UCID-19631,19632,19633

SLATEC5 (REBAK through ZBIRY)

Table of Contents

Preface 8
Introduction 9
    Using SLATEC Documentation 9
    Loading SLATEC Under UNICOS 9
Subroutine Descriptions 11
    REBAK 11
    REBAKB 13
    REDUC 15
    REDUC2 17
    RF 19
    RFFTB1 24
    RFFTF1 26
    RFFTI1 28
    RG 29
    RGAUSS 31
    RGG 32
    RJ 34
    RPQR79 40
    RPZERO 41
    RS 42
    RSB 44
    RSG 46
    RSGAB 48
    RSGBA 50
    RSP 52
    RST 54
    RT 56
    RUNIF 58
    SASUM 59
    SAXPY 60
    SBCG 61
    SBHIN 66
    SBOCLS 69
    SBOLS 78
    SCASUM 85
    SCG 86
    SCGN 91
    SCGS 96
    SCHDC 101
    SCHDD 103
    SCHEX 106
    SCHUD 109
    SCNRM2 111

SCOPY 113
SCOPYM 114
SCOV 115
SCPPLT 118
SDASSL 121
SDOT 137
SDRIV1 138
SDRIV2 143
SDRIV3 149
SDSDOT 162
SEPELI 163
SEPX4 171
SGBCO 178
SGBDI 181
SGBFA 182
SGBMV 184
SGBSL 187
SGECO 189
SGEDI 191
SGEEV 193
SGEFA 195
SGEFS 197
SGEIR 199
SGEMM 201
SGEMV 204
SGER 206
SGESL 208
SGLSS 210
SGMRES 213
SGTSL 220
SINDG 222
SINQB 223
SINQF 225
SINQI 227
SINT 228
SINTI 230
SINTRP 231
SIR 232
SLLTI2 237
SLPDOC 239
SNBCO 247
SNBDI 250
SNBFA 251
SNBFS 253
SNBIR 256
SNBSL 259

SNLS1 261
SNLS1E 272
SNRM2 281
SNSQ 283
SNSQE 291
SOMN 297
SOS 302
SPBCO 306
SPBDI 308
SPBFA 309
SPBSL 311
SPENC 313
SPLP 314
SPOCO 340
SPODI 342
SPOFA 344
SPOFS 345
SPOIR 347
SPOSL 349
SPPCO 351
SPPDI 353
SPPERM 355
SPPFA 356
SPPSL 358
SPSORT 360
SPTSL 362
SQRDC 363
SQRSL 365
SROT 368
SROTG 369
SROTM 371
SROTMG 373
SS2LT 375
SS2Y 377
SSBMV 380
SSCAL 383
SSD2S 384
SSDBCG 386
SSDCG 390
SSDCGN 394
SSDCGS 398
SSDGMR 402
SSDI 408
SSDOMN 410
SSDS 414
SSDSCL 416

SSGS 419
SSICCG 423
SSICO 427
SSICS 429
SSIDI 432
SSIEV 434
SSIFA 436
SSILUR 438
SSILUS 442
SSISL 445
SSJAC 447
SSLI 451
SSLI2 452
SSLLTI 454
SSLUBC 455
SSLUCN 459
SSLUCS 463
SSLUGM 467
SSLUI 473
SSLUI2 474
SSLUI4 477
SSLUOM 480
SSLUTI 484
SSMMI2 485
SSMMTI 488
SSMTV 489
SSMV 491
SSORT 493
SSPCO 494
SSPDI 496
SSPEV 498
SSPFA 500
SSPMV 502
SSPR 504
SSPR2 506
SSPSL 508
SSVDC 510
SSWAP 512
SSYMM 513
SSYMV 516
SSYR 518
SSYR2 520
SSYR2K 522
SSYRK 525
STBMV 528
STBSV 531

STEPS 534
STIN 537
STOUT 540
STPMV 542
STPSV 544
STRCO 546
STRDI 548
STRMM 550
STRMV 553
STRSL 555
STRSM 557
STRSV 560
TINVIT 562
TQL1 564
TQL2 566
TQLRAT 568
TRBAK1 570
TRBAK3 572
TRED1 574
TRED2 576
TRED3 578
TRI3 580
TRIDIB 581
TSTURM 583
ULSIA 585
WNNLS 588
XADD 593
XADJ 594
XC210 595
XCON 596
XERCLR 597
XERDMP 598
XERMAX 599
XERMSG 600
XGETF 604
XGETUA 605
XGETUN 606
XLEGF 607
XNRMP 610
XRED 613
XSET 614
XSETF 618
XSETUA 619
XSETUN 620
ZAIRY 621
ZBESH 624

ZBESI 627
ZBESJ 630
ZBESK 633
ZBESY 636
ZBIRY 639
Disclaimer 642
Structural Keyword Index 643
Date and Revisions 648

Preface

Scope: SLATEC5 contains brief descriptions ("prologues") for the SLATEC (version 4.1) mathematical library subroutines with names from REBAK through ZBIRY.

Availability: The SLATEC library is downloadable through LINMath (URL: http://www.llnl.gov/LCdocs/nmg1) and can be run on all LC production computers.

Consultant: For help contact the LC customer service and support hotline at 925-422-4531 (open e-mail: lc-hotline@llnl.gov, secure e-mail: lc-hotline@pop.scf.cln).

Printing: The print file for this document can be found at:

```
on the OCF: http://www.llnl.gov/LCdocs/slatec5/slatec5.pdf
on the SCF: https://lc.llnl.gov/LCdocs/slatec5/slatec5_scf.pdf
```

Introduction

Using SLATEC Documentation

Over 1600 pages of online documentation describe the 902 user-callable subroutines available in version 4.1 of the SLATEC library. Because of this unwieldy bulk, the documentation is published in five separate, but interrelated, volumes:

SLATEC1 provides introductory information on the whole library, explains the subject categories into which the SLATEC routines are grouped, and includes short descriptions of all routines (alphabetical within each subject category). Every category code is also a link (keyword) for retrieving the brief descriptions of the included routines. SLATEC1 provides the only way to compare related routines by the tasks they perform, rather than just by name.

SLATEC2 contains the calling sequence and usage details for each of the 225 subroutines from AAAAAA through D9UPAK, arranged alphabetically by name. Every subroutine name is also a link (keyword) for retrieving the corresponding description if you start at the index.

SLATEC3 contains the calling sequence and usage details for each of the 225 subroutines from DACOSH through DS2Y, arranged alphabetically by name. Every subroutine name is also a link (keyword) for retrieving the corresponding description if you start at the index.

SLATEC4 contains the calling sequence and usage details for each of the 226 subroutines from DSBMV through RD, arranged alphabetically by name. Every subroutine name is also a link (keyword) for retrieving the corresponding description if you start at the index.

SLATEC5 (THIS DOCUMENT) contains the calling sequence and usage details for each of the 226 subroutines from REBAK through ZBIRY, arranged alphabetically by name. Every subroutine name is also a link (keyword) for retrieving the corresponding description if you start at the index.

You can consult any of these documents from any open machine by running your choice of WWW client and selecting the document you want from the descriptive LC collection directory available at . Or you can specifically request the URL

`http://www.llnl.gov/LCdocs/slatec`*n*

where slatec*n* is any one of slatec1 through slatec5, depending on which volume you want.

Loading SLATEC Under UNICOS

On LC machines, the SLATEC math library file is called LIBSLATEC.A and has the full pathname

```
/usr/local/lib/libslatec.a
```

The routines in LIBSLATEC.A may use externals in LIBSCI for optimization, and that library is on the default search path (loaded automatically) under UNICOS.

Subroutine Descriptions

REBAK

```
      SUBROUTINE REBAK (NM, N, B, DL, M, Z)
***BEGIN PROLOGUE  REBAK
***PURPOSE  Form the eigenvectors of a generalized symmetric
            eigensystem from the eigenvectors of derived matrix output
            from REDUC or REDUC2.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4C4
***TYPE      SINGLE PRECISION (REBAK-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine is a translation of the ALGOL procedure REBAKA,
      NUM. MATH. 11, 99-110(1968) by Martin and Wilkinson.
      HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 303-314(1971).

      This subroutine forms the eigenvectors of a generalized
      SYMMETRIC eigensystem by back transforming those of the
      derived symmetric matrix determined by  REDUC  or  REDUC2.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameters, B and Z, as declared in the calling
           program dimension statement.  NM is an INTEGER variable.

         N is the order of the matrix system.  N is an INTEGER
           variable.  N must be less than or equal to NM.

         B contains information about the similarity transformation
           (Cholesky decomposition) used in the reduction by  REDUC
           or  REDUC2  in its strict lower triangle.  B is a two-
           dimensional REAL array, dimensioned B(NM,N).

         DL contains further information about the transformation.
           DL is a one-dimensional REAL array, dimensioned DL(N).

         M is the number of eigenvectors to be back transformed.
           M is an INTEGER variable.

         Z contains the eigenvectors to be back transformed in its
           first M columns.  Z is a two-dimensional REAL array
           dimensioned Z(NM,M).

      On Output

         Z contains the transformed eigenvectors in its first
           M columns.

      Questions and comments should be directed to B. S. Garbow,
      APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
      ------------------------------------------------------------------
```

```
***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

REBAKB

```
      SUBROUTINE REBAKB (NM, N, B, DL, M, Z)
***BEGIN PROLOGUE  REBAKB
***PURPOSE  Form the eigenvectors of a generalized symmetric
            eigensystem from the eigenvectors of derived matrix output
            from REDUC2.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4C4
***TYPE      SINGLE PRECISION (REBAKB-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine is a translation of the ALGOL procedure REBAKB,
      NUM. MATH. 11, 99-110(1968) by Martin and Wilkinson.
      HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 303-314(1971).

      This subroutine forms the eigenvectors of a generalized
      SYMMETRIC eigensystem by back transforming those of the
      derived symmetric matrix determined by  REDUC2.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameters, B and Z, as declared in the calling
           program dimension statement.  NM is an INTEGER variable.

         N is the order of the matrix system.  N is an INTEGER
           variable.  N must be less than or equal to NM.

         B contains information about the similarity transformation
           (Cholesky decomposition) used in the reduction by  REDUC2
           in its strict lower triangle.  B is a two-dimensional
           REAL array, dimensioned B(NM,N).

