 ldvl) (1 *)))
                 1))
            (setf remax
                    (/ one
                       (abs
                        (f2cl-lib:fref vl-%data%
                                       (ii ki)
                                       ((1 ldvl) (1 *))
                                       vl-%offset%))))
            (dscal n remax
             (f2cl-lib:array-slice vl
                                   double-float
                                   (1 ki)
                                   ((1 ldvl) (1 *)))
             1))))
          (t
           (tagbody
             (cond
              ((>=
                (abs
                 (f2cl-lib:fref t$
                                (ki (f2cl-lib:int-add ki 1))
                                ((1 ldt) (1 *))))
                (abs
                 (f2cl-lib:fref t$
                                ((f2cl-lib:int-add ki 1) ki)
                                ((1 ldt) (1 *)))))
               (setf (f2cl-lib:fref work-%data%
                                    ((f2cl-lib:int-add ki n))
                                    ((1 *))
                                    work-%offset%)
                       (/ wi
                          (f2cl-lib:fref t$-%data%
                                         (ki (f2cl-lib:int-add ki 1))
                                         ((1 ldt) (1 *))
                                         t$-%offset%)))
               (setf (f2cl-lib:fref work-%data%
                                    ((f2cl-lib:int-add ki 1 n2))
                                    ((1 *))
                                    work-%offset%)
                       one))
              (t
```

```
                  (setf (f2cl-lib:fref work-%data%
                                       ((f2cl-lib:int-add ki n))
                                       ((1 *))
                                       work-%offset%)
                        one)
                  (setf (f2cl-lib:fref work-%data%
                                       ((f2cl-lib:int-add ki 1 n2))
                                       ((1 *))
                                       work-%offset%)
                        (/ (- wi)
                           (f2cl-lib:fref t$-%data%
                                          ((f2cl-lib:int-add ki 1) ki)
                                          ((1 ldt) (1 *))
                                          t$-%offset%)))))
            (setf (f2cl-lib:fref work-%data%
                                 ((f2cl-lib:int-add ki 1 n))
                                 ((1 *))
                                 work-%offset%)
                  zero)
            (setf (f2cl-lib:fref work-%data%
                                 ((f2cl-lib:int-add ki n2))
                                 ((1 *))
                                 work-%offset%)
                  zero)
            (f2cl-lib:fdo (k (f2cl-lib:int-add ki 2)
                             (f2cl-lib:int-add k 1))
                          ((> k n) nil)
              (tagbody
                (setf (f2cl-lib:fref work-%data%
                                     ((f2cl-lib:int-add k n))
                                     ((1 *))
                                     work-%offset%)
                      (*
                       (-
                        (f2cl-lib:fref work-%data%
                                       ((f2cl-lib:int-add ki n))
                                       ((1 *))
                                       work-%offset%))
                       (f2cl-lib:fref t$-%data%
                                      (ki k)
                                      ((1 ldt) (1 *))
                                      t$-%offset%)))
                (setf (f2cl-lib:fref work-%data%
                                     ((f2cl-lib:int-add k n2))
                                     ((1 *))
                                     work-%offset%)
```

```
                (*
                 (-
                  (f2cl-lib:fref work-%data%
                                 ((f2cl-lib:int-add ki 1 n2))
                                 ((1 *))
                                 work-%offset%))
                  (f2cl-lib:fref t$-%data%
                                 ((f2cl-lib:int-add ki 1) k)
                                 ((1 ldt) (1 *))
                                 t$-%offset%)))))
(setf vmax one)
(setf vcrit bignum)
(setf jnxt (f2cl-lib:int-add ki 2))
(f2cl-lib:fdo (j (f2cl-lib:int-add ki 2)
                 (f2cl-lib:int-add j 1))
              ((> j n) nil)
  (tagbody
    (if (< j jnxt) (go label200))
    (setf j1 j)
    (setf j2 j)
    (setf jnxt (f2cl-lib:int-add j 1))
    (cond
      ((< j n)
       (cond
         ((/=
           (f2cl-lib:fref t$
                          ((f2cl-lib:int-add j 1) j)
                          ((1 ldt) (1 *)))
           zero)
          (setf j2 (f2cl-lib:int-add j 1))
          (setf jnxt (f2cl-lib:int-add j 2))))))
    (cond
      ((= j1 j2)
       (cond
         ((> (f2cl-lib:fref work (j) ((1 *))) vcrit)
          (setf rec (/ one vmax))
          (dscal
           (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1) rec
           (f2cl-lib:array-slice work
                                 double-float
                                 ((+ ki n))
                                 ((1 *)))
           1)
          (dscal
           (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1) rec
           (f2cl-lib:array-slice work
```

```
                                double-float
                                ((+ ki n2))
                                ((1 *)))
     1)
    (setf vmax one)
    (setf vcrit bignum)))
 (setf (f2cl-lib:fref work-%data%
                      ((f2cl-lib:int-add j n))
                      ((1 *))
                      work-%offset%)
       (-
        (f2cl-lib:fref work-%data%
                       ((f2cl-lib:int-add j n))
                       ((1 *))
                       work-%offset%)
        (ddot (f2cl-lib:int-sub j ki 2)
         (f2cl-lib:array-slice t$
                               double-float
                               ((+ ki 2) j)
                               ((1 ldt) (1 *)))
         1
         (f2cl-lib:array-slice work
                               double-float
                               ((+ ki 2 n))
                               ((1 *)))
         1)))
 (setf (f2cl-lib:fref work-%data%
                      ((f2cl-lib:int-add j n2))
                      ((1 *))
                      work-%offset%)
       (-
        (f2cl-lib:fref work-%data%
                       ((f2cl-lib:int-add j n2))
                       ((1 *))
                       work-%offset%)
        (ddot (f2cl-lib:int-sub j ki 2)
         (f2cl-lib:array-slice t$
                               double-float
                               ((+ ki 2) j)
                               ((1 ldt) (1 *)))
         1
         (f2cl-lib:array-slice work
                               double-float
                               ((+ ki 2 n2))
                               ((1 *)))
         1)))
```

```
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6
       var-7 var-8 var-9 var-10 var-11 var-12 var-13
       var-14 var-15 var-16 var-17)
    (dlaln2 nil 1 2 smin one
     (f2cl-lib:array-slice t$
                           double-float
                           (j j)
                           ((1 ldt) (1 *)))
     ldt one one
     (f2cl-lib:array-slice work
                           double-float
                           ((+ j n))
                           ((1 *)))
     n wr (- wi) x 2 scale xnorm ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4
                   var-5 var-6 var-7 var-8 var-9
                   var-10 var-11 var-12 var-13
                   var-14))
  (setf scale var-15)
  (setf xnorm var-16)
  (setf ierr var-17))
(cond
  ((/= scale one)
   (dscal
    (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
    scale
    (f2cl-lib:array-slice work
                          double-float
                          ((+ ki n))
                          ((1 *)))
    1)
   (dscal
    (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
    scale
    (f2cl-lib:array-slice work
                          double-float
                          ((+ ki n2))
                          ((1 *)))
    1)))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j n))
                     ((1 *))
                     work-%offset%)
        (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
```

```
                        ((f2cl-lib:int-add j n2))
                        ((1 *))
                        work-%offset%)
                (f2cl-lib:fref x (1 2) ((1 2) (1 2))))
         (setf vmax
                (max
                 (abs
                  (f2cl-lib:fref work-%data%
                                 ((f2cl-lib:int-add j n))
                                 ((1 *))
                                 work-%offset%))
                 (abs
                  (f2cl-lib:fref work-%data%
                                 ((f2cl-lib:int-add j n2))
                                 ((1 *))
                                 work-%offset%))
                 vmax))
         (setf vcrit (/ bignum vmax)))
        (t
         (setf beta
                (max
                 (f2cl-lib:fref work-%data%
                                (j)
                                ((1 *))
                                work-%offset%)
                 (f2cl-lib:fref work-%data%
                                ((f2cl-lib:int-add j 1))
                                ((1 *))
                                work-%offset%)))
         (cond
           ((> beta vcrit)
            (setf rec (/ one vmax))
            (dscal
             (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1) rec
             (f2cl-lib:array-slice work
                                   double-float
                                   ((+ ki n))
                                   ((1 *)))
             1)
            (dscal
             (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1) rec
             (f2cl-lib:array-slice work
                                   double-float
                                   ((+ ki n2))
                                   ((1 *)))
             1)
```

```
   (setf vmax one)
   (setf vcrit bignum)))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j n))
                     ((1 *))
                     work-%offset%)
      (-
       (f2cl-lib:fref work-%data%
                      ((f2cl-lib:int-add j n))
                      ((1 *))
                      work-%offset%)
       (ddot (f2cl-lib:int-sub j ki 2)
        (f2cl-lib:array-slice t$
                              double-float
                              ((+ ki 2) j)
                              ((1 ldt) (1 *)))
        1
        (f2cl-lib:array-slice work
                              double-float
                              ((+ ki 2 n))
                              ((1 *)))
        1)))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j n2))
                     ((1 *))
                     work-%offset%)
      (-
       (f2cl-lib:fref work-%data%
                      ((f2cl-lib:int-add j n2))
                      ((1 *))
                      work-%offset%)
       (ddot (f2cl-lib:int-sub j ki 2)
        (f2cl-lib:array-slice t$
                              double-float
                              ((+ ki 2) j)
                              ((1 ldt) (1 *)))
        1
        (f2cl-lib:array-slice work
                              double-float
                              ((+ ki 2 n2))
                              ((1 *)))
        1)))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j 1 n))
                     ((1 *))
                     work-%offset%)
```

```
                                (-
                                 (f2cl-lib:fref work-%data%
                                                ((f2cl-lib:int-add j 1 n))
                                                ((1 *))
                                                work-%offset%)
                                 (ddot (f2cl-lib:int-sub j ki 2)
                                  (f2cl-lib:array-slice t$
                                                        double-float
                                                        ((+ ki 2)
                                                         (f2cl-lib:int-add j
                                                                           1))
                                                        ((1 ldt) (1 *)))
                                  1
                                  (f2cl-lib:array-slice work
                                                        double-float
                                                        ((+ ki 2 n))
                                                        ((1 *)))
                                  1)))
                           (setf (f2cl-lib:fref work-%data%
                                                ((f2cl-lib:int-add j 1 n2))
                                                ((1 *))
                                                work-%offset%)
                                 (-
                                  (f2cl-lib:fref work-%data%
                                                 ((f2cl-lib:int-add j 1 n2))
                                                 ((1 *))
                                                 work-%offset%)
                                  (ddot (f2cl-lib:int-sub j ki 2)
                                   (f2cl-lib:array-slice t$
                                                         double-float
                                                         ((+ ki 2)
                                                          (f2cl-lib:int-add j
                                                                            1))
                                                         ((1 ldt) (1 *)))
                                   1
                                   (f2cl-lib:array-slice work
                                                         double-float
                                                         ((+ ki 2 n2))
                                                         ((1 *)))
                                   1)))
                           (multiple-value-bind
                                 (var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                  var-7 var-8 var-9 var-10 var-11 var-12 var-13
                                  var-14 var-15 var-16 var-17)
                               (dlaln2 t 2 2 smin one
                                (f2cl-lib:array-slice t$
```

```
                              double-float
                              (j j)
                              ((1 ldt) (1 *)))
     ldt one one
     (f2cl-lib:array-slice work
                              double-float
                              ((+ j n))
                              ((1 *)))
     n wr (- wi) x 2 scale xnorm ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4
                   var-5 var-6 var-7 var-8 var-9
                   var-10 var-11 var-12 var-13
                   var-14))
  (setf scale var-15)
  (setf xnorm var-16)
  (setf ierr var-17))
(cond
  ((/= scale one)
   (dscal
    (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
    scale
    (f2cl-lib:array-slice work
                          double-float
                          ((+ ki n))
                          ((1 *)))
    1)
   (dscal
    (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
    scale
    (f2cl-lib:array-slice work
                          double-float
                          ((+ ki n2))
                          ((1 *)))
    1)))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j n))
                     ((1 *))
                     work-%offset%)
        (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j n2))
                     ((1 *))
                     work-%offset%)
        (f2cl-lib:fref x (1 2) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j 1 n))
```

```
                                        ((1 *))
                                        work-%offset%)
                                (f2cl-lib:fref x (2 1) ((1 2) (1 2))))
                            (setf (f2cl-lib:fref work-%data%
                                        ((f2cl-lib:int-add j 1 n2))
                                        ((1 *))
                                        work-%offset%)
                                (f2cl-lib:fref x (2 2) ((1 2) (1 2))))
                            (setf vmax
                                    (max
                                     (abs (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
                                     (abs (f2cl-lib:fref x (1 2) ((1 2) (1 2))))
                                     (abs (f2cl-lib:fref x (2 1) ((1 2) (1 2))))
                                     (abs (f2cl-lib:fref x (2 2) ((1 2) (1 2))))
                                     vmax))
                            (setf vcrit (/ bignum vmax))))
  label200))
  label210
                      (cond
                        ((not over)
                         (dcopy (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
                          (f2cl-lib:array-slice work
                                                double-float
                                                ((+ ki n))
                                                ((1 *)))
                          1
                          (f2cl-lib:array-slice vl
                                                double-float
                                                (ki is)
                                                ((1 ldvl) (1 *)))
                          1)
                         (dcopy (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
                          (f2cl-lib:array-slice work
                                                double-float
                                                ((+ ki n2))
                                                ((1 *)))
                          1
                          (f2cl-lib:array-slice vl
                                                double-float
                                                (ki (f2cl-lib:int-add is 1))
                                                ((1 ldvl) (1 *)))
                          1)
                         (setf emax zero)
                         (f2cl-lib:fdo (k ki (f2cl-lib:int-add k 1))
                                       ((> k n) nil)
                           (tagbody
```

```
    (setf emax
            (max emax
                 (+
                  (abs
                   (f2cl-lib:fref vl-%data%
                                  (k is)
                                  ((1 ldvl) (1 *))
                                  vl-%offset%))
                  (abs
                   (f2cl-lib:fref vl-%data%
                                  (k
                                   (f2cl-lib:int-add is
                                                     1))
                                  ((1 ldvl) (1 *))
                                  vl-%offset%)))))))
(setf remax (/ one emax))
(dscal (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
 remax
 (f2cl-lib:array-slice vl
                       double-float
                       (ki is)
                       ((1 ldvl) (1 *)))
 1)
(dscal (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
 remax
 (f2cl-lib:array-slice vl
                       double-float
                       (ki (f2cl-lib:int-add is 1))
                       ((1 ldvl) (1 *)))
 1)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k
                  (f2cl-lib:int-add ki
                                    (f2cl-lib:int-sub
                                     1)))
               nil)
  (tagbody
    (setf (f2cl-lib:fref vl-%data%
                         (k is)
                         ((1 ldvl) (1 *))
                         vl-%offset%)
          zero)
    (setf (f2cl-lib:fref vl-%data%
                         (k (f2cl-lib:int-add is 1))
                         ((1 ldvl) (1 *))
                         vl-%offset%)
```

```
                zero))))
          (t
           (cond
             ((< ki (f2cl-lib:int-add n (f2cl-lib:int-sub 1)))
              (dgemv "N" n (f2cl-lib:int-sub n ki 1) one
               (f2cl-lib:array-slice vl
                                     double-float
                                     (1 (f2cl-lib:int-add ki 2))
                                     ((1 ldvl) (1 *)))
               ldvl
               (f2cl-lib:array-slice work
                                     double-float
                                     ((+ ki 2 n))
                                     ((1 *)))
               1
               (f2cl-lib:fref work-%data%
                              ((f2cl-lib:int-add ki n))
                              ((1 *))
                              work-%offset%)
               (f2cl-lib:array-slice vl
                                     double-float
                                     (1 ki)
                                     ((1 ldvl) (1 *)))
               1)
              (dgemv "N" n (f2cl-lib:int-sub n ki 1) one
               (f2cl-lib:array-slice vl
                                     double-float
                                     (1 (f2cl-lib:int-add ki 2))
                                     ((1 ldvl) (1 *)))
               ldvl
               (f2cl-lib:array-slice work
                                     double-float
                                     ((+ ki 2 n2))
                                     ((1 *)))
               1
               (f2cl-lib:fref work-%data%
                              ((f2cl-lib:int-add ki 1 n2))
                              ((1 *))
                              work-%offset%)
               (f2cl-lib:array-slice vl
                                     double-float
                                     (1 (f2cl-lib:int-add ki 1))
                                     ((1 ldvl) (1 *)))
               1))
             (t
              (dscal n
```

```
    (f2cl-lib:fref work-%data%
                   ((f2cl-lib:int-add ki n))
                   ((1 *))
                   work-%offset%)
    (f2cl-lib:array-slice vl
                          double-float
                          (1 ki)
                          ((1 ldvl) (1 *)))
    1)
   (dscal n
    (f2cl-lib:fref work-%data%
                   ((f2cl-lib:int-add ki 1 n2))
                   ((1 *))
                   work-%offset%)
    (f2cl-lib:array-slice vl
                          double-float
                          (1 (f2cl-lib:int-add ki 1))
                          ((1 ldvl) (1 *)))
    1)))
(setf emax zero)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k n) nil)
  (tagbody
    (setf emax
            (max emax
                 (+
                  (abs
                   (f2cl-lib:fref vl-%data%
                                  (k ki)
                                  ((1 ldvl) (1 *))
                                  vl-%offset%))
                  (abs
                   (f2cl-lib:fref vl-%data%
                                  (k
                                   (f2cl-lib:int-add ki
                                                     1))
                                  ((1 ldvl) (1 *))
                                  vl-%offset%)))))))
(setf remax (/ one emax))
(dscal n remax
 (f2cl-lib:array-slice vl
                       double-float
                       (1 ki)
                       ((1 ldvl) (1 *)))
 1)
(dscal n remax
```

```
                        (f2cl-lib:array-slice vl
                                              double-float
                                              (1 (f2cl-lib:int-add ki 1))
                                              ((1 ldvl) (1 *)))
                        1))))))
                (setf is (f2cl-lib:int-add is 1))
                (if (/= ip 0) (setf is (f2cl-lib:int-add is 1)))
  label250
                (if (= ip -1) (setf ip 0))
                (if (= ip 1) (setf ip -1))))))
  end_label
         (return
          (values nil nil nil nil nil nil nil nil nil nil nil m nil info))))))
```

7.91 dtrexc LAPACK

⟨*dtrexc.input*⟩≡

```
)set break resume
)sys rm -f dtrexc.output
)spool dtrexc.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*dtrexc.help*⟩≡

```
==========================================================================
dtrexc examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

NAME
       DTREXC - the real Schur factorization of a real matrix A = Q*T*Q**T, so
       that the diagonal block of T with row index IFST is moved to row ILST

SYNOPSIS
       SUBROUTINE DTREXC( COMPQ, N, T, LDT, Q, LDQ, IFST, ILST, WORK, INFO )

           CHARACTER      COMPQ

           INTEGER        IFST, ILST, INFO, LDQ, LDT, N

           DOUBLE         PRECISION Q( LDQ, * ), T( LDT, * ), WORK( * )

PURPOSE
       DTREXC reorders the real Schur factorization  of  a  real  matrix  A  =
       Q*T*Q**T,  so that the diagonal block of T with row index IFST is moved
       to row ILST.

       The real Schur form T is reordered by an orthogonal  similarity  trans-
       formation  Z**T*T*Z,  and  optionally  the matrix Q of Schur vectors is
       updated by postmultiplying it with Z.

       T must be in Schur canonical form (as returned  by  DHSEQR),  that  is,
       block  upper  triangular  with  1-by-1 and 2-by-2 diagonal blocks; each
       2-by-2 diagonal block has its diagonal elements equal and its off-diag-
       onal elements of opposite sign.


ARGUMENTS
       COMPQ   (input) CHARACTER*1
               = 'V':  update the matrix Q of Schur vectors;
               = 'N':  do not update Q.

       N       (input) INTEGER
               The order of the matrix T. N >= 0.

       T       (input/output) DOUBLE PRECISION array, dimension (LDT,N)
```

On entry, the upper quasi-triangular matrix T, in Schur Schur canonical form. On exit, the reordered upper quasi-triangular matrix, again in Schur canonical form.

LDT (input) INTEGER
The leading dimension of the array T. LDT >= max(1,N).

Q (input/output) DOUBLE PRECISION array, dimension (LDQ,N)
On entry, if COMPQ = 'V', the matrix Q of Schur vectors. On exit, if COMPQ = 'V', Q has been postmultiplied by the orthogonal transformation matrix Z which reorders T. If COMPQ = 'N', Q is not referenced.

LDQ (input) INTEGER
The leading dimension of the array Q. LDQ >= max(1,N).

IFST (input/output) INTEGER
ILST (input/output) INTEGER Specify the reordering of the diagonal blocks of T. The block with row index IFST is moved to row ILST, by a sequence of transpositions between adjacent blocks. On exit, if IFST pointed on entry to the second row of a 2-by-2 block, it is changed to point to the first row; ILST always points to the first row of the block in its final position (which may differ from its input value by +1 or -1). 1 <= IFST <= N; 1 <= ILST <= N.

WORK (workspace) DOUBLE PRECISION array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value
= 1: two adjacent blocks were too close to swap (the problem is very ill-conditioned); T may have been partially reordered, and ILST points to the first row of the current position of the block being moved.

⟨*LAPACK dtrexc*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dtrexc (compq n t$ ldt q ldq ifst ilst work info)
    (declare (type (simple-array double-float (*)) work q t$)
             (type fixnum info ilst ifst ldq ldt n)
             (type character compq))
    (f2cl-lib:with-multi-array-data
        ((compq character compq-%data% compq-%offset%)
         (t$ double-float t$-%data% t$-%offset%)
         (q double-float q-%data% q-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((here 0) (nbf 0) (nbl 0) (nbnext 0) (wantq nil))
        (declare (type fixnum here nbf nbl nbnext)
                 (type (member t nil) wantq))
        (setf info 0)
        (setf wantq (char-equal compq #\V))
        (cond
          ((and (not wantq) (not (char-equal compq #\N)))
           (setf info -1))
          ((< n 0)
           (setf info -2))
          ((< ldt (max (the fixnum 1) (the fixnum n)))
           (setf info -4))
          ((or (< ldq 1)
               (and wantq
                    (< ldq
                       (max (the fixnum 1)
                            (the fixnum n)))))
           (setf info -6))
          ((or (< ifst 1) (> ifst n))
           (setf info -7))
          ((or (< ilst 1) (> ilst n))
           (setf info -8)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DTREXC" (f2cl-lib:int-sub info))
           (go end_label)))
        (if (<= n 1) (go end_label))
        (cond
          ((> ifst 1)
           (if
            (/=
             (f2cl-lib:fref t$-%data%
```

```
                    (ifst (f2cl-lib:int-sub ifst 1))
                    ((1 ldt) (1 *))
                    t$-%offset%)
     zero)
    (setf ifst (f2cl-lib:int-sub ifst 1)))))
(setf nbf 1)
(cond
  ((< ifst n)
   (if
    (/=
     (f2cl-lib:fref t$-%data%
                    ((f2cl-lib:int-add ifst 1) ifst)
                    ((1 ldt) (1 *))
                    t$-%offset%)
     zero)
    (setf nbf 2))))
(cond
  ((> ilst 1)
   (if
    (/=
     (f2cl-lib:fref t$-%data%
                    (ilst (f2cl-lib:int-sub ilst 1))
                    ((1 ldt) (1 *))
                    t$-%offset%)
     zero)
    (setf ilst (f2cl-lib:int-sub ilst 1)))))
(setf nbl 1)
(cond
  ((< ilst n)
   (if
    (/=
     (f2cl-lib:fref t$-%data%
                    ((f2cl-lib:int-add ilst 1) ilst)
                    ((1 ldt) (1 *))
                    t$-%offset%)
     zero)
    (setf nbl 2))))
(if (= ifst ilst) (go end_label))
(cond
  ((< ifst ilst)
   (tagbody
     (if (and (= nbf 2) (= nbl 1))
         (setf ilst (f2cl-lib:int-sub ilst 1)))
     (if (and (= nbf 1) (= nbl 2))
         (setf ilst (f2cl-lib:int-add ilst 1)))
     (setf here ifst)
```

```
label10
            (cond
              ((or (= nbf 1) (= nbf 2))
               (setf nbnext 1)
               (cond
                 ((<= (f2cl-lib:int-add here nbf 1) n)
                  (if
                   (/=
                    (f2cl-lib:fref t$-%data%
                                   ((f2cl-lib:int-add here nbf 1)
                                    (f2cl-lib:int-add here nbf))
                                   ((1 ldt) (1 *))
                                   t$-%offset%)
                    zero)
                   (setf nbnext 2))))
               (multiple-value-bind
                     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                      var-9 var-10)
                   (dlaexc wantq n t$ ldt q ldq here nbf nbnext work info)
                 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                  var-7 var-8 var-9))
                 (setf info var-10))
               (cond
                 ((/= info 0)
                  (setf ilst here)
                  (go end_label)))
               (setf here (f2cl-lib:int-add here nbnext))
               (cond
                 ((= nbf 2)
                  (if
                   (=
                    (f2cl-lib:fref t$-%data%
                                   ((f2cl-lib:int-add here 1) here)
                                   ((1 ldt) (1 *))
                                   t$-%offset%)
                    zero)
                   (setf nbf 3)))))
              (t
               (setf nbnext 1)
               (cond
                 ((<= (f2cl-lib:int-add here 3) n)
                  (if
                   (/=
                    (f2cl-lib:fref t$-%data%
                                   ((f2cl-lib:int-add here 3)
                                    (f2cl-lib:int-add here 2))
```