         DL contains further information about the transformation.
           DL is a one-dimensional REAL array, dimensioned DL(N).

         M is the number of eigenvectors to be back transformed.
           M is an INTEGER variable.

         Z contains the eigenvectors to be back transformed in its
           first M columns.  Z is a two-dimensional REAL array
           dimensioned Z(NM,M).

      On Output

         Z contains the transformed eigenvectors in its first
           M columns.

      Questions and comments should be directed to B. S. Garbow,
      APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
      ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
```

```
                 1976.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

REDUC

```
      SUBROUTINE REDUC (NM, N, A, B, DL, IERR)
***BEGIN PROLOGUE  REDUC
***PURPOSE  Reduce a generalized symmetric eigenproblem to a standard
            symmetric eigenproblem using Cholesky factorization.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4C1C
***TYPE      SINGLE PRECISION (REDUC-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine is a translation of the ALGOL procedure REDUC1,
      NUM. MATH. 11, 99-110(1968) by Martin and Wilkinson.
      HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 303-314(1971).

      This subroutine reduces the generalized SYMMETRIC eigenproblem
      Ax=(LAMBDA)Bx, where B is POSITIVE DEFINITE, to the standard
      symmetric eigenproblem using the Cholesky factorization of B.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameters, A and B, as declared in the calling
           program dimension statement.  NM is an INTEGER variable.

         N is the order of the matrices A and B.  If the Cholesky
           factor L of B is already available, N should be prefixed
           with a minus sign.  N is an INTEGER variable.

         A and B contain the real symmetric input matrices.  Only
           the full upper triangles of the matrices need be supplied.
           If N is negative, the strict lower triangle of B contains,
           instead, the strict lower triangle of its Cholesky factor L.
           A and B are two-dimensional REAL arrays, dimensioned A(NM,N)
           and B(NM,N).

        DL contains, if N is negative, the diagonal elements of L.
           DL is a one-dimensional REAL array, dimensioned DL(N).

      On Output

         A contains in its full lower triangle the full lower triangle
           of the symmetric matrix derived from the reduction to the
           standard form.  The strict upper triangle of A is unaltered.

         B contains in its strict lower triangle the strict lower
           triangle of its Cholesky factor L.  The full upper triangle
           of B is unaltered.

         DL contains the diagonal elements of L.

         IERR is an INTEGER flag set to
           Zero       for normal return,
           7*N+1      if B is not positive definite.

      Questions and comments should be directed to B. S. Garbow,
```

```
     APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
     ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

REDUC2

```
      SUBROUTINE REDUC2 (NM, N, A, B, DL, IERR)
***BEGIN PROLOGUE  REDUC2
***PURPOSE  Reduce a certain generalized symmetric eigenproblem to a
            standard symmetric eigenproblem using Cholesky
            factorization.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4C1C
***TYPE      SINGLE PRECISION (REDUC2-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine is a translation of the ALGOL procedure REDUC2,
      NUM. MATH. 11, 99-110(1968) by Martin and Wilkinson.
      HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 303-314(1971).

      This subroutine reduces the generalized SYMMETRIC eigenproblems
      ABx=(LAMBDA)x OR BAy=(LAMBDA)y, where B is POSITIVE DEFINITE,
      to the standard symmetric eigenproblem using the Cholesky
      factorization of B.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameters, A and B, as declared in the calling
           program dimension statement.  NM is an INTEGER variable.

         N is the order of the matrices A and B.  If the Cholesky
           factor L of B is already available, N should be prefixed
           with a minus sign.  N is an INTEGER variable.

         A and B contain the real symmetric input matrices.  Only
           the full upper triangles of the matrices need be supplied.
           If N is negative, the strict lower triangle of B contains,
           instead, the strict lower triangle of its Cholesky factor L.
           A and B are two-dimensional REAL arrays, dimensioned A(NM,N)
           and B(NM,N).

        DL contains, if N is negative, the diagonal elements of L.
           DL is a one-dimensional REAL array, dimensioned DL(N).

      On Output

         A contains in its full lower triangle the full lower triangle
           of the symmetric matrix derived from the reduction to the
           standard form.  The strict upper triangle of A is unaltered.

         B contains in its strict lower triangle the strict lower
           triangle of its Cholesky factor L.  The full upper triangle
           of B is unaltered.

         DL contains the diagonal elements of L.

         IERR is an INTEGER flag set to
           Zero       for normal return,
           7*N+1      if B is not positive definite.
```

```
     Questions and comments should be directed to B. S. Garbow,
     APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
     ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

RF

```
      REAL FUNCTION RF (X, Y, Z, IER)
***BEGIN PROLOGUE  RF
***PURPOSE  Compute the incomplete or complete elliptic integral of the
            1st kind.  For X, Y, and Z non-negative and at most one of
            them zero, RF(X,Y,Z) = Integral from zero to infinity of
                                    -1/2     -1/2     -1/2
                         (1/2)(t+X)    (t+Y)    (t+Z)    dt.
            If X, Y or Z is zero, the integral is complete.
***LIBRARY   SLATEC
***CATEGORY  C14
***TYPE      SINGLE PRECISION (RF-S, DRF-D)
***KEYWORDS  COMPLETE ELLIPTIC INTEGRAL, DUPLICATION THEOREM,
             INCOMPLETE ELLIPTIC INTEGRAL, INTEGRAL OF THE FIRST KIND,
             TAYLOR SERIES
***AUTHOR  Carlson, B. C.
             Ames Laboratory-DOE
             Iowa State University
             Ames, IA  50011
           Notis, E. M.
             Ames Laboratory-DOE
             Iowa State University
             Ames, IA  50011
           Pexton, R. L.
             Lawrence Livermore National Laboratory
             Livermore, CA  94550
***DESCRIPTION

   1.     RF
          Evaluate an INCOMPLETE (or COMPLETE) ELLIPTIC INTEGRAL
          of the first kind
          Standard FORTRAN function routine
          Single precision version
          The routine calculates an approximation result to
          RF(X,Y,Z) = Integral from zero to infinity of

                                    -1/2     -1/2     -1/2
                         (1/2)(t+X)    (t+Y)    (t+Z)    dt,

          where X, Y, and Z are nonnegative and at most one of them
          is zero.  If one of them is zero, the integral is COMPLETE.
          The duplication theorem is iterated until the variables are
          nearly equal, and the function is then expanded in Taylor
          series to fifth order.

   2.     Calling Sequence
          RF( X, Y, Z, IER )

          Parameters on Entry
          Values assigned by the calling routine

          X      - Single precision, nonnegative variable

          Y      - Single precision, nonnegative variable

          Z      - Single precision, nonnegative variable
```

On Return (values assigned by the RF routine)

RF - Single precision approximation to the integral

IER - Integer

IER = 0 Normal and reliable termination of the routine. It is assumed that the requested accuracy has been achieved.

IER > 0 Abnormal termination of the routine

X, Y, Z are unaltered.

3. Error Messages

Value of IER assigned by the RF routine

Value assigned	Error Message Printed
IER = 1	MIN(X,Y,Z) .LT. 0.0E0
= 2	MIN(X+Y,X+Z,Y+Z) .LT. LOLIM
= 3	MAX(X,Y,Z) .GT. UPLIM

4. Control Parameters

Values of LOLIM, UPLIM, and ERRTOL are set by the routine.

LOLIM and UPLIM determine the valid range of X, Y and Z

LOLIM - Lower limit of valid arguments

Not less than 5 * (machine minimum).

UPLIM - Upper limit of valid arguments

Not greater than (machine maximum) / 5.

Acceptable Values For:	LOLIM	UPLIM
IBM 360/370 SERIES :	3.0E-78	1.0E+75
CDC 6000/7000 SERIES :	1.0E-292	1.0E+321
UNIVAC 1100 SERIES :	1.0E-37	1.0E+37
CRAY :	2.3E-2466	1.09E+2465
VAX 11 SERIES :	1.5E-38	3.0E+37

ERRTOL determines the accuracy of the answer

The value assigned by the routine will result in solution precision within 1-2 decimals of "machine precision".

ERRTOL - Relative error due to truncation is less than
ERRTOL ** 6 / (4 * (1-ERRTOL) .

The accuracy of the computed approximation to the integral can be controlled by choosing the value of ERRTOL. Truncation of a Taylor series after terms of fifth order introduces an error less than the amount shown in the second column of the following table for each value of ERRTOL in the first column. In addition to the truncation error there will be round-off error, but in practice the total error from both sources is usually less than the amount given in the table.

Sample Choices:

ERRTOL	Relative Truncation error less than
1.0E-3	3.0E-19
3.0E-3	2.0E-16
1.0E-2	3.0E-13
3.0E-2	2.0E-10
1.0E-1	3.0E-7

Decreasing ERRTOL by a factor of 10 yields six more decimal digits of accuracy at the expense of one or two more iterations of the duplication theorem.

*Long Description:

RF Special Comments

Check by addition theorem: RF(X,X+Z,X+W) + RF(Y,Y+Z,Y+W) = RF(0,Z,W), where X,Y,Z,W are positive and X * Y = Z * W.

On Input:

X, Y, and Z are the variables in the integral RF(X,Y,Z).

On Output:

X, Y, and Z are unaltered.

****

Warning: Changes in the program may improve speed at the expense of robustness.

Special Functions via RF

Legendre form of ELLIPTIC INTEGRAL of 1st kind

$$F(PHI,K) = SIN(PHI)\ RF(COS^2(PHI),1-K^2 SIN^2(PHI),1)$$

$$K(K) = RF(0,1-K^2,1)$$

$$= \int_0^{PI/2} (1-K^2 SIN^2(PHI))^{-1/2}\ D\ PHI$$

Bulirsch form of ELLIPTIC INTEGRAL of 1st kind

$$EL1(X,KC) = X\ RF(1,1+KC^2 X^2,1+X^2)$$

Lemniscate constant A

$$A = \int_0^1 (1-S^4)^{-1/2}\ DS = RF(0,1,2) = RF(0,2,1)$$

***REFERENCES B. C. Carlson and E. M. Notis, Algorithms for incomplete elliptic integrals, ACM Transactions on Mathematical Software 7, 3 (September 1981), pp. 398-403.