```
                    ((1 ldt) (1 *))
                    t$-%offset%)
     zero)
    (setf nbnext 2))))
(multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10)
    (dlaexc wantq n t$ ldt q ldq (f2cl-lib:int-add here 1) 1
     nbnext work info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9))
  (setf info var-10))
(cond
  ((/= info 0)
   (setf ilst here)
   (go end_label)))
(cond
  ((= nbnext 1)
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
          var-9 var-10)
       (dlaexc wantq n t$ ldt q ldq here 1 nbnext work info)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7 var-8 var-9))
     (setf info var-10))
   (setf here (f2cl-lib:int-add here 1)))
  (t
   (if
    (=
     (f2cl-lib:fref t$-%data%
                    ((f2cl-lib:int-add here 2)
                     (f2cl-lib:int-add here 1))
                    ((1 ldt) (1 *))
                    t$-%offset%)
     zero)
    (setf nbnext 1))
   (cond
     ((= nbnext 2)
      (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
             var-8 var-9 var-10)
          (dlaexc wantq n t$ ldt q ldq here 1 nbnext work info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                         var-6 var-7 var-8 var-9))
        (setf info var-10))
      (cond
```

```
                        ((/= info 0)
                         (setf ilst here)
                         (go end_label)))
                       (setf here (f2cl-lib:int-add here 2)))
                      (t
                       (multiple-value-bind
                             (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                              var-8 var-9 var-10)
                           (dlaexc wantq n t$ ldt q ldq here 1 1 work info)
                         (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                                          var-6 var-7 var-8 var-9))
                         (setf info var-10))
                       (multiple-value-bind
                             (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                              var-8 var-9 var-10)
                           (dlaexc wantq n t$ ldt q ldq
                            (f2cl-lib:int-add here 1) 1 1 work info)
                         (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                                          var-6 var-7 var-8 var-9))
                         (setf info var-10))
                       (setf here (f2cl-lib:int-add here 2))))))))
              (if (< here ilst) (go label10))))
          (t
           (tagbody
             (setf here ifst)
label20
             (cond
               ((or (= nbf 1) (= nbf 2))
                (setf nbnext 1)
                (cond
                  ((>= here 3)
                   (if
                    (/=
                     (f2cl-lib:fref t$-%data%
                                    ((f2cl-lib:int-sub here 1)
                                     (f2cl-lib:int-sub here 2))
                                    ((1 ldt) (1 *))
                                    t$-%offset%)
                     zero)
                    (setf nbnext 2))))
                (multiple-value-bind
                      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                       var-9 var-10)
                    (dlaexc wantq n t$ ldt q ldq (f2cl-lib:int-sub here nbnext)
                     nbnext nbf work info)
                  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
```

```
                    var-7 var-8 var-9))
  (setf info var-10))
 (cond
   ((/= info 0)
    (setf ilst here)
    (go end_label)))
 (setf here (f2cl-lib:int-sub here nbnext))
 (cond
   ((= nbf 2)
    (if
     (=
      (f2cl-lib:fref t$-%data%
                     ((f2cl-lib:int-add here 1) here)
                     ((1 ldt) (1 *))
                     t$-%offset%)
      zero)
     (setf nbf 3)))))
(t
 (setf nbnext 1)
 (cond
   ((>= here 3)
    (if
     (/=
      (f2cl-lib:fref t$-%data%
                     ((f2cl-lib:int-sub here 1)
                      (f2cl-lib:int-sub here 2))
                     ((1 ldt) (1 *))
                     t$-%offset%)
      zero)
     (setf nbnext 2))))
 (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10)
     (dlaexc wantq n t$ ldt q ldq (f2cl-lib:int-sub here nbnext)
      nbnext 1 work info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9))
   (setf info var-10))
 (cond
   ((/= info 0)
    (setf ilst here)
    (go end_label)))
 (cond
   ((= nbnext 1)
    (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
```

```
        var-9 var-10)
      (dlaexc wantq n t$ ldt q ldq here nbnext 1 work info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                     var-7 var-8 var-9))
    (setf info var-10))
  (setf here (f2cl-lib:int-sub here 1)))
 (t
  (if
   (=
    (f2cl-lib:fref t$-%data%
                   (here (f2cl-lib:int-sub here 1))
                   ((1 ldt) (1 *))
                   t$-%offset%)
    zero)
   (setf nbnext 1))
  (cond
    ((= nbnext 2)
     (multiple-value-bind
           (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
            var-8 var-9 var-10)
         (dlaexc wantq n t$ ldt q ldq
          (f2cl-lib:int-sub here 1) 2 1 work info)
       (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                        var-6 var-7 var-8 var-9))
       (setf info var-10))
     (cond
       ((/= info 0)
        (setf ilst here)
        (go end_label)))
     (setf here (f2cl-lib:int-sub here 2)))
    (t
     (multiple-value-bind
           (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
            var-8 var-9 var-10)
         (dlaexc wantq n t$ ldt q ldq here 1 1 work info)
       (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                        var-6 var-7 var-8 var-9))
       (setf info var-10))
     (multiple-value-bind
           (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
            var-8 var-9 var-10)
         (dlaexc wantq n t$ ldt q ldq
          (f2cl-lib:int-sub here 1) 1 1 work info)
       (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                        var-6 var-7 var-8 var-9))
       (setf info var-10))
```

```
                        (setf here (f2cl-lib:int-sub here 2))))))))
               (if (> here ilst) (go label20)))))
         (setf ilst here)
  end_label
         (return (values nil nil nil nil nil nil ifst ilst nil info))))))
```

7.92 dtrsna LAPACK

⟨*dtrsna.input*⟩≡

```
  )set break resume
  )sys rm -f dtrsna.output
  )spool dtrsna.output
  )set message test on
  )set message auto off
  )clear all

  )spool
  )lisp (bye)
```

⟨*dtrsna.help*⟩≡

```
======================================================================
dtrsna examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       DTRSNA  - reciprocal condition numbers for specified eigenvalues and/or
       right eigenvectors of a real upper quasi-triangular matrix T (or of any
       matrix Q*T*Q**T with Q orthogonal)

SYNOPSIS
       SUBROUTINE DTRSNA( JOB,  HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR,
                          S, SEP, MM, M, WORK, LDWORK, IWORK, INFO )

           CHARACTER      HOWMNY, JOB

           INTEGER        INFO, LDT, LDVL, LDVR, LDWORK, M, MM, N

           LOGICAL        SELECT( * )

           INTEGER        IWORK( * )

           DOUBLE         PRECISION S( * ), SEP( * ), T( LDT, * ), VL( LDVL, *
                          ), VR( LDVR, * ), WORK( LDWORK, * )

PURPOSE
       DTRSNA estimates reciprocal condition numbers for specified eigenvalues
       and/or right eigenvectors of a real upper quasi-triangular matrix T (or
       of any matrix Q*T*Q**T with Q orthogonal).

       T  must  be  in  Schur canonical form (as returned by DHSEQR), that is,
       block upper triangular with 1-by-1 and  2-by-2  diagonal  blocks;  each
       2-by-2 diagonal block has its diagonal elements equal and its off-diag-
       onal elements of opposite sign.


ARGUMENTS
       JOB     (input) CHARACTER*1
               Specifies whether condition numbers are required for  eigenval-
               ues (S) or eigenvectors (SEP):
               = 'E': for eigenvalues only (S);
               = 'V': for eigenvectors only (SEP);
```

= 'B': for both eigenvalues and eigenvectors (S and SEP).

HOWMNY (input) CHARACTER*1
= 'A': compute condition numbers for all eigenpairs;
= 'S': compute condition numbers for selected eigenpairs specified by the array SELECT.

SELECT (input) LOGICAL array, dimension (N)
If HOWMNY = 'S', SELECT specifies the eigenpairs for which condition numbers are required. To select condition numbers for the eigenpair corresponding to a real eigenvalue w(j), SELECT(j) must be set to .TRUE.. To select condition numbers corresponding to a complex conjugate pair of eigenvalues w(j) and w(j+1), either SELECT(j) or SELECT(j+1) or both, must be set to .TRUE.. If HOWMNY = 'A', SELECT is not referenced.

N (input) INTEGER
The order of the matrix T. N >= 0.

T (input) DOUBLE PRECISION array, dimension (LDT,N)
The upper quasi-triangular matrix T, in Schur canonical form.

LDT (input) INTEGER
The leading dimension of the array T. LDT >= max(1,N).

VL (input) DOUBLE PRECISION array, dimension (LDVL,M)
If JOB = 'E' or 'B', VL must contain left eigenvectors of T (or of any Q*T*Q**T with Q orthogonal), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VL, as returned by DHSEIN or DTREVC. If JOB = 'V', VL is not referenced.

LDVL (input) INTEGER
The leading dimension of the array VL. LDVL >= 1; and if JOB = 'E' or 'B', LDVL >= N.

VR (input) DOUBLE PRECISION array, dimension (LDVR,M)
If JOB = 'E' or 'B', VR must contain right eigenvectors of T (or of any Q*T*Q**T with Q orthogonal), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VR, as returned by DHSEIN or DTREVC. If JOB = 'V', VR is not referenced.

LDVR (input) INTEGER
The leading dimension of the array VR. LDVR >= 1; and if JOB = 'E' or 'B', LDVR >= N.

```
        S       (output) DOUBLE PRECISION array, dimension (MM)
                If JOB = 'E' or 'B', the reciprocal condition numbers of the
                selected eigenvalues, stored in consecutive elements of the
                array. For a complex conjugate pair of eigenvalues two consecu-
                tive elements of S are set to the same value. Thus S(j),
                SEP(j), and the j-th columns of VL and VR all correspond to the
                same eigenpair (but not in general the j-th eigenpair, unless
                all eigenpairs are selected). If JOB = 'V', S is not refer-
                enced.

        SEP     (output) DOUBLE PRECISION array, dimension (MM)
                If JOB = 'V' or 'B', the estimated reciprocal condition numbers
                of the selected eigenvectors, stored in consecutive elements of
                the array. For a complex eigenvector two consecutive elements
                of SEP are set to the same value. If the eigenvalues cannot be
                reordered to compute SEP(j), SEP(j) is set to 0; this can only
                occur when the true value would be very small anyway. If JOB =
                'E', SEP is not referenced.

        MM      (input) INTEGER
                The number of elements in the arrays S (if JOB = 'E' or 'B')
                and/or SEP (if JOB = 'V' or 'B'). MM >= M.

        M       (output) INTEGER
                The number of elements of the arrays S and/or SEP actually used
                to store the estimated condition numbers. If HOWMNY = 'A', M
                is set to N.

        WORK    (workspace) DOUBLE PRECISION array, dimension (LDWORK,N+6)
                If JOB = 'E', WORK is not referenced.

        LDWORK  (input) INTEGER
                The leading dimension of the array WORK. LDWORK >= 1; and if
                JOB = 'V' or 'B', LDWORK >= N.

        IWORK   (workspace) INTEGER array, dimension (2*(N-1))
                If JOB = 'E', IWORK is not referenced.