B. C. Carlson, Computing elliptic integrals by duplication, Numerische Mathematik 33, (1979), pp. 1-16.

B. C. Carlson, Elliptic integrals of the first kind, SIAM Journal of Mathematical Analysis 8, (1977), pp. 231-242.

***ROUTINES CALLED R1MACH, XERMSG

***REVISION HISTORY (YYMMDD)

790801 DATE WRITTEN

890531 Changed all specific intrinsics to generic. (WRB)

891009 Removed unreferenced statement labels. (WRB)

891009 REVISION DATE from Version 3.2

```
891214  Prologue converted to Version 4.0 format.  (BAB)
900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
900326  Removed duplicate information from DESCRIPTION section.
        (WRB)
900510  Changed calls to XERMSG to standard form, and some
        editorial changes.  (RWC))
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

RFFTB1

```
      SUBROUTINE RFFTB1 (N, C, CH, WA, IFAC)
***BEGIN PROLOGUE  RFFTB1
***PURPOSE  Compute the backward fast Fourier transform of a real
            coefficient array.
***LIBRARY   SLATEC (FFTPACK)
***CATEGORY  J1A1
***TYPE      SINGLE PRECISION (RFFTB1-S, CFFTB1-C)
***KEYWORDS  FFTPACK, FOURIER TRANSFORM
***AUTHOR  Swarztrauber, P. N., (NCAR)
***DESCRIPTION

   Subroutine RFFTB1 computes the real periodic sequence from its
   Fourier coefficients (Fourier synthesis).  The transform is defined
   below at output parameter C.

   The arrays WA and IFAC which are used by subroutine RFFTB1 must be
   initialized by calling subroutine RFFTI1.

   Input Arguments

   N       the length of the array R to be transformed.  The method
           is most efficient when N is a product of small primes.
           N may change so long as different work arrays are provided.

   C       a real array of length N which contains the sequence
           to be transformed.

   CH      a real work array of length at least N.

   WA      a real work array which must be dimensioned at least N.

   IFAC    an integer work array which must be dimensioned at least 15.

           The WA and IFAC arrays must be initialized by calling
           subroutine RFFTI1, and different WA and IFAC arrays must be
           used for each different value of N.  This initialization
           does not have to be repeated so long as N remains unchanged.
           Thus subsequent transforms can be obtained faster than the
           first.  The same WA and IFAC arrays can be used by RFFTF1
           and RFFTB1.

   Output Argument

   C       For N even and for I = 1,...,N

                C(I) = C(1)+(-1)**(I-1)*C(N)

                     plus the sum from K=2 to K=N/2 of

                      2.*C(2*K-2)*COS((K-1)*(I-1)*2*PI/N)

                     -2.*C(2*K-1)*SIN((K-1)*(I-1)*2*PI/N)

           For N odd and for I = 1,...,N

                C(I) = C(1) plus the sum from K=2 to K=(N+1)/2 of
```

```
                    2.*C(2*K-2)*COS((K-1)*(I-1)*2*PI/N)

                   -2.*C(2*K-1)*SIN((K-1)*(I-1)*2*PI/N)

   Notes:  This transform is unnormalized since a call of RFFTF1
           followed by a call of RFFTB1 will multiply the input
           sequence by N.

           WA and IFAC contain initialization calculations which must
           not be destroyed between calls of subroutine RFFTF1 or
           RFFTB1.

***REFERENCES  P. N. Swarztrauber, Vectorizing the FFTs, in Parallel
                 Computations (G. Rodrigue, ed.), Academic Press,
                 1982, pp. 51-83.
***ROUTINES CALLED  RADB2, RADB3, RADB4, RADB5, RADBG
***REVISION HISTORY  (YYMMDD)
   790601  DATE WRITTEN
   830401  Modified to use SLATEC library source file format.
   860115  Modified by Ron Boisvert to adhere to Fortran 77 by
           changing dummy array size declarations (1) to (*).
   881128  Modified by Dick Valent to meet prologue standards.
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900131  Routine changed from subsidiary to user-callable.  (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

RFFTF1

```
      SUBROUTINE RFFTF1 (N, C, CH, WA, IFAC)
***BEGIN PROLOGUE  RFFTF1
***PURPOSE  Compute the forward transform of a real, periodic sequence.
***LIBRARY   SLATEC (FFTPACK)
***CATEGORY  J1A1
***TYPE      SINGLE PRECISION (RFFTF1-S, CFFTF1-C)
***KEYWORDS  FFTPACK, FOURIER TRANSFORM
***AUTHOR  Swarztrauber, P. N., (NCAR)
***DESCRIPTION

   Subroutine RFFTF1 computes the Fourier coefficients of a real
   periodic sequence (Fourier analysis).  The transform is defined
   below at output parameter C.

   The arrays WA and IFAC which are used by subroutine RFFTB1 must be
   initialized by calling subroutine RFFTI1.

   Input Arguments

   N       the length of the array R to be transformed.  The method
           is most efficient when N is a product of small primes.
           N may change so long as different work arrays are provided.

   C       a real array of length N which contains the sequence
           to be transformed.

   CH      a real work array of length at least N.

   WA      a real work array which must be dimensioned at least N.

   IFAC    an integer work array which must be dimensioned at least 15.

           The WA and IFAC arrays must be initialized by calling
           subroutine RFFTI1, and different WA and IFAC arrays must be
           used for each different value of N.  This initialization
           does not have to be repeated so long as N remains unchanged.
           Thus subsequent transforms can be obtained faster than the
           first.  The same WA and IFAC arrays can be used by RFFTF1
           and RFFTB1.

   Output Argument

   C       C(1) = the sum from I=1 to I=N of R(I)

           If N is even set L = N/2; if N is odd set L = (N+1)/2

             then for K = 2,...,L

                C(2*K-2) = the sum from I = 1 to I = N of

                     C(I)*COS((K-1)*(I-1)*2*PI/N)

                C(2*K-1) = the sum from I = 1 to I = N of

                    -C(I)*SIN((K-1)*(I-1)*2*PI/N)
```

```
            If N is even

                 C(N) = the sum from I = 1 to I = N of

                      (-1)**(I-1)*C(I)

   Notes:  This transform is unnormalized since a call of RFFTF1
           followed by a call of RFFTB1 will multiply the input
           sequence by N.

           WA and IFAC contain initialization calculations which must
           not be destroyed between calls of subroutine RFFTF1 or
           RFFTB1.

***REFERENCES  P. N. Swarztrauber, Vectorizing the FFTs, in Parallel
                Computations (G. Rodrigue, ed.), Academic Press,
                1982, pp. 51-83.
***ROUTINES CALLED  RADF2, RADF3, RADF4, RADF5, RADFG
***REVISION HISTORY  (YYMMDD)
   790601  DATE WRITTEN
   830401  Modified to use SLATEC library source file format.
   860115  Modified by Ron Boisvert to adhere to Fortran 77 by
           changing dummy array size declarations (1) to (*).
   881128  Modified by Dick Valent to meet prologue standards.
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900131  Routine changed from subsidiary to user-callable.  (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

RFFTI1

```
      SUBROUTINE RFFTI1 (N, WA, IFAC)
***BEGIN PROLOGUE  RFFTI1
***PURPOSE  Initialize a real and an integer work array for RFFTF1 and
            RFFTB1.
***LIBRARY   SLATEC (FFTPACK)
***CATEGORY  J1A1
***TYPE      SINGLE PRECISION (RFFTI1-S, CFFTI1-C)
***KEYWORDS  FFTPACK, FOURIER TRANSFORM
***AUTHOR  Swarztrauber, P. N., (NCAR)
***DESCRIPTION

   Subroutine RFFTI1 initializes the work arrays WA and IFAC which are
   used in both RFFTF1 and RFFTB1.  The prime factorization of N and a
   tabulation of the trigonometric functions are computed and stored in
   IFAC and WA, respectively.

   Input Argument

   N       the length of the sequence to be transformed.

   Output Arguments

   WA      a real work array which must be dimensioned at least N.

   IFAC    an integer work array which must be dimensioned at least 15.

   The same work arrays can be used for both RFFTF1 and RFFTB1 as long
   as N remains unchanged.  Different WA and IFAC arrays are required
   for different values of N.  The contents of WA and IFAC must not be
   changed between calls of RFFTF1 or RFFTB1.

***REFERENCES  P. N. Swarztrauber, Vectorizing the FFTs, in Parallel
                 Computations (G. Rodrigue, ed.), Academic Press,
                 1982, pp. 51-83.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   790601  DATE WRITTEN
   830401  Modified to use SLATEC library source file format.
   860115  Modified by Ron Boisvert to adhere to Fortran 77 by
           (a) changing dummy array size declarations (1) to (*),
           (b) changing references to intrinsic function FLOAT
               to REAL, and
           (c) changing definition of variable TPI by using
               FORTRAN intrinsic functions instead of DATA
               statements.
   881128  Modified by Dick Valent to meet prologue standards.
   890531  Changed all specific intrinsics to generic.  (WRB)
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900131  Routine changed from subsidiary to user-callable.  (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

RG

```
      SUBROUTINE RG (NM, N, A, WR, WI, MATZ, Z, IV1, FV1, IERR)
***BEGIN PROLOGUE  RG
***PURPOSE  Compute the eigenvalues and, optionally, the eigenvectors
            of a real general matrix.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4A2
***TYPE      SINGLE PRECISION (RG-S, CG-C)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine calls the recommended sequence of
      subroutines from the eigensystem subroutine package (EISPACK)
      To find the eigenvalues and eigenvectors (if desired)
      of a REAL GENERAL matrix.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameters, A and Z, as declared in the calling
           program dimension statement.  NM is an INTEGER variable.

         N is the order of the matrix A.  N is an INTEGER variable.
           N must be less than or equal to NM.

         A contains the real general matrix.  A is a two-dimensional
           REAL array, dimensioned A(NM,N).

         MATZ is an INTEGER variable set equal to zero if only
           eigenvalues are desired.  Otherwise, it is set to any
           non-zero integer for both eigenvalues and eigenvectors.

      On Output

         A has been destroyed.

         WR and WI contain the real and imaginary parts, respectively,
           of the eigenvalues.  The eigenvalues are unordered except
           that complex conjugate pairs of eigenvalues appear consecu-
           tively with the eigenvalue having the positive imaginary part
           first.  If an error exit is made, the eigenvalues should be
           correct for indices IERR+1, IERR+2, ..., N.  WR and WI are
           one-dimensional REAL arrays, dimensioned WR(N) and WI(N).

         Z contains the real and imaginary parts of the eigenvectors
           if MATZ is not zero.  If the J-th eigenvalue is real, the
           J-th column of Z contains its eigenvector.  If the J-th
           eigenvalue is complex with positive imaginary part, the
           J-th and (J+1)-th columns of Z contain the real and
           imaginary parts of its eigenvector.  The conjugate of this
           vector is the eigenvector for the conjugate eigenvalue.
           Z is a two-dimensional REAL array, dimensioned Z(NM,N).

         IERR is an INTEGER flag set to
           Zero       for normal return,
           10*N       if N is greater than NM,
```

```
          J          if the J-th eigenvalue has not been
                     determined after a total of 30 iterations.
                     The eigenvalues should be correct for indices
                     IERR+1, IERR+2, ..., N, but no eigenvectors are
                     computed.

        IV1 and FV1 are one-dimensional temporary storage arrays of
          dimension N.  IV1 is of type INTEGER and FV1 of type REAL.

     Questions and comments should be directed to B. S. Garbow,
     APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
     ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  BALANC, BALBAK, ELMHES, ELTRAN, HQR, HQR2
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   921103  Corrected description of IV1.  (DWL, FNF and WRB)
   END PROLOGUE
```

RGAUSS

```
      FUNCTION RGAUSS (XMEAN, SD)
***BEGIN PROLOGUE  RGAUSS
***PURPOSE  Generate a normally distributed (Gaussian) random number.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  L6A14
***TYPE      SINGLE PRECISION (RGAUSS-S)
***KEYWORDS  FNLIB, GAUSSIAN, NORMAL, RANDOM NUMBER, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 Generate a normally distributed random number, i.e., generate random
 numbers with a Gaussian distribution.  These random numbers are not
 exceptionally good -- especially in the tails of the distribution,
 but this implementation is simple and suitable for most applications.
 See R. W. Hamming, Numerical Methods for Scientists and Engineers,
 McGraw-Hill, 1962, pages 34 and 389.

             Input Arguments --
 XMEAN  the mean of the Guassian distribution.
 SD     the standard deviation of the Guassian function
          EXP (-1/2 * (X-XMEAN)**2 / SD**2)

***REFERENCES  (NONE)
***ROUTINES CALLED  RAND
***REVISION HISTORY  (YYMMDD)
   770401  DATE WRITTEN
   861211  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   910819  Added EXTERNAL statement for RAND due to problem on IBM
           RS 6000.  (WRB)
   END PROLOGUE
```

RGG

```
      SUBROUTINE RGG (NM, N, A, B, ALFR, ALFI, BETA, MATZ, Z, IERR)
***BEGIN PROLOGUE  RGG
***PURPOSE  Compute the eigenvalues and eigenvectors for a real
            generalized eigenproblem.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4B2
***TYPE      SINGLE PRECISION (RGG-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine calls the recommended sequence of
      subroutines from the eigensystem subroutine package (EISPACK)
      to find the eigenvalues and eigenvectors (if desired)
      for the REAL GENERAL GENERALIZED eigenproblem  Ax = (LAMBDA)Bx.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameters, A, B, and Z, as declared in the calling
           program dimension statement.  NM is an INTEGER variable.

         N is the order of the matrices A and B.  N is an INTEGER
           variable.  N must be less than or equal to NM.

         A contains a real general matrix.  A is a two-dimensional
           REAL array, dimensioned A(NM,N).

         B contains a real general matrix.  B is a two-dimensional
           REAL array, dimensioned B(NM,N).

         MATZ is an INTEGER variable set equal to zero if only
           eigenvalues are desired.  Otherwise, it is set to any
           non-zero integer for both eigenvalues and eigenvectors.

      On Output

         A and B have been destroyed.

         ALFR and ALFI contain the real and imaginary parts,
           respectively, of the numerators of the eigenvalues.
           ALFR and ALFI are one-dimensional REAL arrays,
           dimensioned ALFR(N) and ALFI(N).

         BETA contains the denominators of the eigenvalues,
           which are thus given by the ratios  (ALFR+I*ALFI)/BETA.
           Complex conjugate pairs of eigenvalues appear consecutively
           with the eigenvalue having the positive imaginary part first.
           BETA is a one-dimensional REAL array, dimensioned BETA(N).

         Z contains the real and imaginary parts of the eigenvectors
           if MATZ is not zero.  If the J-th eigenvalue is real, the
           J-th column of  Z  contains its eigenvector.  If the J-th
           eigenvalue is complex with positive imaginary part, the
           J-th and (J+1)-th columns of  Z  contain the real and
           imaginary parts of its eigenvector.  The conjugate of this
```

```
          vector is the eigenvector for the conjugate eigenvalue.
          Z is a two-dimensional REAL array, dimensioned Z(NM,N).

        IERR is an INTEGER flag set to
          Zero       for normal return,
          10*N       if N is greater than NM,
          J          if the J-th eigenvalue has not been
                     determined after a total of 30*N iterations.
                     The eigenvalues should be correct for indices
                     IERR+1, IERR+2, ..., N, but no eigenvectors are
                     computed.

     Questions and comments should be directed to B. S. Garbow,
     APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
     ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  QZHES, QZIT, QZVAL, QZVEC
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

RJ

```
      REAL FUNCTION RJ (X, Y, Z, P, IER)
***BEGIN PROLOGUE  RJ
***PURPOSE  Compute the incomplete or complete (X or Y or Z is zero)
            elliptic integral of the 3rd kind.  For X, Y, and Z non-
            negative, at most one of them zero, and P positive,
             RJ(X,Y,Z,P) = Integral from zero to infinity of
                                    -1/2     -1/2     -1/2     -1
                        (3/2)(t+X)    (t+Y)    (t+Z)    (t+P)  dt.
***LIBRARY   SLATEC
***CATEGORY  C14
***TYPE      SINGLE PRECISION (RJ-S, DRJ-D)
***KEYWORDS  COMPLETE ELLIPTIC INTEGRAL, DUPLICATION THEOREM,
             INCOMPLETE ELLIPTIC INTEGRAL, INTEGRAL OF THE THIRD KIND,
             TAYLOR SERIES
***AUTHOR  Carlson, B. C.
             Ames Laboratory-DOE
             Iowa State University
             Ames, IA  50011
           Notis, E. M.
             Ames Laboratory-DOE
             Iowa State University
             Ames, IA  50011
           Pexton, R. L.
             Lawrence Livermore National Laboratory
             Livermore, CA  94550
***DESCRIPTION

   1.     RJ
          Standard FORTRAN function routine
          Single precision version
          The routine calculates an approximation result to
          RJ(X,Y,Z,P) = Integral from zero to infinity of

                                   -1/2     -1/2     -1/2     -1
                      (3/2)(t+X)    (t+Y)    (t+Z)    (t+P)  dt,

          where X, Y, and Z are nonnegative, at most one of them is
          zero, and P is positive.  If X or Y or Z is zero, the
          integral is COMPLETE.  The duplication theorem is iterated
          until the variables are nearly equal, and the function is
          then expanded in Taylor series to fifth order.


   2.     Calling Sequence
          RJ( X, Y, Z, P, IER )

          Parameters On Entry
          Values assigned by the calling routine

          X      - Single precision, nonnegative variable

          Y      - Single precision, nonnegative variable

          Z      - Single precision, nonnegative variable

          P      - Single precision, positive variable
```

On Return (values assigned by the RJ routine)

RJ - Single precision approximation to the integral

IER - Integer

IER = 0 Normal and reliable termination of the routine. It is assumed that the requested accuracy has been achieved.

IER > 0 Abnormal termination of the routine

X, Y, Z, P are unaltered.

3. Error Messages

Value of IER assigned by the RJ routine

Value Assigned	Error Message Printed
IER = 1	MIN(X,Y,Z) .LT. 0.0E0
= 2	MIN(X+Y,X+Z,Y+Z,P) .LT. LOLIM
= 3	MAX(X,Y,Z,P) .GT. UPLIM

4. Control Parameters

Values of LOLIM, UPLIM, and ERRTOL are set by the routine.

LOLIM and UPLIM determine the valid range of X Y, Z, and P

LOLIM is not less than the cube root of the value of LOLIM used in the routine for RC.

UPLIM is not greater than 0.3 times the cube root of the value of UPLIM used in the routine for RC.

Acceptable Values For:	LOLIM	UPLIM
IBM 360/370 SERIES :	2.0E-26	3.0E+24
CDC 6000/7000 SERIES :	5.0E-98	3.0E+106
UNIVAC 1100 SERIES :	5.0E-13	6.0E+11
CRAY :	1.32E-822	1.4E+821
VAX 11 SERIES :	2.5E-13	9.0E+11

ERRTOL determines the accuracy of the answer

The value assigned by the routine will result in solution precision within 1-2 decimals of "machine precision".

Relative error due to truncation of the series for RJ is less than 3 * ERRTOL ** 6 / (1 - ERRTOL) ** 3/2.

The accuracy of the computed approximation to the integral can be controlled by choosing the value of ERRTOL. Truncation of a Taylor series after terms of fifth order Introduces an error less than the amount shown in the second column of the following table for each value of ERRTOL in the first column. In addition to the truncation error there will be round-off error, but in practice the total error from both sources is usually less than the amount given in the table.

Sample choices:

ERRTOL	Relative Truncation error less than
1.0E-3	4.0E-18
3.0E-3	3.0E-15
1.0E-2	4.0E-12
3.0E-2	3.0E-9
1.0E-1	4.0E-6

Decreasing ERRTOL by a factor of 10 yields six more decimal digits of accuracy at the expense of one or two more iterations of the duplication theorem.

*Long Description:

RJ Special Comments

Check by addition theorem: RJ(X,X+Z,X+W,X+P) + RJ(Y,Y+Z,Y+W,Y+P) + (A-B) * RJ(A,B,B,A) + 3 / SQRT(A) = RJ(0,Z,W,P), where X,Y,Z,W,P are positive and X * Y = Z * W, A = P * P * (X+Y+Z+W), B = P * (P+X) * (P+Y), and B - A = P * (P-Z) * (P-W). The sum of the third and fourth terms on the left side is 3 * RC(A,B).

On Input:

X, Y, Z, and P are the variables in the integral RJ(X,Y,Z,P).

On Output:

X, Y, Z, and P are unaltered.

****

Warning: Changes in the program may improve speed at the expense of robustness.

Special Functions via RJ and RF

Legendre form of ELLIPTIC INTEGRAL of 3rd kind

$$P(PHI,K,N) = \int_0^{PHI} (1+N\,SIN^2(THETA))^{-1} * (1-K^2\,SIN^2(THETA))^{-1/2}\,D\,THETA$$

$$= SIN(PHI)\,RF(COS^2(PHI),\ 1-K^2\,SIN^2(PHI),1)$$

$$-(N/3)\,SIN^3(PHI)\,RJ(COS^2(PHI),1-K^2\,SIN^2(PHI),$$

$$1,1+N\,SIN^2(PHI))$$

Bulirsch form of ELLIPTIC INTEGRAL of 3rd kind

$$EL3(X,KC,P) = X\,RF(1,1+KC^2X^2,1+X^2) +$$

$$+(1/3)(1-P)\,X^3\,RJ(1,1+KC^2X^2,1+X^2,1+PX^2)$$

$$CEL(KC,P,A,B) = A\,RF(0,KC^2,1) +$$

$$+(1/3)(B-PA)\,RJ(0,KC^2,1,P)$$

Heuman's LAMBDA function

$$L(A,B,P) = (COS^2(A)SIN(B)COS(B)/(1-COS^2(A)SIN^2(B))^{1/2})$$

$$*(SIN(P)\,RF(COS^2(P),1-SIN^2(A)\,SIN^2(P),1)$$

$$+(SIN^2(A)\,SIN^3(P)/(3(1-COS^2(A)\,SIN^2(B))))$$