        INFO    (output) INTEGER
                = 0: successful exit
                < 0: if INFO = -i, the i-th argument had an illegal value

    FURTHER DETAILS
            The reciprocal of the condition number of an eigenvalue lambda is
            defined as
```

```
        S(lambda) = |v'*u| / (norm(u)*norm(v))
```

where u and v are the right and left eigenvectors of T corresponding to lambda; v' denotes the conjugate-transpose of v, and norm(u) denotes the Euclidean norm. These reciprocal condition numbers always lie between zero (very badly conditioned) and one (very well conditioned). If n = 1, S(lambda) is defined to be 1.

An approximate error bound for a computed eigenvalue W(i) is given by

```
                    EPS * norm(T) / S(i)
```

where EPS is the machine precision.

The reciprocal of the condition number of the right eigenvector u corresponding to lambda is defined as follows. Suppose

```
            T = ( lambda  c  )
                (   0    T22 )
```

Then the reciprocal condition number is

```
        SEP( lambda, T22 ) = sigma-min( T22 - lambda*I )
```

where sigma-min denotes the smallest singular value. We approximate the smallest singular value by the reciprocal of an estimate of the one-norm of the inverse of T22 - lambda*I. If n = 1, SEP(1) is defined to be abs(T(1,1)).

An approximate error bound for a computed right eigenvector VR(i) is given by

```
                    EPS * norm(T) / SEP(i)
```

⟨*LAPACK dtrsna*⟩≡

```
(let* ((zero 0.0) (one 1.0) (two 2.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two))
  (defun dtrsna
         (job howmny select n t$ ldt vl ldvl vr ldvr s sep mm m work ldwork
          iwork info)
    (declare (type (simple-array fixnum (*)) iwork)
             (type (simple-array double-float (*)) work sep s vr vl t$)
             (type fixnum info ldwork m mm ldvr ldvl ldt n)
             (type (simple-array (member t nil) (*)) select)
             (type character howmny job))
    (f2cl-lib:with-multi-array-data
        ((job character job-%data% job-%offset%)
         (howmny character howmny-%data% howmny-%offset%)
         (select (member t nil) select-%data% select-%offset%)
         (t$ double-float t$-%data% t$-%offset%)
         (vl double-float vl-%data% vl-%offset%)
         (vr double-float vr-%data% vr-%offset%)
         (s double-float s-%data% s-%offset%)
         (sep double-float sep-%data% sep-%offset%)
         (work double-float work-%data% work-%offset%)
         (iwork fixnum iwork-%data% iwork-%offset%))
      (prog ((dummy (make-array 1 :element-type 'double-float)) (bignum 0.0)
             (cond$ 0.0) (cs 0.0) (delta 0.0) (dumm 0.0) (eps 0.0) (est 0.0)
             (lnrm 0.0) (mu 0.0) (prod 0.0) (prod1 0.0) (prod2 0.0) (rnrm 0.0)
             (scale 0.0) (smlnum 0.0) (sn 0.0) (i 0) (ierr 0) (ifst 0) (ilst 0)
             (j 0) (k 0) (kase 0) (ks 0) (n2 0) (nn 0) (pair nil) (somcon nil)
             (wantbh nil) (wants nil) (wantsp nil) (/=$ 0.0f0))
        (declare (type (single-float) /=$)
                 (type (simple-array double-float (1)) dummy)
                 (type (double-float) bignum cond$ cs delta dumm eps est lnrm
                                      mu prod prod1 prod2 rnrm scale smlnum sn)
                 (type fixnum i ierr ifst ilst j k kase ks n2 nn)
                 (type (member t nil) pair somcon wantbh wants wantsp))
        (setf wantbh (char-equal job #\B))
        (setf wants (or (char-equal job #\E) wantbh))
        (setf wantsp (or (char-equal job #\V) wantbh))
        (setf somcon (char-equal howmny #\S))
        (setf info 0)
        (cond
          ((and (not wants) (not wantsp))
           (setf info -1))
          ((and (not (char-equal howmny #\A)) (not somcon))
           (setf info -2))
```

```
((< n 0)
 (setf info -4))
((< ldt (max (the fixnum 1) (the fixnum n)))
 (setf info -6))
((or (< ldvl 1) (and wants (< ldvl n)))
 (setf info -8))
((or (< ldvr 1) (and wants (< ldvr n)))
 (setf info -10))
(t
 (cond
   (somcon
    (setf m 0)
    (setf pair nil)
    (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                  ((> k n) nil)
      (tagbody
        (cond
          (pair
           (setf pair nil))
          (t
           (cond
             ((< k n)
              (cond
                ((=
                  (f2cl-lib:fref t$
                                 ((f2cl-lib:int-add k 1) k)
                                 ((1 ldt) (1 *)))
                  zero)
                 (if
                  (f2cl-lib:fref select-%data%
                                 (k)
                                 ((1 *))
                                 select-%offset%)
                  (setf m (f2cl-lib:int-add m 1))))
                (t
                 (setf pair t)
                 (if
                  (or
                   (f2cl-lib:fref select-%data%
                                  (k)
                                  ((1 *))
                                  select-%offset%)
                   (f2cl-lib:fref select-%data%
                                  ((f2cl-lib:int-add k 1))
                                  ((1 *))
                                  select-%offset%))
```

```
                        (setf m (f2cl-lib:int-add m 2))))))
                   (t
                    (if
                     (f2cl-lib:fref select-%data%
                                    (n)
                                    ((1 *))
                                    select-%offset%)
                     (setf m (f2cl-lib:int-add m 1)))))))))
       (t
        (setf m n)))
     (cond
       ((< mm m)
        (setf info -13))
       ((or (< ldwork 1) (and wantsp (< ldwork n)))
        (setf info -16)))))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DTRSNA" (f2cl-lib:int-sub info))
   (go end_label)))
(if (= n 0) (go end_label))
(cond
  ((= n 1)
   (cond
     (somcon
      (if
       (not (f2cl-lib:fref select-%data% (1) ((1 *)) select-%offset%))
       (go end_label))))
   (if wants (setf (f2cl-lib:fref s-%data% (1) ((1 *)) s-%offset%) one))
   (if wantsp
       (setf (f2cl-lib:fref sep-%data% (1) ((1 *)) sep-%offset%)
               (abs
                (f2cl-lib:fref t$-%data%
                               (1 1)
                               ((1 ldt) (1 *))
                               t$-%offset%))))
   (go end_label)))
(setf eps (dlamch "P"))
(setf smlnum (/ (dlamch "S") eps))
(setf bignum (/ one smlnum))
(multiple-value-bind (var-0 var-1)
    (dlabad smlnum bignum)
  (declare (ignore))
  (setf smlnum var-0)
  (setf bignum var-1))
```

```
(setf ks 0)
(setf pair nil)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k n) nil)
  (tagbody
    (cond
      (pair
       (setf pair nil)
       (go label60))
      (t
       (if (< k n)
           (setf pair
                   (coerce
                    (/=
                     (f2cl-lib:fref t$-%data%
                                    ((f2cl-lib:int-add k 1) k)
                                    ((1 ldt) (1 *))
                                    t$-%offset%)
                     zero)
                    '(member t nil))))))
    (cond
      (somcon
       (cond
         (pair
          (if
           (and
            (not
             (f2cl-lib:fref select-%data% (k) ((1 *)) select-%offset%))
            (not
             (f2cl-lib:fref select-%data%
                            ((f2cl-lib:int-add k 1))
                            ((1 *))
                            select-%offset%)))
           (go label60)))
         (t
          (if
           (not
            (f2cl-lib:fref select-%data% (k) ((1 *)) select-%offset%))
           (go label60))))))
    (setf ks (f2cl-lib:int-add ks 1))
    (cond
      (wants
       (cond
         ((not pair)
          (setf prod
                  (ddot n
```

```
                (f2cl-lib:array-slice vr
                                      double-float
                                      (1 ks)
                                      ((1 ldvr) (1 *)))
                1
                (f2cl-lib:array-slice vl
                                      double-float
                                      (1 ks)
                                      ((1 ldvl) (1 *)))
                1))
         (setf rnrm
                (dnrm2 n
                 (f2cl-lib:array-slice vr
                                       double-float
                                       (1 ks)
                                       ((1 ldvr) (1 *)))
                 1))
         (setf lnrm
                (dnrm2 n
                 (f2cl-lib:array-slice vl
                                       double-float
                                       (1 ks)
                                       ((1 ldvl) (1 *)))
                 1))
         (setf (f2cl-lib:fref s-%data% (ks) ((1 *)) s-%offset%)
                (/ (abs prod) (* rnrm lnrm))))
        (t
         (setf prod1
                (ddot n
                 (f2cl-lib:array-slice vr
                                       double-float
                                       (1 ks)
                                       ((1 ldvr) (1 *)))
                 1
                 (f2cl-lib:array-slice vl
                                       double-float
                                       (1 ks)
                                       ((1 ldvl) (1 *)))
                 1))
         (setf prod1
                (+ prod1
                   (ddot n
                    (f2cl-lib:array-slice vr
                                          double-float
                                          (1 (f2cl-lib:int-add ks 1))
                                          ((1 ldvr) (1 *)))
```

```
                  1
                  (f2cl-lib:array-slice vl
                                        double-float
                                        (1 (f2cl-lib:int-add ks 1))
                                        ((1 ldvl) (1 *)))
                  1)))
           (setf prod2
                 (ddot n
                  (f2cl-lib:array-slice vl
                                        double-float
                                        (1 ks)
                                        ((1 ldvl) (1 *)))
                  1
                  (f2cl-lib:array-slice vr
                                        double-float
                                        (1 (f2cl-lib:int-add ks 1))
                                        ((1 ldvr) (1 *)))
                  1))
           (setf prod2
                 (- prod2
                    (ddot n
                     (f2cl-lib:array-slice vl
                                           double-float
                                           (1 (f2cl-lib:int-add ks 1))
                                           ((1 ldvl) (1 *)))
                     1
                     (f2cl-lib:array-slice vr
                                           double-float
                                           (1 ks)
                                           ((1 ldvr) (1 *)))
                     1)))
           (setf rnrm
                 (dlapy2
                  (dnrm2 n
                   (f2cl-lib:array-slice vr
                                         double-float
                                         (1 ks)
                                         ((1 ldvr) (1 *)))
                   1)
                  (dnrm2 n
                   (f2cl-lib:array-slice vr
                                         double-float
                                         (1 (f2cl-lib:int-add ks 1))
                                         ((1 ldvr) (1 *)))
                   1)))
           (setf lnrm
```

```
                    (dlapy2
                     (dnrm2 n
                      (f2cl-lib:array-slice vl
                                            double-float
                                            (1 ks)
                                            ((1 ldvl) (1 *)))
                      1)
                     (dnrm2 n
                      (f2cl-lib:array-slice vl
                                            double-float
                                            (1 (f2cl-lib:int-add ks 1))
                                            ((1 ldvl) (1 *)))
                      1)))
               (setf cond$ (/ (dlapy2 prod1 prod2) (* rnrm lnrm)))
               (setf (f2cl-lib:fref s-%data% (ks) ((1 *)) s-%offset%) cond$)
               (setf (f2cl-lib:fref s-%data%
                                    ((f2cl-lib:int-add ks 1))
                                    ((1 *))
                                    s-%offset%)
                     cond$)))))
        (cond
          (wantsp
           (dlacpy "Full" n n t$ ldt work ldwork)
           (setf ifst k)
           (setf ilst 1)
           (multiple-value-bind
                 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                  var-9)
               (dtrexc "No Q" n work ldwork dummy 1 ifst ilst
                (f2cl-lib:array-slice work
                                      double-float
                                      (1 (f2cl-lib:int-add n 1))
                                      ((1 ldwork) (1 *)))
                ierr)
             (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-8))
             (setf ifst var-6)
             (setf ilst var-7)
             (setf ierr var-9))
           (cond
             ((or (= ierr 1) (= ierr 2))
              (setf scale one)
              (setf est bignum))
             (t
              (tagbody
                (cond
                  ((= (f2cl-lib:fref work (2 1) ((1 ldwork) (1 *))) zero)
```

```
(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
              ((> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data%
                         (i i)
                         ((1 ldwork) (1 *))
                         work-%offset%)
            (-
             (f2cl-lib:fref work-%data%
                            (i i)
                            ((1 ldwork) (1 *))
                            work-%offset%)
             (f2cl-lib:fref work-%data%
                            (1 1)
                            ((1 ldwork) (1 *))
                            work-%offset%)))))
 (setf n2 1)
 (setf nn (f2cl-lib:int-sub n 1)))
(t
 (setf mu
       (*
        (f2cl-lib:fsqrt
         (abs
          (f2cl-lib:fref work-%data%
                         (1 2)
                         ((1 ldwork) (1 *))
                         work-%offset%)))
        (f2cl-lib:fsqrt
         (abs
          (f2cl-lib:fref work-%data%
                         (2 1)
                         ((1 ldwork) (1 *))
                         work-%offset%)))))
 (setf delta
       (dlapy2 mu
        (f2cl-lib:fref work-%data%
                       (2 1)
                       ((1 ldwork) (1 *))
                       work-%offset%)))
 (setf cs (/ mu delta))
 (setf sn
       (/
        (-
         (f2cl-lib:fref work-%data%
                        (2 1)
                        ((1 ldwork) (1 *))
```