```
                                 2          2        2
                            *RJ(COS (P),1-SIN (A) SIN (P),1,1-

                                 2        2           2        2
                            -SIN (A) SIN (P)/(1-COS (A) SIN (B))))



                 (PI/2) LAMBDA0(A,B) =L(A,B,PI/2) =

                   2                            2        2    -1/2
              = COS (A)  SIN(B) COS(B) (1-COS (A) SIN (B))

                            2                    2        2
                 *RF(0,COS (A),1) + (1/3) SIN (A) COS (A)

                                        2        2    -3/2
                 *SIN(B) COS(B) (1-COS (A) SIN (B))

                          2          2        2           2        2
                 *RJ(0,COS (A),1,COS (A) COS (B)/(1-COS (A) SIN (B)))


                 Jacobi ZETA function
                 --------------------


                            2                      2   2    1/2
                 Z(B,K) = (K/3) SIN(B) COS(B) (1-K SIN (B))


                                        2      2   2                2
                            *RJ(0,1-K ,1,1-K SIN (B)) / RF (0,1-K ,1)
```

```
***REFERENCES  B. C. Carlson and E. M. Notis, Algorithms for incomplete
                 elliptic integrals, ACM Transactions on Mathematical
                 Software 7, 3 (September 1981), pp. 398-403.
               B. C. Carlson, Computing elliptic integrals by
                 duplication, Numerische Mathematik 33, (1979),
                 pp. 1-16.
               B. C. Carlson, Elliptic integrals of the first kind,
                 SIAM Journal of Mathematical Analysis 8, (1977),
                 pp. 231-242.
***ROUTINES CALLED  R1MACH, RC, XERMSG
***REVISION HISTORY  (YYMMDD)
   790801  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   891009  Removed unreferenced statement labels.  (WRB)
   891009  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
```

```
900510  Changed calls to XERMSG to standard form, and some
        editorial changes.  (RWC)).
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

RPQR79

```
      SUBROUTINE RPQR79 (NDEG, COEFF, ROOT, IERR, WORK)
***BEGIN PROLOGUE  RPQR79
***PURPOSE  Find the zeros of a polynomial with real coefficients.
***LIBRARY   SLATEC
***CATEGORY  F1A1A
***TYPE      SINGLE PRECISION (RPQR79-S, CPQR79-C)
***KEYWORDS  COMPLEX POLYNOMIAL, POLYNOMIAL ROOTS, POLYNOMIAL ZEROS
***AUTHOR  Vandevender, W. H., (SNLA)
***DESCRIPTION

   Abstract
       This routine computes all zeros of a polynomial of degree NDEG
       with real coefficients by computing the eigenvalues of the
       companion matrix.

   Description of Parameters
       The user must dimension all arrays appearing in the call list
            COEFF(NDEG+1), ROOT(NDEG), WORK(NDEG*(NDEG+2))

    --Input--
      NDEG    degree of polynomial

      COEFF   REAL coefficients in descending order.  i.e.,
              P(Z)= COEFF(1)*(Z**NDEG) + COEFF(NDEG)*Z + COEFF(NDEG+1)

      WORK    REAL work array of dimension at least NDEG*(NDEG+2)

   --Output--
      ROOT    COMPLEX vector of roots

      IERR    Output Error Code
           - Normal Code
          0  means the roots were computed.
           - Abnormal Codes
          1  more than 30 QR iterations on some eigenvalue of the
             companion matrix
          2  COEFF(1)=0.0
          3  NDEG is invalid (less than or equal to 0)

***REFERENCES  (NONE)
***ROUTINES CALLED  HQR, XERMSG
***REVISION HISTORY  (YYMMDD)
   800601  DATE WRITTEN
   890505  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   911010  Code reworked and simplified.  (RWC and WRB)
   END PROLOGUE
```

RPZERO

```
      SUBROUTINE RPZERO (N, A, R, T, IFLG, S)
***BEGIN PROLOGUE  RPZERO
***PURPOSE  Find the zeros of a polynomial with real coefficients.
***LIBRARY   SLATEC
***CATEGORY  F1A1A
***TYPE      SINGLE PRECISION (RPZERO-S, CPZERO-C)
***KEYWORDS  POLYNOMIAL ROOTS, POLYNOMIAL ZEROS, REAL ROOTS
***AUTHOR  Kahaner, D. K., (NBS)
***DESCRIPTION

       Find the zeros of the real polynomial
          P(X)= A(1)*X**N + A(2)*X**(N-1) +...+ A(N+1)

    Input...
       N = degree of P(X)
       A = real vector containing coefficients of P(X),
            A(I) = coefficient of X**(N+1-I)
       R = N word complex vector containing initial estimates for zeros
            if these are known.
       T = 6(N+1) word array used for temporary storage
       IFLG = flag to indicate if initial estimates of
              zeros are input.
            If IFLG .EQ. 0, no estimates are input.
            If IFLG .NE. 0, the vector R contains estimates of
               the zeros
       ** Warning ****** If estimates are input, they must
                         be separated; that is, distinct or
                         not repeated.
       S = an N word array

    Output...
       R(I) = ith zero,
       S(I) = bound for R(I) .
       IFLG = error diagnostic
    Error Diagnostics...
       If IFLG .EQ. 0 on return, all is well.
       If IFLG .EQ. 1 on return, A(1)=0.0 or N=0 on input.
       If IFLG .EQ. 2 on return, the program failed to converge
                after 25*N iterations.  Best current estimates of the
                zeros are in R(I).  Error bounds are not calculated.

***REFERENCES  (NONE)
***ROUTINES CALLED  CPZERO
***REVISION HISTORY  (YYMMDD)
   810223  DATE WRITTEN
   890206  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   END PROLOGUE
```

RS

```
      SUBROUTINE RS (NM, N, A, W, MATZ, Z, FV1, FV2, IERR)
***BEGIN PROLOGUE  RS
***PURPOSE  Compute the eigenvalues and, optionally, the eigenvectors
            of a real symmetric matrix.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4A1
***TYPE      SINGLE PRECISION (RS-S, CH-C)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine calls the recommended sequence of
      subroutines from the eigensystem subroutine package (EISPACK)
      to find the eigenvalues and eigenvectors (if desired)
      of a REAL SYMMETRIC matrix.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameters, A and Z, as declared in the calling
           program dimension statement.  NM is an INTEGER variable.

         N is the order of the matrix A.  N is an INTEGER variable.
           N must be less than or equal to NM.

         A contains the real symmetric matrix.  A is a two-dimensional
           REAL array, dimensioned A(NM,N).

         MATZ is an INTEGER variable set equal to zero if only
           eigenvalues are desired.  Otherwise, it is set to any
           non-zero integer for both eigenvalues and eigenvectors.

      On Output

         A is unaltered.

         W contains the eigenvalues in ascending order.  W is a one-
           dimensional REAL array, dimensioned W(N).

         Z contains the eigenvectors if MATZ is not zero.  The
           eigenvectors are orthonormal.  Z is a two-dimensional
           REAL array, dimensioned Z(NM,N).

         IERR is an INTEGER flag set to
           Zero       for normal return,
           10*N       if N is greater than NM,
           J          if the J-th eigenvalue has not been
                      determined after 30 iterations.
                      The eigenvalues, and eigenvectors if requested,
                      should be correct for indices 1, 2, ..., IERR-1.

         FV1 and FV2 are one-dimensional REAL arrays used for temporary
           storage, dimensioned FV1(N) and FV2(N).

      Questions and comments should be directed to B. S. Garbow,
      APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
```

```
     ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  TQL2, TQLRAT, TRED1, TRED2
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

RSB

```
      SUBROUTINE RSB (NM, N, MB, A, W, MATZ, Z, FV1, FV2, IERR)
***BEGIN PROLOGUE  RSB
***PURPOSE  Compute the eigenvalues and, optionally, the eigenvectors
            of a symmetric band matrix.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4A6
***TYPE      SINGLE PRECISION (RSB-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine calls the recommended sequence of
      subroutines from the eigensystem subroutine package (EISPACK)
      to find the eigenvalues and eigenvectors (if desired)
      of a REAL SYMMETRIC BAND matrix.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameters, A and Z, as declared in the calling
           program dimension statement.  NM is an INTEGER variable.

         N is the order of the matrix A.  N is an INTEGER variable.
           N must be less than or equal to NM.

         MB is the half band width of the matrix, defined as the
           number of adjacent diagonals, including the principal
           diagonal, required to specify the non-zero portion of the
           lower triangle of the matrix.  MB must be less than or
           equal to N.  MB is an INTEGER variable.

         A contains the lower triangle of the real symmetric band
           matrix.  Its lowest subdiagonal is stored in the last
           N+1-MB  positions of the first column, its next subdiagonal
           in the last  N+2-MB  positions of the second column, further
           subdiagonals similarly, and finally its principal diagonal
           in the  N  positions of the last column.  Contents of storage
           locations not part of the matrix are arbitrary.  A is a
           two-dimensional REAL array, dimensioned A(NM,MB).