```
                          work-%offset%))
         delta))
(f2cl-lib:fdo (j 3 (f2cl-lib:int-add j 1))
              ((> j n) nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data%
                         (2 j)
                         ((1 ldwork) (1 *))
                         work-%offset%)
            (* cs
               (f2cl-lib:fref work-%data%
                              (2 j)
                              ((1 ldwork) (1 *))
                              work-%offset%)))
    (setf (f2cl-lib:fref work-%data%
                         (j j)
                         ((1 ldwork) (1 *))
                         work-%offset%)
            (-
             (f2cl-lib:fref work-%data%
                            (j j)
                            ((1 ldwork) (1 *))
                            work-%offset%)
             (f2cl-lib:fref work-%data%
                            (1 1)
                            ((1 ldwork) (1 *))
                            work-%offset%)))))
(setf (f2cl-lib:fref work-%data%
                     (2 2)
                     ((1 ldwork) (1 *))
                     work-%offset%)
        zero)
(setf (f2cl-lib:fref work-%data%
                     (1 (f2cl-lib:int-add n 1))
                     ((1 ldwork) (1 *))
                     work-%offset%)
        (* two mu))
(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
              ((> i
                  (f2cl-lib:int-add n
                                    (f2cl-lib:int-sub
                                     1)))
               nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data%
                         (i (f2cl-lib:int-add n 1))
```

```
                                          ((1 ldwork) (1 *))
                                          work-%offset%)
                                  (* sn
                                     (f2cl-lib:fref work-%data%
                                                    (1 (f2cl-lib:int-add i 1))
                                                    ((1 ldwork) (1 *))
                                                    work-%offset%)))))
                           (setf n2 2)
                           (setf nn (f2cl-lib:int-mul 2 (f2cl-lib:int-sub n 1)))))
                        (setf est zero)
                        (setf kase 0)
  label50
                        (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
                            (dlacon nn
                             (f2cl-lib:array-slice work
                                                   double-float
                                                   (1 (f2cl-lib:int-add n 2))
                                                   ((1 ldwork) (1 *)))
                             (f2cl-lib:array-slice work
                                                   double-float
                                                   (1 (f2cl-lib:int-add n 4))
                                                   ((1 ldwork) (1 *)))
                             iwork est kase)
                          (declare (ignore var-0 var-1 var-2 var-3))
                          (setf est var-4)
                          (setf kase var-5))
                        (cond
                          ((/= kase 0)
                           (cond
                             ((= kase 1)
                              (cond
                                ((= n2 1)
                                 (multiple-value-bind
                                       (var-0 var-1 var-2 var-3 var-4 var-5 var-6
                                        var-7 var-8 var-9 var-10)
                                     (dlaqtr t t
                                      (f2cl-lib:int-sub n 1)
                                      (f2cl-lib:array-slice work
                                                            double-float
                                                            (2 2)
                                                            ((1 ldwork) (1 *)))
                                      ldwork dummy dumm scale
                                      (f2cl-lib:array-slice work
                                                            double-float
                                                            (1
                                                             (f2cl-lib:int-add n
```

```
                                                    4))
                                ((1 ldwork) (1 *)))
        (f2cl-lib:array-slice work
                                double-float
                                (1
                                 (f2cl-lib:int-add n
                                                    6))
                                ((1 ldwork) (1 *)))
        ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4
                      var-5 var-6 var-8 var-9))
     (setf scale var-7)
     (setf ierr var-10)))
  (t
   (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6
          var-7 var-8 var-9 var-10)
       (dlaqtr t nil
        (f2cl-lib:int-sub n 1)
        (f2cl-lib:array-slice work
                                double-float
                                (2 2)
                                ((1 ldwork) (1 *)))
        ldwork
        (f2cl-lib:array-slice work
                                double-float
                                (1
                                 (f2cl-lib:int-add n
                                                    1))
                                ((1 ldwork) (1 *)))
        mu scale
        (f2cl-lib:array-slice work
                                double-float
                                (1
                                 (f2cl-lib:int-add n
                                                    4))
                                ((1 ldwork) (1 *)))
        (f2cl-lib:array-slice work
                                double-float
                                (1
                                 (f2cl-lib:int-add n
                                                    6))
                                ((1 ldwork) (1 *)))
        ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4
                      var-5 var-6 var-8 var-9))
```

```
      (setf scale var-7)
      (setf ierr var-10)))))
(t
 (cond
   ((= n2 1)
    (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6
           var-7 var-8 var-9 var-10)
        (dlaqtr nil t
         (f2cl-lib:int-sub n 1)
         (f2cl-lib:array-slice work
                               double-float
                               (2 2)
                               ((1 ldwork) (1 *)))
         ldwork dummy dumm scale
         (f2cl-lib:array-slice work
                               double-float
                               (1
                                (f2cl-lib:int-add n
                                                  4))
                               ((1 ldwork) (1 *)))
         (f2cl-lib:array-slice work
                               double-float
                               (1
                                (f2cl-lib:int-add n
                                                  6))
                               ((1 ldwork) (1 *)))
         ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4
                       var-5 var-6 var-8 var-9))
      (setf scale var-7)
      (setf ierr var-10)))
   (t
    (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6
           var-7 var-8 var-9 var-10)
        (dlaqtr nil nil
         (f2cl-lib:int-sub n 1)
         (f2cl-lib:array-slice work
                               double-float
                               (2 2)
                               ((1 ldwork) (1 *)))
         ldwork
         (f2cl-lib:array-slice work
                               double-float
                               (1
```

```
                                                    (f2cl-lib:int-add n
                                                                      1))
                                                   ((1 ldwork) (1 *)))
                                     mu scale
                                     (f2cl-lib:array-slice work
                                                           double-float
                                                           (1
                                                            (f2cl-lib:int-add n
                                                                              4))
                                                           ((1 ldwork) (1 *)))
                                     (f2cl-lib:array-slice work
                                                           double-float
                                                           (1
                                                            (f2cl-lib:int-add n
                                                                              6))
                                                           ((1 ldwork) (1 *)))
                                     ierr)
                                    (declare (ignore var-0 var-1 var-2 var-3 var-4
                                                     var-5 var-6 var-8 var-9))
                                    (setf scale var-7)
                                    (setf ierr var-10))))))
                            (go label50))))))
                    (setf (f2cl-lib:fref sep-%data% (ks) ((1 *)) sep-%offset%)
                            (/ scale (max est smlnum)))
                    (if pair
                        (setf (f2cl-lib:fref sep-%data%
                                             ((f2cl-lib:int-add ks 1))
                                             ((1 *))
                                             sep-%offset%)
                                (f2cl-lib:fref sep-%data%
                                               (ks)
                                               ((1 *))
                                               sep-%offset%)))))
                (if pair (setf ks (f2cl-lib:int-add ks 1)))
      label60))
      end_label
            (return
             (values nil
                     nil
                     nil
                     nil
                     nil
                     nil
                     nil
                     nil
                     nil
```

```
                nil
                nil
                nil
                nil
                m
                nil
                nil
                nil
                info))))))
```

7.93 ieeeck LAPACK

⟨*ieeeck.input*⟩≡

```
)set break resume
)sys rm -f ieeeck.output
)spool ieeeck.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*ieeeck.help*⟩≡

```
==========================================================================
ieeeck examples
==========================================================================

==========================================================================
Man Page Details
==========================================================================

NAME
       IEEECK  -  called  from the ILAENV to verify that Infinity and possibly
       NaN arithmetic is safe (i.e

SYNOPSIS
       INTEGER FUNCTION IEEECK( ISPEC, ZERO, ONE )

           INTEGER      ISPEC

           REAL         ONE, ZERO

PURPOSE
       IEEECK is called from the ILAENV to verify that Infinity  and  possibly
       NaN arithmetic is safe (i.e. will not trap).


ARGUMENTS
       ISPEC   (input) INTEGER
               Specifies  whether  to  test  just  for inifinity arithmetic or
               whether to test for infinity and NaN arithmetic.  =  0:  Verify
               infinity arithmetic only.
               = 1: Verify infinity and NaN arithmetic.

       ZERO    (input) REAL
               Must  contain  the value 0.0 This is passed to prevent the com-
               piler from optimizing away this code.

       ONE     (input) REAL
               Must contain the value 1.0 This is passed to prevent  the  com-
               piler from optimizing away this code.

               RETURN  VALUE:   INTEGER = 0:  Arithmetic failed to produce the
               correct answers
               = 1:  Arithmetic produced the correct answers
```

```
                Return if we were only asked to check infinity arithmetic
```

⟨*LAPACK ieeeck*⟩≡

```
(defun ieeeck (ispec zero one)
  (declare (type (single-float) one zero) (type fixnum ispec))
  (prog ((nan1 0.0f0) (nan2 0.0f0) (nan3 0.0f0) (nan4 0.0f0) (nan5 0.0f0)
         (nan6 0.0f0) (neginf 0.0f0) (negzro 0.0f0) (newzro 0.0f0)
         (posinf 0.0f0) (ieeeck 0))
    (declare (type fixnum ieeeck)
             (type (single-float) posinf newzro negzro neginf nan6 nan5 nan4
                                  nan3 nan2 nan1))
    (setf ieeeck 1)
    (setf posinf (/ one zero))
    (cond
      ((<= posinf one)
       (setf ieeeck 0)
       (go end_label)))
    (setf neginf (/ (- one) zero))
    (cond
      ((>= neginf zero)
       (setf ieeeck 0)
       (go end_label)))
    (setf negzro (/ one (+ neginf one)))
    (cond
      ((/= negzro zero)
       (setf ieeeck 0)
       (go end_label)))
    (setf neginf (/ one negzro))
    (cond
      ((>= neginf zero)
       (setf ieeeck 0)
       (go end_label)))
    (setf newzro (+ negzro zero))
    (cond
      ((/= newzro zero)
       (setf ieeeck 0)
       (go end_label)))
    (setf posinf (/ one newzro))
    (cond
      ((<= posinf one)
       (setf ieeeck 0)
       (go end_label)))
    (setf neginf (* neginf posinf))
    (cond
      ((>= neginf zero)
       (setf ieeeck 0)
       (go end_label)))
    (setf posinf (* posinf posinf))
```

```
      (cond
        ((<= posinf one)
         (setf ieeeck 0)
         (go end_label)))
      (if (= ispec 0) (go end_label))
      (setf nan1 (+ posinf neginf))
      (setf nan2 (/ posinf neginf))
      (setf nan3 (/ posinf posinf))
      (setf nan4 (* posinf zero))
      (setf nan5 (* neginf negzro))
      (setf nan6 (* nan5 0.0f0))
      (cond
        ((= nan1 nan1)
         (setf ieeeck 0)
         (go end_label)))
      (cond
        ((= nan2 nan2)
         (setf ieeeck 0)
         (go end_label)))
      (cond
        ((= nan3 nan3)
         (setf ieeeck 0)
         (go end_label)))
      (cond
        ((= nan4 nan4)
         (setf ieeeck 0)
         (go end_label)))
      (cond
        ((= nan5 nan5)
         (setf ieeeck 0)
         (go end_label)))
      (cond
        ((= nan6 nan6)
         (setf ieeeck 0)
         (go end_label)))
  end_label
      (return (values ieeeck nil nil nil))))
```

7.94 ilaenv LAPACK

⟨*ilaenv.input*⟩≡

```
)set break resume
)sys rm -f ilaenv.output
)spool ilaenv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*ilaenv.help*⟩≡

```
  ==========================================================================
  ilaenv examples
  ==========================================================================

  ==========================================================================
  Man Page Details
  ==========================================================================

  NAME
         ILAENV  -  called  from the LAPACK routines to choose problem-dependent
         parameters for the local environment

  SYNOPSIS
         INTEGER FUNCTION ILAENV( ISPEC, NAME, OPTS, N1, N2, N3, N4 )

             CHARACTER*(  * ) NAME, OPTS

             INTEGER      ISPEC, N1, N2, N3, N4

  PURPOSE
         ILAENV is called from the LAPACK routines to  choose  problem-dependent
         parameters  for  the local environment.  See ISPEC for a description of
         the parameters.

         ILAENV returns an INTEGER
         if ILAENV >= 0: ILAENV returns the value of the parameter specified  by
         ISPEC  if ILAENV < 0:  if ILAENV = -k, the k-th argument had an illegal
         value.

         This version provides a set of parameters which should give  good,  but
         not  optimal, performance on many of the currently available computers.
         Users are encouraged to modify this subroutine to set the tuning param-
         eters  for  their  particular machine using the option and problem size
         information in the arguments.

         This routine will not function correctly if  it  is  converted  to  all
         lower case.  Converting it to all upper case is allowed.