         MATZ is an INTEGER variable set equal to zero if only
           eigenvalues are desired.  Otherwise, it is set to any
           non-zero integer for both eigenvalues and eigenvectors.

      On Output

         A has been destroyed.

         W contains the eigenvalues in ascending order.  W is a one-
           dimensional REAL array, dimensioned W(N).

         Z contains the eigenvectors if MATZ is not zero.  The
           eigenvectors are orthonormal.  Z is a two-dimensional
           REAL array, dimensioned Z(NM,N).

         IERR is an INTEGER flag set to
```

```
          Zero       for normal return,
          10*N       if N is greater than NM,
          12*N       if MB is either non-positive or greater than N,
          J          if the J-th eigenvalue has not been
                     determined after 30 iterations.
                     The eigenvalues and eigenvectors, if requested,
                     should be correct for indices 1, 2, ..., IERR-1.

        FV1 and FV2 are one-dimensional REAL arrays used for temporary
          storage, dimensioned FV1(N) and FV2(N).

     Questions and comments should be directed to B. S. Garbow,
     APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
     ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  BANDR, TQL2, TQLRAT
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

RSG

```
      SUBROUTINE RSG (NM, N, A, B, W, MATZ, Z, FV1, FV2, IERR)
***BEGIN PROLOGUE  RSG
***PURPOSE  Compute the eigenvalues and, optionally, the eigenvectors
            of a symmetric generalized eigenproblem.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4B1
***TYPE      SINGLE PRECISION (RSG-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine calls the recommended sequence of
      subroutines from the eigensystem subroutine package (EISPACK)
      To find the eigenvalues and eigenvectors (if desired)
      for the REAL SYMMETRIC generalized eigenproblem  Ax = (LAMBDA)Bx.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameters, A, B, and Z, as declared in the calling
           program dimension statement.  NM is an INTEGER variable.

         N is the order of the matrices A and B.  N is an INTEGER
           variable.  N must be less than or equal to NM.

         A contains a real symmetric matrix.  A is a two-dimensional
           REAL array, dimensioned A(NM,N).

         B contains a positive definite real symmetric matrix.  B is a
           two-dimensional REAL array, dimensioned B(NM,N).

         MATZ is an INTEGER variable set equal to zero if only
           eigenvalues are desired.  Otherwise, it is set to any
           non-zero integer for both eigenvalues and eigenvectors.

      On Output

         W contains the eigenvalues in ascending order.  W is a
           one-dimensional REAL array, dimensioned W(N).

         Z contains the eigenvectors if MATZ is not zero.  Z is a
           two-dimensional REAL array, dimensioned Z(NM,N).

         IERR is an INTEGER flag set to
           Zero       for normal return,
           10*N       if N is greater than NM,
           7*N+1      if B is not positive definite,
           J          if the J-th eigenvalue has not been
                      determined after 30 iterations.
                      The eigenvalues should be correct for indices
                      1, 2, ..., IERR-1, but no eigenvectors are
                      computed.

         FV1 and FV2 are one-dimensional REAL arrays used for temporary
           storage, dimensioned FV1(N) and FV2(N).
```

```
     Questions and comments should be directed to B. S. Garbow,
     APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
     ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  REBAK, REDUC, TQL2, TQLRAT, TRED1, TRED2
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

RSGAB

```
      SUBROUTINE RSGAB (NM, N, A, B, W, MATZ, Z, FV1, FV2, IERR)
***BEGIN PROLOGUE  RSGAB
***PURPOSE  Compute the eigenvalues and, optionally, the eigenvectors
            of a symmetric generalized eigenproblem.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4B1
***TYPE      SINGLE PRECISION (RSGAB-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine calls the recommended sequence of
      subroutines from the eigensystem subroutine package (EISPACK)
      to find the eigenvalues and eigenvectors (if desired)
      for the REAL SYMMETRIC generalized eigenproblem  ABx = (LAMBDA)x.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameters, A, B, and Z, as declared in the calling
           program dimension statement.  NM is an INTEGER variable.

         N is the order of the matrices A and B.  N is an INTEGER
           variable.  N must be less than or equal to NM.

         A contains a real symmetric matrix.  A is a two-dimensional
           REAL array, dimensioned A(NM,N).

         B contains a positive definite real symmetric matrix.  B is a
           two-dimensional REAL array, dimensioned B(NM,N).

         MATZ is an INTEGER variable set equal to zero if only
           eigenvalues are desired.  Otherwise, it is set to any
           non-zero integer for both eigenvalues and eigenvectors.

      On Output

         W contains the eigenvalues in ascending order.  W is a
           one-dimensional REAL array, dimensioned W(N).

         Z contains the eigenvectors if MATZ is not zero.  Z is a
           two-dimensional REAL array, dimensioned Z(NM,N).

         IERR is an INTEGER flag set to
           Zero       for normal return,
           10*N       if N is greater than NM,
           7*N+1      if B is not positive definite,
           J          if the J-th eigenvalue has not been
                      determined after 30 iterations.
                      The eigenvalues should be correct for indices
                      1, 2, ..., IERR-1, but no eigenvectors are
                      computed.

         FV1 and FV2 are one-dimensional REAL arrays used for temporary
           storage, dimensioned FV1(N) and FV2(N).
```

```
     Questions and comments should be directed to B. S. Garbow,
     APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
     ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  REBAK, REDUC2, TQL2, TQLRAT, TRED1, TRED2
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

RSGBA

```
      SUBROUTINE RSGBA (NM, N, A, B, W, MATZ, Z, FV1, FV2, IERR)
***BEGIN PROLOGUE  RSGBA
***PURPOSE  Compute the eigenvalues and, optionally, the eigenvectors
            of a symmetric generalized eigenproblem.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4B1
***TYPE      SINGLE PRECISION (RSGBA-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine calls the recommended sequence of
      subroutines from the eigensystem subroutine package (EISPACK)
      to find the eigenvalues and eigenvectors (if desired)
      for the REAL SYMMETRIC generalized eigenproblem  BAx = (LAMBDA)x.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameters, A, B, and Z, as declared in the calling
           program dimension statement.  NM is an INTEGER variable.

         N is the order of the matrices A and B.  N is an INTEGER
           variable.  N must be less than or equal to NM.

         A contains a real symmetric matrix.  A is a two-dimensional
           REAL array, dimensioned A(NM,N).

         B contains a positive definite real symmetric matrix.  B is a
           two-dimensional REAL array, dimensioned B(NM,N).

         MATZ is an INTEGER variable set equal to zero if only
           eigenvalues are desired.  Otherwise, it is set to any
           non-zero integer for both eigenvalues and eigenvectors.

      On Output

         W contains the eigenvalues in ascending order.  W is a
           one-dimensional REAL array, dimensioned W(N).

         Z contains the eigenvectors if MATZ is not zero.  Z is a
           two-dimensional REAL array, dimensioned Z(NM,N).

         IERR is an INTEGER flag set to
           Zero       for normal return,
           10*N       if N is greater than NM,
           7*N+1      if B is not positive definite,
           J          if the J-th eigenvalue has not been
                      determined after 30 iterations.
                      The eigenvalues should be correct for indices
                      1, 2, ..., IERR-1, but no eigenvectors are
                      computed.

         FV1 and FV2 are one-dimensional REAL arrays used for temporary
           storage, dimensioned FV1(N) and FV2(N).
```

```
     Questions and comments should be directed to B. S. Garbow,
     APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
     ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  REBAKB, REDUC2, TQL2, TQLRAT, TRED1, TRED2
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

RSP

```
      SUBROUTINE RSP (NM, N, NV, A, W, MATZ, Z, FV1, FV2, IERR)
***BEGIN PROLOGUE  RSP
***PURPOSE  Compute the eigenvalues and, optionally, the eigenvectors
            of a real symmetric matrix packed into a one dimensional
            array.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4A1
***TYPE      SINGLE PRECISION (RSP-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine calls the recommended sequence of
      subroutines from the eigensystem subroutine package (EISPACK)
      to find the eigenvalues and eigenvectors (if desired)
      of a REAL SYMMETRIC PACKED matrix.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameter, Z, as declared in the calling program
           dimension statement.  NM is an INTEGER variable.

         N is the order of the matrix A.  N is an INTEGER variable.
           N must be less than or equal to NM.

         NV is an INTEGER variable set equal to the dimension of the
           array A as specified in the calling program.  NV must not
           be less than  N*(N+1)/2.

         A contains the lower triangle, stored row-wise, of the real
           symmetric packed matrix.  A is a one-dimensional REAL
           array, dimensioned A(NV).

         MATZ is an INTEGER variable set equal to zero if only
           eigenvalues are desired.  Otherwise, it is set to any
           non-zero integer for both eigenvalues and eigenvectors.

      On Output

         A has been destroyed.

         W contains the eigenvalues in ascending order.  W is a
           one-dimensional REAL array, dimensioned W(N).

         Z contains the eigenvectors if MATZ is not zero.  The eigen-
           vectors are orthonormal.  Z is a two-dimensional REAL array,
           dimensioned Z(NM,N).

         IERR is an INTEGER flag set to
           Zero       for normal return,
           10*N       if N is greater than NM,
           20*N       if NV is less than N*(N+1)/2,
           J          if the J-th eigenvalue has not been
                      determined after 30 iterations.
                      The eigenvalues and eigenvectors in the W and Z
```

```
                    arrays should be correct for indices
                    1, 2, ..., IERR-1.

        FV1 and FV2 are one-dimensional REAL arrays used for temporary
          storage, dimensioned FV1(N) and FV2(N).

     Questions and comments should be directed to B. S. Garbow,
     APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
     ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  TQL2, TQLRAT, TRBAK3, TRED3
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

RST

```
      SUBROUTINE RST (NM, N, W, E, MATZ, Z, IERR)
***BEGIN PROLOGUE  RST
***PURPOSE  Compute the eigenvalues and, optionally, the eigenvectors
            of a real symmetric tridiagonal matrix.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4A5
***TYPE      SINGLE PRECISION (RST-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine calls the recommended sequence of
      subroutines from the eigensystem subroutine package (EISPACK)
      to find the eigenvalues and eigenvectors (if desired)
      of a REAL SYMMETRIC TRIDIAGONAL matrix.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameter, Z, as declared in the calling program
           dimension statement.  NM is an INTEGER variable.

         N is the order of the matrix.  N is an INTEGER variable.
           N must be less than or equal to NM.

         W contains the diagonal elements of the real symmetric
           tridiagonal matrix.  W is a one-dimensional REAL array,
           dimensioned W(N).

         E contains the subdiagonal elements of the matrix in its last
           N-1 positions.  E(1) is arbitrary.  E is a one-dimensional
           REAL array, dimensioned E(N).

         MATZ is an INTEGER variable set equal to zero if only
           eigenvalues are desired.  Otherwise, it is set to any
           non-zero integer for both eigenvalues and eigenvectors.

      On Output

         W contains the eigenvalues in ascending order.

         Z contains the eigenvectors if MATZ is not zero.  The eigen-
           vectors are orthonormal.  Z is a two-dimensional REAL array,
           dimensioned Z(NM,N).

         IERR is an INTEGER flag set to
           Zero       for normal return,
           10*N       if N is greater than NM,
           J          if the J-th eigenvalue has not been
                      determined after 30 iterations.
                      The eigenvalues and eigenvectors in the W and Z
                      arrays should be correct for indices
                      1, 2, ..., IERR-1.

      Questions and comments should be directed to B. S. Garbow,
      APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
```

```
     ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  IMTQL1, IMTQL2
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

RT

```
      SUBROUTINE RT (NM, N, A, W, MATZ, Z, FV1, IERR)
***BEGIN PROLOGUE  RT
***PURPOSE  Compute the eigenvalues and eigenvectors of a special real
            tridiagonal matrix.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4A5
***TYPE      SINGLE PRECISION (RT-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine calls the recommended sequence of subroutines
      from the eigensystem subroutine package (EISPACK) to find the
      eigenvalues and eigenvectors (if desired) of a special REAL
      TRIDIAGONAL matrix.  The property of the matrix required for use
      of this subroutine is that the products of pairs of corresponding
      off-diagonal elements be all non-negative.  If eigenvectors are
      desired, no product can be zero unless both factors are zero.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameter, A and Z, as declared in the calling
           program dimension statement.  NM is an INTEGER variable.

         N is the order of the matrix A.  N is an INTEGER variable.
           N must be less than or equal to NM.

         A contains the special real tridiagonal matrix in its first
           three columns.  The subdiagonal elements are stored in the
           last N-1 positions of the first column, the diagonal elements
           in the second column, and the superdiagonal elements in the
           first N-1 positions of the third column.  Elements A(1,1) and
           A(N,3) are arbitrary.  A is a two-dimensional REAL array,
           dimensioned A(NM,3).

         MATZ is an INTEGER variable set equal to zero if only
           eigenvalues are desired.  Otherwise, it is set to any
           non-zero integer for both eigenvalues and eigenvectors.

      On Output

         W contains the eigenvalues in ascending order.  W is a
           one-dimensional REAL array, dimensioned W(N).

         Z contains the eigenvectors if MATZ is not zero.  The eigen-
           vectors are not normalized.  Z is a two-dimensional REAL
           array, dimensioned Z(NM,N).

         IERR is an INTEGER flag set to
           Zero       for normal return,
           10*N       if N is greater than NM,
           N+J        if A(J,1)*A(J-1,3) is negative,
           2*N+J      if the product is zero with one factor non-zero,
                      and MATZ is non-zero;
           J          if the J-th eigenvalue has not been
```

```
                    determined after 30 iterations.
                    The eigenvalues and eigenvectors in the W and Z
                    arrays should be correct for indices
                    1, 2, ..., IERR-1.

        FV1 is a one-dimensional REAL array used for temporary storage,
          dimensioned FV1(N).

     Questions and comments should be directed to B. S. Garbow,
     APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
     ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  FIGI, FIGI2, IMTQL1, IMTQL2
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

RUNIF

```
      FUNCTION RUNIF (T, N)
***BEGIN PROLOGUE  RUNIF
***PURPOSE  Generate a uniformly distributed random number.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  L6A21
***TYPE      SINGLE PRECISION (RUNIF-S)
***KEYWORDS  FNLIB, RANDOM NUMBER, SPECIAL FUNCTIONS, UNIFORM
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 This random number generator is portable among a wide variety of
 computers.  It generates a random number between 0.0 and 1.0 accord-
 ing to the algorithm presented by Bays and Durham (TOMS, 2, 59,
 1976).  The motivation for using this scheme, which resembles the
 Maclaren-Marsaglia method, is to greatly increase the period of the
 random sequence.  If the period of the basic generator (RAND) is P,
 then the expected mean period of the sequence generated by RUNIF is
 given by   new mean P = SQRT (PI*FACTORIAL(N)/(8*P)),
 where FACTORIAL(N) must be much greater than P in this asymptotic
 formula.  Generally, N should be around 32 if P=4.E6 as for RAND.

             Input Argument --
 N      ABS(N) is the number of random numbers in an auxiliary table.
        Note though that ABS(N)+1 is the number of items in array T.
        If N is positive and differs from its value in the previous
        invocation, then the table is initialized for the new value of
        N.  If N is negative, ABS(N) is the number of items in an
        auxiliary table, but the tables are now assumed already to
        be initialized.  This option enables the user to save the
        table T at the end of a long computer run and to restart with
        the same sequence.  Normally, RUNIF would be called at most
        once with negative N.  Subsequent invocations would have N
        positive and of the correct magnitude.

             Input and Output Argument  --
 T      an array of ABS(N)+1 random numbers from a previous invocation
        of RUNIF.  Whenever N is positive and differs from the old
        N, the table is initialized.  The first ABS(N) numbers are the
        table discussed in the reference, and the N+1 -st value is Y.
        This array may be saved in order to restart a sequence.

             Output Value --
 RUNIF  a random number between 0.0 and 1.0.

***REFERENCES  (NONE)
***ROUTINES CALLED  RAND
***REVISION HISTORY  (YYMMDD)
   770401  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890531  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   910819  Added EXTERNAL statement for RAND due to problem on IBM
           RS 6000.  (WRB)
   END PROLOGUE
```

SASUM

```
      REAL FUNCTION SASUM (N, SX, INCX)
***BEGIN PROLOGUE  SASUM
***PURPOSE  Compute the sum of the magnitudes of the elements of a
            vector.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A3A
***TYPE      SINGLE PRECISION (SASUM-S, DASUM-D, SCASUM-C)
***KEYWORDS  BLAS, LINEAR ALGEBRA, SUM OF MAGNITUDES OF A VECTOR
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
           Kincaid, D. R., (U. of Texas)
           Krogh, F. T., (JPL)
***DESCRIPTION

                B L A S  Subprogram
    Description of Parameters

     --Input--
        N  number of elements in input vector(S)
       SX  single precision vector with N elements
     INCX  storage spacing between elements of SX

     --Output--
    SASUM  single precision result (zero if N .LE. 0)

     Returns sum of magnitudes of single precision SX.
     SASUM = sum from 0 to N-1 of ABS(SX(IX+I*INCX)),
     where IX = 1 if INCX .GE. 0, else IX = 1+(1-N)*INCX.

***REFERENCES  C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T.
                 Krogh, Basic linear algebra subprograms for Fortran
                 usage, Algorithm No. 539, Transactions on Mathematical
                 Software 5, 3 (September 1979), pp. 308-323.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   791001  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900821  Modified to correct problem with a negative increment.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SAXPY

```
      SUBROUTINE SAXPY (N, SA, SX, INCX, SY, INCY)
***BEGIN PROLOGUE  SAXPY
***PURPOSE  Compute a constant times a vector plus a vector.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A7
***TYPE      SINGLE PRECISION (SAXPY-S, DAXPY-D, CAXPY-C)
***KEYWORDS  BLAS, LINEAR ALGEBRA, TRIAD, VECTOR
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
           Kincaid, D. R., (U. of Texas)
           Krogh, F. T., (JPL)
***DESCRIPTION

                B L A S  Subprogram
    Description of Parameters

     --Input--
        N  number of elements in input vector(s)
       SA  single precision scalar multiplier
       SX  single precision vector with N elements
     INCX  storage spacing between elements of SX
       SY  single precision vector with N elements
     INCY  storage spacing between elements of SY

     --Output--
       SY  single precision result (unchanged if N .LE. 0)

     Overwrite single precision SY with single precision SA*SX +SY.
     For I = 0 to N-1, replace  SY(LY+I*INCY) with SA*SX(LX+I*INCX) +
       SY(LY+I*INCY),
     where LX = 1 if INCX .GE. 0, else LX = 1+(1-N)*INCX, and LY is
     defined in a similar way using INCY.

***REFERENCES  C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T.
                 Krogh, Basic linear algebra subprograms for Fortran
                 usage, Algorithm No. 539, Transactions on Mathematical
                 Software 5, 3 (September 1979), pp. 308-323.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   791001  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920310  Corrected definition of LX in DESCRIPTION.  (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SBCG

```
      SUBROUTINE SBCG(N, B, X, NELT, IA, JA, A, ISYM, MATVEC, MTTVEC,
     $     MSOLVE, MTSOLV, ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT,
     $     R, Z, P, RR, ZZ, PP, DZ, RWORK, IWORK)
***BEGIN PROLOGUE  SBCG
***PURPOSE  Preconditioned BiConjugate Gradient Sparse Ax = b Solver.
            Routine to solve a Non-Symmetric linear system  Ax = b
            using the Preconditioned BiConjugate Gradient method.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SBCG-S, DBCG-D)
***KEYWORDS  BICONJUGATE GRADIENT, ITERATIVE PRECONDITION,
             NON-SYMMETRIC LINEAR SYSTEM, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
      INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
      INTEGER ITER, IERR, IUNIT, IWORK(USER DEFINED)
      REAL    B(N), X(N), A(NELT), TOL, ERR, R(N), Z(N), P(N)
      REAL    RR(N), ZZ(N), PP(N), DZ(N)
      REAL    RWORK(USER DEFINED)
      EXTERNAL MATVEC, MTTVEC, MSOLVE, MTSOLV

      CALL SBCG(N, B, X, NELT, IA, JA, A, ISYM, MATVEC, MTTVEC,
     $     MSOLVE, MTSOLV, ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT,
     $     R, Z, P, RR, ZZ, PP, DZ, RWORK, IWORK)

 *Arguments:
 N      :IN       Integer
         Order of the Matrix.
 B      :IN       Real B(N).
         Right-hand side vector.
 X      :INOUT    Real X(N).
         On input X is your initial guess for solution vector.
         On output X is the final approximate solution.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :IN       Integer IA(NELT).
 JA     :IN       Integer JA(NELT).
 A      :IN       Real A(NELT).
         These arrays contain the matrix data structure for A.
         It could take any form.  See "Description", below, for more
         details.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 MATVEC :EXT      External.
         Name of a routine which  performs the matrix vector multiply
         operation  Y = A*X  given A and X.  The  name of  the MATVEC
```

routine must be declared external in the calling program. The calling sequence of MATVEC is:

```
    CALL MATVEC( N, X, Y, NELT, IA, JA, A, ISYM )
```

Where N is the number of unknowns, Y is the product A*X upon return, X is an input vector. NELT, IA, JA, A and ISYM define the SLAP matrix data structure: see Description,below.