  ARGUMENTS
         ISPEC   (input) INTEGER
                 Specifies  the parameter to be returned as the value of ILAENV.
                 = 1: the optimal blocksize; if this value is  1,  an  unblocked
                 algorithm  will  give  the  best performance.  = 2: the minimum
                 block size for which the block routine should be used;  if  the
```

usable block size is less than this value, an unblocked routine should be used. = 3: the crossover point (in a block routine, for N less than this value, an unblocked routine should be used) = 4: the number of shifts, used in the nonsymmetric eigenvalue routines (DEPRECATED) = 5: the minimum column dimension for blocking to be used; rectangular blocks must have dimension at least k by m, where k is given by ILAENV(2,...) and m by ILAENV(5,...) = 6: the crossover point for the SVD (when reducing an m by n matrix to bidiagonal form, if max(m,n)/min(m,n) exceeds this value, a QR factorization is used first to reduce the matrix to a triangular form.) = 7: the number of processors
= 8: the crossover point for the multishift QR method for nonsymmetric eigenvalue problems (DEPRECATED) = 9: maximum size of the subproblems at the bottom of the computation tree in the divide-and-conquer algorithm (used by xGELSD and xGESDD) =10: ieee NaN arithmetic can be trusted not to trap
=11: infinity arithmetic can be trusted not to trap
12 <= ISPEC <= 16: xHSEQR or one of its subroutines, see IPARMQ for detailed explanation

NAME (input) CHARACTER*(*)
The name of the calling subroutine, in either upper case or lower case.

OPTS (input) CHARACTER*(*)
The character options to the subroutine NAME, concatenated into a single character string. For example, UPLO = 'U', TRANS = 'T', and DIAG = 'N' for a triangular routine would be specified as OPTS = 'UTN'.

N1 (input) INTEGER
N2 (input) INTEGER N3 (input) INTEGER N4 (input) INTEGER Problem dimensions for the subroutine NAME; these may not all be required.

FURTHER DETAILS

The following conventions have been used when calling ILAENV from the LAPACK routines:

1) OPTS is a concatenation of all of the character options to subroutine NAME, in the same order that they appear in the argument list for NAME, even if they are not used in determining the value of the parameter specified by ISPEC.
2) The problem dimensions N1, N2, N3, N4 are specified in the order that they appear in the argument list for NAME. N1 is used first, N2 second, and so on, and unused problem dimensions are

```
      passed a value of -1.
   3) The parameter value returned by ILAENV is checked for validity in
      the calling subroutine.  For example, ILAENV is used to retrieve
      the optimal blocksize for STRTRI as follows:

      NB = ILAENV( 1, 'STRTRI', UPLO // DIAG, N, -1, -1, -1 )
      IF( NB.LE.1 ) NB = MAX( 1, N )
```

⟨*LAPACK ilaenv*⟩≡

```
(defun ilaenv (ispec name opts n1 n2 n3 n4)
  (declare (type character opts name)
           (type fixnum n4 n3 n2 n1 ispec))
  (f2cl-lib:with-multi-array-data
      ((name character name-%data% name-%offset%)
       (opts character opts-%data% opts-%offset%))
    (prog ((i 0) (ic 0) (iz 0) (nb 0) (nbmin 0) (nx 0)
           (subnam
            (make-array '(6) :element-type 'character :initial-element #\ ))
           (c3 (make-array '(3) :element-type 'character :initial-element #\ ))
           (c2 (make-array '(2) :element-type 'character :initial-element #\ ))
           (c4 (make-array '(2) :element-type 'character :initial-element #\ ))
           (c1 (make-array '(1) :element-type 'character :initial-element #\ ))
           (cname nil) (sname nil) (ilaenv 0) (char$ 0.0f0))
      (declare (type (single-float) char$)
               (type (member t nil) sname cname)
               (type (simple-array character (1)) c1)
               (type (simple-array character (2)) c4 c2)
               (type (simple-array character (3)) c3)
               (type (simple-array character (6)) subnam)
               (type fixnum ilaenv nx nbmin nb iz ic i))
      (f2cl-lib:computed-goto
       (label100 label100 label100 label400 label500 label600 label700 label800
        label900 label1000 label1100)
       ispec)
      (setf ilaenv -1)
      (go end_label)
 label100
      (setf ilaenv 1)
      (f2cl-lib:f2cl-set-string subnam name (string 6))
      (setf ic (f2cl-lib:ichar (f2cl-lib:fref-string subnam (1 1))))
      (setf iz (f2cl-lib:ichar "Z"))
      (cond
        ((or (= iz 90) (= iz 122))
         (cond
           ((and (>= ic 97) (<= ic 122))
            (f2cl-lib:fset-string (f2cl-lib:fref-string subnam (1 1))
                                  (code-char (f2cl-lib:int-sub ic 32)))
            (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                          ((> i 6) nil)
              (tagbody
                (setf ic (f2cl-lib:ichar (f2cl-lib:fref-string subnam (i i))))
                (if (and (>= ic 97) (<= ic 122))
                    (f2cl-lib:fset-string (f2cl-lib:fref-string subnam (i i))
                                          (code-char
```

```
                                        (f2cl-lib:int-sub ic 32)))))))))
          ((or (= iz 233) (= iz 169))
           (cond
             ((or (and (>= ic 129) (<= ic 137))
                  (and (>= ic 145) (<= ic 153))
                  (and (>= ic 162) (<= ic 169)))
              (f2cl-lib:fset-string (f2cl-lib:fref-string subnam (1 1))
                                    (code-char (f2cl-lib:int-add ic 64)))
              (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                            ((> i 6) nil)
                (tagbody
                  (setf ic (f2cl-lib:ichar (f2cl-lib:fref-string subnam (i i))))
                  (if
                   (or (and (>= ic 129) (<= ic 137))
                       (and (>= ic 145) (<= ic 153))
                       (and (>= ic 162) (<= ic 169)))
                   (f2cl-lib:fset-string (f2cl-lib:fref-string subnam (i i))
                                         (code-char
                                          (f2cl-lib:int-add ic 64)))))))))
          ((or (= iz 218) (= iz 250))
           (cond
             ((and (>= ic 225) (<= ic 250))
              (f2cl-lib:fset-string (f2cl-lib:fref-string subnam (1 1))
                                    (code-char (f2cl-lib:int-sub ic 32)))
              (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                            ((> i 6) nil)
                (tagbody
                  (setf ic (f2cl-lib:ichar (f2cl-lib:fref-string subnam (i i))))
                  (if (and (>= ic 225) (<= ic 250))
                      (f2cl-lib:fset-string (f2cl-lib:fref-string subnam (i i))
                                            (code-char
                                             (f2cl-lib:int-sub ic 32)))))))))
          (f2cl-lib:f2cl-set-string c1
                                    (f2cl-lib:fref-string subnam (1 1))
                                    (string 1))
          (setf sname (or (f2cl-lib:fstring-= c1 "S") (f2cl-lib:fstring-= c1 "D")))
          (setf cname (or (f2cl-lib:fstring-= c1 "C") (f2cl-lib:fstring-= c1 "Z")))
          (if (not (or cname sname)) (go end_label))
          (f2cl-lib:f2cl-set-string c2
                                    (f2cl-lib:fref-string subnam (2 3))
                                    (string 2))
          (f2cl-lib:f2cl-set-string c3
                                    (f2cl-lib:fref-string subnam (4 6))
                                    (string 3))
          (f2cl-lib:f2cl-set-string c4 (f2cl-lib:fref-string c3 (2 3)) (string 2))
          (f2cl-lib:computed-goto (label110 label200 label300) ispec)
```

```
label110
     (setf nb 1)
     (cond
       ((f2cl-lib:fstring-= c2 "GE")
        (cond
          ((f2cl-lib:fstring-= c3 "TRF")
           (cond
             (sname
              (setf nb 64))
             (t
              (setf nb 64))))
          ((or (f2cl-lib:fstring-= c3 "QRF")
               (f2cl-lib:fstring-= c3 "RQF")
               (f2cl-lib:fstring-= c3 "LQF")
               (f2cl-lib:fstring-= c3 "QLF"))
           (cond
             (sname
              (setf nb 32))
             (t
              (setf nb 32))))
          ((f2cl-lib:fstring-= c3 "HRD")
           (cond
             (sname
              (setf nb 32))
             (t
              (setf nb 32))))
          ((f2cl-lib:fstring-= c3 "BRD")
           (cond
             (sname
              (setf nb 32))
             (t
              (setf nb 32))))
          ((f2cl-lib:fstring-= c3 "TRI")
           (cond
             (sname
              (setf nb 64))
             (t
              (setf nb 64))))))
       ((f2cl-lib:fstring-= c2 "PO")
        (cond
          ((f2cl-lib:fstring-= c3 "TRF")
           (cond
             (sname
              (setf nb 64))
             (t
              (setf nb 64))))))
```

```
((f2cl-lib:fstring-= c2 "SY")
 (cond
   ((f2cl-lib:fstring-= c3 "TRF")
    (cond
      (sname
       (setf nb 64))
      (t
       (setf nb 64))))
   ((and sname (f2cl-lib:fstring-= c3 "TRD"))
    (setf nb 32))
   ((and sname (f2cl-lib:fstring-= c3 "GST"))
    (setf nb 64))))
((and cname (f2cl-lib:fstring-= c2 "HE"))
 (cond
   ((f2cl-lib:fstring-= c3 "TRF")
    (setf nb 64))
   ((f2cl-lib:fstring-= c3 "TRD")
    (setf nb 32))
   ((f2cl-lib:fstring-= c3 "GST")
    (setf nb 64))))
((and sname (f2cl-lib:fstring-= c2 "OR"))
 (cond
   ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "G")
    (cond
      ((or (f2cl-lib:fstring-= c4 "QR")
           (f2cl-lib:fstring-= c4 "RQ")
           (f2cl-lib:fstring-= c4 "LQ")
           (f2cl-lib:fstring-= c4 "QL")
           (f2cl-lib:fstring-= c4 "HR")
           (f2cl-lib:fstring-= c4 "TR")
           (f2cl-lib:fstring-= c4 "BR"))
       (setf nb 32))))
   ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "M")
    (cond
      ((or (f2cl-lib:fstring-= c4 "QR")
           (f2cl-lib:fstring-= c4 "RQ")
           (f2cl-lib:fstring-= c4 "LQ")
           (f2cl-lib:fstring-= c4 "QL")
           (f2cl-lib:fstring-= c4 "HR")
           (f2cl-lib:fstring-= c4 "TR")
           (f2cl-lib:fstring-= c4 "BR"))
       (setf nb 32))))))
((and cname (f2cl-lib:fstring-= c2 "UN"))
 (cond
   ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "G")
    (cond
```

```
          ((or (f2cl-lib:fstring-= c4 "QR")
               (f2cl-lib:fstring-= c4 "RQ")
               (f2cl-lib:fstring-= c4 "LQ")
               (f2cl-lib:fstring-= c4 "QL")
               (f2cl-lib:fstring-= c4 "HR")
               (f2cl-lib:fstring-= c4 "TR")
               (f2cl-lib:fstring-= c4 "BR"))
           (setf nb 32))))
        ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "M")
         (cond
          ((or (f2cl-lib:fstring-= c4 "QR")
               (f2cl-lib:fstring-= c4 "RQ")
               (f2cl-lib:fstring-= c4 "LQ")
               (f2cl-lib:fstring-= c4 "QL")
               (f2cl-lib:fstring-= c4 "HR")
               (f2cl-lib:fstring-= c4 "TR")
               (f2cl-lib:fstring-= c4 "BR"))
           (setf nb 32))))))
    ((f2cl-lib:fstring-= c2 "GB")
     (cond
      ((f2cl-lib:fstring-= c3 "TRF")
       (cond
        (sname
         (cond
          ((<= n4 64)
           (setf nb 1))
          (t
           (setf nb 32))))
        (t
         (cond
          ((<= n4 64)
           (setf nb 1))
          (t
           (setf nb 32)))))))))
    ((f2cl-lib:fstring-= c2 "PB")
     (cond
      ((f2cl-lib:fstring-= c3 "TRF")
       (cond
        (sname
         (cond
          ((<= n2 64)
           (setf nb 1))
          (t
           (setf nb 32))))
        (t
         (cond
```

```
                  ((<= n2 64)
                   (setf nb 1))
                  (t
                   (setf nb 32))))))))
          ((f2cl-lib:fstring-= c2 "TR")
           (cond
             ((f2cl-lib:fstring-= c3 "TRI")
              (cond
                (sname
                 (setf nb 64))
                (t
                 (setf nb 64))))))
          ((f2cl-lib:fstring-= c2 "LA")
           (cond
             ((f2cl-lib:fstring-= c3 "UUM")
              (cond
                (sname
                 (setf nb 64))
                (t
                 (setf nb 64))))))
          ((and sname (f2cl-lib:fstring-= c2 "ST"))
           (cond
             ((f2cl-lib:fstring-= c3 "EBZ")
              (setf nb 1)))))
        (setf ilaenv nb)
        (go end_label)
   label200
        (setf nbmin 2)
        (cond
          ((f2cl-lib:fstring-= c2 "GE")
           (cond
             ((or (f2cl-lib:fstring-= c3 "QRF")
                  (f2cl-lib:fstring-= c3 "RQF")
                  (f2cl-lib:fstring-= c3 "LQF")
                  (f2cl-lib:fstring-= c3 "QLF"))
              (cond
                (sname
                 (setf nbmin 2))
                (t
                 (setf nbmin 2))))
             ((f2cl-lib:fstring-= c3 "HRD")
              (cond
                (sname
                 (setf nbmin 2))
                (t
                 (setf nbmin 2))))
```

```
   ((f2cl-lib:fstring-= c3 "BRD")
    (cond
      (sname
       (setf nbmin 2))
      (t
       (setf nbmin 2))))
   ((f2cl-lib:fstring-= c3 "TRI")
    (cond
      (sname
       (setf nbmin 2))
      (t
       (setf nbmin 2))))))
((f2cl-lib:fstring-= c2 "SY")
 (cond
   ((f2cl-lib:fstring-= c3 "TRF")
    (cond
      (sname
       (setf nbmin 8))
      (t
       (setf nbmin 8))))
   ((and sname (f2cl-lib:fstring-= c3 "TRD"))
    (setf nbmin 2))))
((and cname (f2cl-lib:fstring-= c2 "HE"))
 (cond
   ((f2cl-lib:fstring-= c3 "TRD")
    (setf nbmin 2))))
((and sname (f2cl-lib:fstring-= c2 "OR"))
 (cond
   ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "G")
    (cond
      ((or (f2cl-lib:fstring-= c4 "QR")
           (f2cl-lib:fstring-= c4 "RQ")
           (f2cl-lib:fstring-= c4 "LQ")
           (f2cl-lib:fstring-= c4 "QL")
           (f2cl-lib:fstring-= c4 "HR")
           (f2cl-lib:fstring-= c4 "TR")
           (f2cl-lib:fstring-= c4 "BR"))
       (setf nbmin 2))))
   ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "M")
    (cond
      ((or (f2cl-lib:fstring-= c4 "QR")
           (f2cl-lib:fstring-= c4 "RQ")
           (f2cl-lib:fstring-= c4 "LQ")
           (f2cl-lib:fstring-= c4 "QL")
           (f2cl-lib:fstring-= c4 "HR")
           (f2cl-lib:fstring-= c4 "TR")
```

```
                    (f2cl-lib:fstring-= c4 "BR"))
                (setf nbmin 2))))))
          ((and cname (f2cl-lib:fstring-= c2 "UN"))
           (cond
             ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "G")
              (cond
                ((or (f2cl-lib:fstring-= c4 "QR")
                     (f2cl-lib:fstring-= c4 "RQ")
                     (f2cl-lib:fstring-= c4 "LQ")
                     (f2cl-lib:fstring-= c4 "QL")
                     (f2cl-lib:fstring-= c4 "HR")
                     (f2cl-lib:fstring-= c4 "TR")
                     (f2cl-lib:fstring-= c4 "BR"))
                 (setf nbmin 2))))
             ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "M")
              (cond
                ((or (f2cl-lib:fstring-= c4 "QR")
                     (f2cl-lib:fstring-= c4 "RQ")
                     (f2cl-lib:fstring-= c4 "LQ")
                     (f2cl-lib:fstring-= c4 "QL")
                     (f2cl-lib:fstring-= c4 "HR")
                     (f2cl-lib:fstring-= c4 "TR")
                     (f2cl-lib:fstring-= c4 "BR"))
                 (setf nbmin 2)))))))
      (setf ilaenv nbmin)
      (go end_label)
  label300
      (setf nx 0)
      (cond
        ((f2cl-lib:fstring-= c2 "GE")
         (cond
           ((or (f2cl-lib:fstring-= c3 "QRF")
                (f2cl-lib:fstring-= c3 "RQF")
                (f2cl-lib:fstring-= c3 "LQF")
                (f2cl-lib:fstring-= c3 "QLF"))
            (cond
              (sname
               (setf nx 128))
              (t
               (setf nx 128))))
           ((f2cl-lib:fstring-= c3 "HRD")
            (cond
              (sname
               (setf nx 128))
              (t
               (setf nx 128))))
```

```
            ((f2cl-lib:fstring-= c3 "BRD")
             (cond
               (sname
                (setf nx 128))
               (t
                (setf nx 128))))))
          ((f2cl-lib:fstring-= c2 "SY")
           (cond
             ((and sname (f2cl-lib:fstring-= c3 "TRD"))
              (setf nx 32))))
          ((and cname (f2cl-lib:fstring-= c2 "HE"))
           (cond
             ((f2cl-lib:fstring-= c3 "TRD")
              (setf nx 32))))
          ((and sname (f2cl-lib:fstring-= c2 "OR"))
           (cond
             ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "G")
              (cond
                ((or (f2cl-lib:fstring-= c4 "QR")
                     (f2cl-lib:fstring-= c4 "RQ")
                     (f2cl-lib:fstring-= c4 "LQ")
                     (f2cl-lib:fstring-= c4 "QL")
                     (f2cl-lib:fstring-= c4 "HR")
                     (f2cl-lib:fstring-= c4 "TR")
                     (f2cl-lib:fstring-= c4 "BR"))
                 (setf nx 128))))))
          ((and cname (f2cl-lib:fstring-= c2 "UN"))
           (cond
             ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "G")
              (cond
                ((or (f2cl-lib:fstring-= c4 "QR")
                     (f2cl-lib:fstring-= c4 "RQ")
                     (f2cl-lib:fstring-= c4 "LQ")
                     (f2cl-lib:fstring-= c4 "QL")
                     (f2cl-lib:fstring-= c4 "HR")
                     (f2cl-lib:fstring-= c4 "TR")
                     (f2cl-lib:fstring-= c4 "BR"))
                 (setf nx 128)))))))
      (setf ilaenv nx)
      (go end_label)
label400
      (setf ilaenv 6)
      (go end_label)
label500
      (setf ilaenv 2)
      (go end_label)
```

```
  label600
       (setf ilaenv
               (f2cl-lib:int
                (*
                 (coerce (realpart
                  (min (the fixnum n1) (the fixnum n2))) 'single-float)
                 1.6f0)))
       (go end_label)
  label700
       (setf ilaenv 1)
       (go end_label)
  label800
       (setf ilaenv 50)
       (go end_label)
  label900
       (setf ilaenv 25)
       (go end_label)
  label1000
       (setf ilaenv 0)
       (cond
         ((= ilaenv 1)
          (setf ilaenv (ieeeck 0 0.0f0 1.0f0))))
       (go end_label)
  label1100
       (setf ilaenv 0)
       (cond
         ((= ilaenv 1)
          (setf ilaenv (ieeeck 1 0.0f0 1.0f0))))
  end_label
       (return (values ilaenv nil nil nil nil nil nil nil)))))
```

7.95 zlange LAPACK

⟨*zlange.input*⟩≡

```
)set break resume
)sys rm -f zlange.output
)spool zlange.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zlange.help*⟩≡

```
======================================================================
zlange examples
======================================================================

======================================================================
Man Page Details
======================================================================

NAME
       ZLANGE  -  the  value  of  the  one norm, or the Frobenius norm, or the
       infinity norm, or the element of largest absolute value  of  a  complex
       matrix A

SYNOPSIS
       DOUBLE PRECISION FUNCTION ZLANGE( NORM, M, N, A, LDA, WORK )

           CHARACTER    NORM

           INTEGER      LDA, M, N

           DOUBLE       PRECISION WORK( * )

           COMPLEX*16   A( LDA, * )

PURPOSE
       ZLANGE   returns  the value of the one norm,  or the Frobenius norm, or
       the  infinity norm,  or the  element of  largest absolute value   of  a
       complex matrix A.


DESCRIPTION
       ZLANGE returns the value

          ZLANGE = ( max(abs(A(i,j))), NORM = 'M' or 'm'
                   (
                   ( norm1(A),         NORM = '1', 'O' or 'o'
                   (
                   ( normI(A),         NORM = 'I' or 'i'
                   (
                   ( normF(A),         NORM = 'F', 'f', 'E' or 'e'

       where   norm1   denotes the  one norm of a matrix (maximum column sum),
       normI  denotes the  infinity norm  of a matrix  (maximum row  sum)  and
       normF   denotes  the  Frobenius norm of a matrix (square root of sum of
       squares).  Note that  max(abs(A(i,j)))   is  not  a  consistent  matrix
```

```
       norm.


 ARGUMENTS
       NORM    (input) CHARACTER*1
               Specifies the value to be returned in ZLANGE as described
               above.

       M       (input) INTEGER
               The number of rows of the matrix A.  M >= 0.  When M = 0,
               ZLANGE is set to zero.

       N       (input) INTEGER
               The number of columns of the matrix A.  N >= 0.  When N = 0,
               ZLANGE is set to zero.

       A       (input) COMPLEX*16 array, dimension (LDA,N)
               The m by n matrix A.

       LDA     (input) INTEGER
               The leading dimension of the array A.  LDA >= max(M,1).

       WORK    (workspace) DOUBLE PRECISION array, dimension (MAX(1,LWORK)),
               where LWORK >= M when NORM = 'I'; otherwise, WORK is not refer-
               enced.
```

⟨*LAPACK zlange*⟩≡

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun zlange (norm m n a lda work)
    (declare (type (simple-array double-float (*)) work)
             (type (simple-array (complex double-float) (*)) a)
             (type fixnum lda n m)
             (type character norm))
    (f2cl-lib:with-multi-array-data
        ((norm character norm-%data% norm-%offset%)
         (a (complex double-float) a-%data% a-%offset%)
         (work double-float work-%data% work-%offset%))
      (prog ((scale 0.0) (sum 0.0) (value 0.0) (i 0) (j 0) (zlange 0.0))
        (declare (type fixnum i j)
                 (type (double-float) scale sum value zlange))
        (cond
          ((= (min (the fixnum m) (the fixnum n)) 0)
           (setf value zero))
          ((char-equal norm #\M)
           (setf value zero)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i m) nil)
                 (tagbody
                   (setf value
                           (max value
                                (abs
                                 (f2cl-lib:fref a-%data%
                                                (i j)
                                                ((1 lda) (1 *))
                                                a-%offset%)))))))))
          ((or (char-equal norm #\O) (f2cl-lib:fstring-= norm "1"))
           (setf value zero)
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                         ((> j n) nil)
             (tagbody
               (setf sum zero)
               (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                             ((> i m) nil)
                 (tagbody
                   (setf sum
                           (+ sum
                              (abs
```

```
                        (f2cl-lib:fref a-%data%
                                       (i j)
                                       ((1 lda) (1 *))
                                       a-%offset%))))))
          (setf value (max value sum)))))
     ((char-equal norm #\I)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i m) nil)
        (tagbody
          (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
                  zero)))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
                      (+
                       (f2cl-lib:fref work-%data%
                                      (i)
                                      ((1 *))
                                      work-%offset%)
                       (abs
                        (f2cl-lib:fref a-%data%
                                       (i j)
                                       ((1 lda) (1 *))
                                       a-%offset%))))))))))
      (setf value zero)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i m) nil)
        (tagbody
          (setf value
                  (max value
                       (f2cl-lib:fref work-%data%
                                      (i)
                                      ((1 *))
                                      work-%offset%))))))
     ((or (char-equal norm #\F) (char-equal norm #\E))
      (setf scale zero)
      (setf sum one)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
        (tagbody
          (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
              (zlassq m
```

```
                  (f2cl-lib:array-slice a
                                        (complex double-float)
                                        (1 j)
                                        ((1 lda) (1 *)))
                  1 scale sum)
                 (declare (ignore var-0 var-1 var-2))
                 (setf scale var-3)
                 (setf sum var-4))))
            (setf value (* scale (f2cl-lib:fsqrt sum)))))
         (setf zlange value)
         (return (values zlange nil nil nil nil nil nil))))))
```

7.96 zlassq LAPACK

⟨*zlassq.input*⟩≡

```
)set break resume
)sys rm -f zlassq.output
)spool zlassq.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

⟨*zlassq.help*⟩≡

```
  ======================================================================
  zlassq examples
  ======================================================================

  ======================================================================
  Man Page Details
  ======================================================================

  NAME
         ZLASSQ  - the values scl and ssq such that   ( scl**2 )*ssq = x( 1 )**2
         +...+ x( n )**2 + ( scale**2 )*sumsq,

  SYNOPSIS
         SUBROUTINE ZLASSQ( N, X, INCX, SCALE, SUMSQ )

             INTEGER        INCX, N

             DOUBLE         PRECISION SCALE, SUMSQ

             COMPLEX*16     X( * )

  PURPOSE
         ZLASSQ returns the values scl and ssq such that

         where x( i ) = abs( X( 1 + ( i - 1 )*INCX ) ). The value  of  sumsq  is
         assumed to be at least unity and the value of ssq will then satisfy

            1.0 .le. ssq .le. ( sumsq + 2*n ).

         scale is assumed to be non-negative and scl returns the value

            scl = max( scale, abs( real( x( i ) ) ), abs( aimag( x( i ) ) ) ),
                  i

         scale  and  sumsq  must  be  supplied  in SCALE and SUMSQ respectively.
         SCALE and SUMSQ are overwritten by scl and ssq respectively.

         The routine makes only one pass through the vector X.


  ARGUMENTS
         N       (input) INTEGER
                 The number of elements to be used from the vector X.

         X       (input) COMPLEX*16 array, dimension (N)
```

```
          The vector x as described above.  x( i )  = X( 1  +  (  i  -  1
          )*INCX ), 1 <= i <= n.

  INCX    (input) INTEGER
          The  increment between successive values of the vector X.  INCX
          > 0.

  SCALE   (input/output) DOUBLE PRECISION
          On entry, the value  scale  in the equation  above.   On  exit,
          SCALE is overwritten with the value  scl .

  SUMSQ   (input/output) DOUBLE PRECISION
          On  entry,  the  value  sumsq  in the equation above.  On exit,
          SUMSQ is overwritten with the value  ssq .
```

⟨*LAPACK zlassq*⟩≡

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun zlassq (n x incx scale sumsq)
    (declare (type (double-float) sumsq scale)
             (type (simple-array (complex double-float) (*)) x)
             (type fixnum incx n))
    (f2cl-lib:with-multi-array-data
        ((x (complex double-float) x-%data% x-%offset%))
      (prog ((temp1 0.0) (ix 0))
        (declare (type (double-float) temp1) (type fixnum ix))
        (cond
          ((> n 0)
           (f2cl-lib:fdo (ix 1 (f2cl-lib:int-add ix incx))
                         ((> ix
                             (f2cl-lib:int-add 1
                              (f2cl-lib:int-mul
                               (f2cl-lib:int-add n
                                (f2cl-lib:int-sub 1))
                               incx)))
                          nil)
             (tagbody
               (cond
                 ((/= (coerce (realpart (f2cl-lib:fref x (ix) ((1 *)))) 'double-float) zero)
                  (setf temp1
                          (abs
                           (coerce (realpart
                            (f2cl-lib:fref x-%data% (ix) ((1 *)) x-%offset%)) 'double-float)))
                  (cond
                    ((< scale temp1)
                     (setf sumsq (+ 1 (* sumsq (expt (/ scale temp1) 2))))
                     (setf scale temp1))
                    (t
                     (setf sumsq (+ sumsq (expt (/ temp1 scale) 2)))))))
               (cond
                 ((/= (f2cl-lib:dimag (f2cl-lib:fref x (ix) ((1 *)))) zero)
                  (setf temp1
                          (abs
                           (f2cl-lib:dimag
                            (f2cl-lib:fref x-%data% (ix) ((1 *)) x-%offset%))))
                  (cond
                    ((< scale temp1)
                     (setf sumsq (+ 1 (* sumsq (expt (/ scale temp1) 2))))
                     (setf scale temp1))
                    (t
                     (setf sumsq (+ sumsq (expt (/ temp1 scale) 2)))))))))))))))
```

```
(return (values nil nil nil scale sumsq))))))
```

Chapter 8

Chunk collections

⟨*Numerics*⟩≡
```
(in-package "BOOT")

⟨BLAS dcabs1⟩
⟨BLAS 1 dasum⟩
⟨BLAS 1 daxpy⟩
⟨BLAS 1 dcopy⟩
```

⟨*untested*⟩≡
⟨*BLAS lsame*⟩
⟨*BLAS xerbla*⟩

⟨*BLAS 1 ddot*⟩
⟨*BLAS 1 dnrm2*⟩
⟨*BLAS 1 drotg*⟩
⟨*BLAS 1 drot*⟩
⟨*BLAS 1 dscal*⟩
⟨*BLAS 1 dswap*⟩
⟨*BLAS 1 dzasum*⟩
⟨*BLAS 1 dznrm2*⟩
⟨*BLAS 1 icamax*⟩
⟨*BLAS 1 idamax*⟩
⟨*BLAS 1 isamax*⟩
⟨*BLAS 1 izamax*⟩
⟨*BLAS 1 zaxpy*⟩
⟨*BLAS 1 zcopy*⟩
⟨*BLAS 1 zdotc*⟩
⟨*BLAS 1 zdotu*⟩
⟨*BLAS 1 zdscal*⟩
⟨*BLAS 1 zrotg*⟩
⟨*BLAS 1 zscal*⟩
⟨*BLAS 1 zswap*⟩

⟨*BLAS 2 dgbmv*⟩
⟨*BLAS 2 dgemv*⟩
⟨*BLAS 2 dger*⟩
⟨*BLAS 2 dsbmv*⟩
⟨*BLAS 2 dspmv*⟩
⟨*BLAS 2 dspr2*⟩
⟨*BLAS 2 dspr*⟩
⟨*BLAS 2 dsymv*⟩
⟨*BLAS 2 dsyr2*⟩
⟨*BLAS 2 dsyr*⟩
⟨*BLAS 2 dtbmv*⟩
⟨*BLAS 2 dtbsv*⟩
⟨*BLAS 2 dtpmv*⟩
⟨*BLAS 2 dtpsv*⟩
⟨*BLAS 2 dtrmv*⟩
⟨*BLAS 2 dtrsv*⟩
⟨*BLAS 2 zgbmv*⟩
⟨*BLAS 2 zgemv*⟩
⟨*BLAS 2 zgerc*⟩
⟨*BLAS 2 zgeru*⟩
⟨*BLAS 2 zhbmv*⟩

⟨*BLAS 2 zhemv*⟩
⟨*BLAS 2 zher2*⟩
⟨*BLAS 2 zher*⟩
⟨*BLAS 2 zhpmv*⟩
⟨*BLAS 2 zhpr2*⟩
⟨*BLAS 2 zhpr*⟩
⟨*BLAS 2 ztbmv*⟩
⟨*BLAS 2 ztbsv*⟩
⟨*BLAS 2 ztpmv*⟩
⟨*BLAS 2 ztpsv*⟩
⟨*BLAS 2 ztrmv*⟩
⟨*BLAS 2 ztrsv*⟩

⟨*BLAS 3 dgemm*⟩
⟨*BLAS 3 dsymm*⟩
⟨*BLAS 3 dsyr2k*⟩
⟨*BLAS 3 dsyrk*⟩
⟨*BLAS 3 dtrmm*⟩
⟨*BLAS 3 dtrsm*⟩
⟨*BLAS 3 zgemm*⟩
⟨*BLAS 3 zhemm*⟩
⟨*BLAS 3 zher2k*⟩
⟨*BLAS 3 zherk*⟩
⟨*BLAS 3 zsymm*⟩
⟨*BLAS 3 zsyr2k*⟩
⟨*BLAS 3 zsyrk*⟩
⟨*BLAS 3 ztrmm*⟩
⟨*BLAS 3 ztrsm*⟩

⟨*LAPACK dbdsdc*⟩
⟨*LAPACK dbdsqr*⟩
⟨*LAPACK ddisna*⟩
⟨*LAPACK dgebak*⟩
⟨*LAPACK dgebal*⟩
⟨*LAPACK dgebd2*⟩
⟨*LAPACK dgebrd*⟩
⟨*LAPACK dgeev*⟩
⟨*LAPACK dgeevx*⟩
⟨*LAPACK dgehd2*⟩
⟨*LAPACK dgehrd*⟩
⟨*LAPACK dgelq2*⟩
⟨*LAPACK dgelqf*⟩
⟨*LAPACK dgeqr2*⟩
⟨*LAPACK dgeqrf*⟩
⟨*LAPACK dgesdd*⟩
⟨*LAPACK dgesvd*⟩

⟨*LAPACK dgesv*⟩
⟨*LAPACK dgetf2*⟩
⟨*LAPACK dgetrf*⟩
⟨*LAPACK dgetrs*⟩
⟨*LAPACK dhseqr*⟩
⟨*LAPACK dlabad*⟩
⟨*LAPACK dlabrd*⟩
⟨*LAPACK dlacon*⟩
⟨*LAPACK dlacpy*⟩
⟨*LAPACK dladiv*⟩
⟨*LAPACK dlaed6*⟩
⟨*LAPACK dlaexc*⟩
⟨*LAPACK dlahqr*⟩
⟨*LAPACK dlahrd*⟩
⟨*LAPACK dlaln2*⟩
⟨*LAPACK dlamch*⟩
⟨*LAPACK dlamc1*⟩
⟨*LAPACK dlamc2*⟩
⟨*LAPACK dlamc3*⟩
⟨*LAPACK dlamc4*⟩
⟨*LAPACK dlamc5*⟩
⟨*LAPACK dlamrg*⟩
⟨*LAPACK dlange*⟩
⟨*LAPACK dlanhs*⟩
⟨*LAPACK dlanst*⟩
⟨*LAPACK dlanv2*⟩
⟨*LAPACK dlapy2*⟩
⟨*LAPACK dlaqtr*⟩
⟨*LAPACK dlarfb*⟩
⟨*LAPACK dlarfg*⟩
⟨*LAPACK dlarf*⟩
⟨*LAPACK dlarft*⟩
⟨*LAPACK dlarfx*⟩
⟨*LAPACK dlartg*⟩
⟨*LAPACK dlas2*⟩
⟨*LAPACK dlascl*⟩
⟨*LAPACK dlasd0*⟩
⟨*LAPACK dlasd1*⟩
⟨*LAPACK dlasd2*⟩
⟨*LAPACK dlasd3*⟩
⟨*LAPACK dlasd4*⟩
⟨*LAPACK dlasd5*⟩
⟨*LAPACK dlasd6*⟩
⟨*LAPACK dlasd7*⟩
⟨*LAPACK dlasd8*⟩
⟨*LAPACK dlasda*⟩

⟨ *LAPACK dlasdq* ⟩
⟨ *LAPACK dlasdt* ⟩
⟨ *LAPACK dlaset* ⟩
⟨ *LAPACK dlasq1* ⟩
⟨ *LAPACK dlasq2* ⟩
⟨ *LAPACK dlasq3* ⟩
⟨ *LAPACK dlasq4* ⟩
⟨ *LAPACK dlasq5* ⟩
⟨ *LAPACK dlasq6* ⟩
⟨ *LAPACK dlasr* ⟩
⟨ *LAPACK dlasrt* ⟩
⟨ *LAPACK dlassq* ⟩
⟨ *LAPACK dlasv2* ⟩
⟨ *LAPACK dlaswp* ⟩
⟨ *LAPACK dlasy2* ⟩
⟨ *LAPACK dorg2r* ⟩
⟨ *LAPACK dorgbr* ⟩
⟨ *LAPACK dorghr* ⟩
⟨ *LAPACK dorgl2* ⟩
⟨ *LAPACK dorglq* ⟩
⟨ *LAPACK dorgqr* ⟩
⟨ *LAPACK dorm2r* ⟩
⟨ *LAPACK dormbr* ⟩
⟨ *LAPACK dorml2* ⟩
⟨ *LAPACK dormlq* ⟩
⟨ *LAPACK dormqr* ⟩
⟨ *LAPACK dtrevc* ⟩
⟨ *LAPACK dtrexc* ⟩
⟨ *LAPACK dtrsna* ⟩
⟨ *LAPACK ieeeck* ⟩
⟨ *LAPACK ilaenv* ⟩
⟨ *LAPACK zlange* ⟩
⟨ *LAPACK zlassq* ⟩

Bibliography

[1] documentation source `ftp://ftp.netlib.org/lapack/manpages.tgz`

[2] documentation source `http://www.math.utah.edu/software/lapack/lapack-blas.html`

[3] documentation source Written on 22-October-1986. Jack Dongarra, Argonne National Lab. Jeremy Du Croz, Nag Central Office. Sven Hammarling, Nag Central Office. Richard Hanson, Sandia National Labs.

[4] Householder, Alston S. "Principles of Numerical Analysis" Dover Publications, Mineola, NY ISBN 0-486-45312-X (1981)

[5] Golub, Gene H. and Van Loan, Charles F. "Matrix Computations" Johns Hopkins University Press ISBN 0-8018-3772-3 (1989)

[6] Higham, Nicholas J. "Accuracy and stability of numerical algorithms" SIAM Philadelphia, PA ISBN 0-89871-521-0 (2002)

Chapter 9

Index