MTTVEC :EXT External.
Name of a routine which performs the matrix transpose vector multiply y = A'*X given A and X (where ' denotes transpose). The name of the MTTVEC routine must be declared external in the calling program. The calling sequence to MTTVEC is the same as that for MTTVEC, viz.:

```
    CALL MTTVEC( N, X, Y, NELT, IA, JA, A, ISYM )
```

Where N is the number of unknowns, Y is the product A'*X upon return, X is an input vector. NELT, IA, JA, A and ISYM define the SLAP matrix data structure: see Description,below.

MSOLVE :EXT External.
Name of a routine which solves a linear system MZ = R for Z given R with the preconditioning matrix M (M is supplied via RWORK and IWORK arrays). The name of the MSOLVE routine must be declared external in the calling program. The calling sequence of MSOLVE is:

```
    CALL MSOLVE(N, R, Z, NELT, IA, JA, A, ISYM, RWORK, IWORK)
```

Where N is the number of unknowns, R is the right-hand side vector, and Z is the solution upon return. NELT, IA, JA, A and ISYM define the SLAP matrix data structure: see Description, below. RWORK is a real array that can be used to pass necessary preconditioning information and/or workspace to MSOLVE. IWORK is an integer work array for the same purpose as RWORK.

MTSOLV :EXT External.
Name of a routine which solves a linear system M'ZZ = RR for ZZ given RR with the preconditioning matrix M (M is supplied via RWORK and IWORK arrays). The name of the MTSOLV routine must be declared external in the calling program. The calling sequence to MTSOLV is:

```
   CALL MTSOLV(N, RR, ZZ, NELT, IA, JA, A, ISYM, RWORK, IWORK)
```

Where N is the number of unknowns, RR is the right-hand side vector, and ZZ is the solution upon return. NELT, IA, JA, A and ISYM define the SLAP matrix data structure: see Description, below. RWORK is a real array that can be used to pass necessary preconditioning information and/or workspace to MTSOLV. IWORK is an integer work array for the same purpose as RWORK.

ITOL :IN Integer.
Flag to indicate type of convergence criterion.
If ITOL=1, iteration stops when the 2-norm of the residual divided by the 2-norm of the right-hand side is less than TOL.
If ITOL=2, iteration stops when the 2-norm of M-inv times the residual divided by the 2-norm of M-inv times the right hand side is less than TOL, where M-inv is the inverse of the diagonal of A.
ITOL=11 is often useful for checking and comparing different routines. For this case, the user must supply the "exact" solution or a very accurate approximation (one with an error much less than TOL) through a common block,

```
    COMMON /SSLBLK/ SOLN( )
```

If ITOL=11, iteration stops when the 2-norm of the difference between the iterative approximation and the user-supplied solution divided by the 2-norm of the user-supplied solution

```
        is less than TOL.  Note that this requires the user to set up
        the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine.
        The routine with this declaration should be loaded before the
        stop test so that the correct length is used by the loader.
        This procedure is not standard Fortran and may not work
        correctly on your system (although it has worked on every
        system the authors have tried).  If ITOL is not 11 then this
        common block is indeed standard Fortran.
TOL    :INOUT    Real.
        Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX  :IN       Integer.
        Maximum number of iterations.
ITER   :OUT      Integer.
        Number of iterations required to reach convergence, or
        ITMAX+1 if convergence criterion could not be achieved in
        ITMAX iterations.
ERR    :OUT      Real.
        Error estimate of error in final approximate solution, as
        defined by ITOL.
IERR   :OUT      Integer.
        Return error flag.
          IERR = 0 => All went well.
          IERR = 1 => Insufficient space allocated for WORK or IWORK.
          IERR = 2 => Method failed to converge in ITMAX steps.
          IERR = 3 => Error in user input.
                      Check input values of N, ITOL.
          IERR = 4 => User error tolerance set too tight.
                      Reset to 500*R1MACH(3).  Iteration proceeded.
          IERR = 5 => Preconditioning matrix, M, is not positive
                      definite.  (r,z) < 0.
          IERR = 6 => Matrix A is not positive definite.  (p,Ap) < 0.
IUNIT  :IN       Integer.
        Unit number on which to write the error at each iteration,
        if this is desired for monitoring convergence.  If unit
        number is 0, no writing will occur.
R      :WORK     Real R(N).
Z      :WORK     Real Z(N).
P      :WORK     Real P(N).
RR     :WORK     Real RR(N).
ZZ     :WORK     Real ZZ(N).
PP     :WORK     Real PP(N).
DZ     :WORK     Real DZ(N).
        Real arrays used for workspace.
RWORK  :WORK     Real RWORK(USER DEFINED).
        Real array that can be used for workspace in MSOLVE
        and MTSOLV.
IWORK  :WORK     Integer IWORK(USER DEFINED).
        Integer array that can be used for workspace in MSOLVE
        and MTSOLV.

*Description
     This routine does not care what matrix data structure is used
      for A and M.  It simply calls MATVEC, MTTVEC, MSOLVE, MTSOLV
      routines, with arguments as above.  The user could write any
      type of structure, and  appropriate  MATVEC, MSOLVE, MTTVEC,
      and MTSOLV routines.  It  is assumed that A is stored in the
      IA, JA, A  arrays in some fashion and  that M (or INV(M)) is
      stored  in  IWORK  and  RWORK   in  some fashion.   The SLAP
      routines SSDBCG and SSLUBC are examples of this procedure.
```

Two examples of matrix data structures are the: 1) SLAP Triad format and 2) SLAP Column format.

```
=================== S L A P Triad format ===================
```

In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

```
=================== S L A P Column format ==================
```

In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Cautions:

This routine will attempt to write to the Fortran logical output

```
     unit IUNIT, if IUNIT .ne. 0.  Thus, the user must make sure that
     this logical unit is attached to a file or terminal before calling
     this routine with a non-zero value for IUNIT.  This routine does
     not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO  SSDBCG, SSLUBC
***REFERENCES  1. Mark K. Seager, A SLAP for the Masses, in
                  G. F. Carey, Ed., Parallel Supercomputing: Methods,
                  Algorithms and Applications, Wiley, 1989, pp.135-155.
***ROUTINES CALLED  ISSBCG, R1MACH, SAXPY, SCOPY, SDOT
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890921  Removed TeX from comments.  (FNF)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   891004  Added new reference.
   910411  Prologue converted to Version 4.0 format.  (BAB)
   910502  Removed MATVEC, MTTVEC, MSOLVE, MTSOLV from ROUTINES
           CALLED list.  (FNF)
   920407  COMMON BLOCK renamed SSLBLK.  (WRB)
   920511  Added complete declaration section.  (WRB)
   920929  Corrected format of reference.  (FNF)
   921019  Changed 500.0 to 500 to reduce SP/DP differences.  (FNF)
   921113  Corrected C***CATEGORY line.  (FNF)
   END PROLOGUE
```

SBHIN

```
      SUBROUTINE SBHIN (N, NELT, IA, JA, A, ISYM, SOLN, RHS, IUNIT, JOB)
***BEGIN PROLOGUE  SBHIN
***PURPOSE  Read a Sparse Linear System in the Boeing/Harwell Format.
            The matrix is read in and if the right hand side is also
            present in the input file then it too is read in.  The
            matrix is then modified to be in the SLAP Column format.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  N1
***TYPE      SINGLE PRECISION (SBHIN-S, DBHIN-D)
***KEYWORDS  LINEAR SYSTEM, MATRIX READ, SLAP SPARSE
***AUTHOR  Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, IUNIT, JOB
     REAL    A(NELT), SOLN(N), RHS(N)

     CALL SBHIN( N, NELT, IA, JA, A, ISYM, SOLN, RHS, IUNIT, JOB )

 *Arguments:
 N       :OUT      Integer
          Order of the Matrix.
 NELT    :INOUT    Integer.
          On input NELT is the maximum number of non-zeros that
          can be stored in the IA, JA, A arrays.
          On output NELT is the number of non-zeros stored in A.
 IA      :OUT      Integer IA(NELT).
 JA      :OUT      Integer JA(NELT).
 A       :OUT      Real A(NELT).
          On output these arrays hold the matrix A in the SLAP
          Triad format.  See "Description", below.
 ISYM    :OUT      Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the lower
          triangle of the matrix is stored.
 SOLN    :OUT      Real SOLN(N).
          The solution to the linear system, if present.  This array
          is accessed if and only if JOB is set to read it in, see
          below.  If the user requests that SOLN be read in, but it is
          not in the file, then it is simply zeroed out.
 RHS     :OUT      Real RHS(N).
          The right hand side vector.  This array is accessed if and
          only if JOB is set to read it in, see below.
          If the user requests that RHS be read in, but it is not in
          the file, then it is simply zeroed out.
 IUNIT   :IN       Integer.
          Fortran logical I/O device unit number to read the matrix
          from.  This unit must be connected in a system dependent
          fashion to a file, or you will get a nasty message
          from the Fortran I/O libraries.
 JOB     :INOUT    Integer.
```

```
        Flag indicating what I/O operations to perform.
        On input JOB indicates what Input operations to try to
        perform.
        JOB = 0 => Read only the matrix.
        JOB = 1 => Read matrix and RHS (if present).
        JOB = 2 => Read matrix and SOLN (if present).
        JOB = 3 => Read matrix, RHS and SOLN (if present).
        On output JOB indicates what operations were actually
        performed.
        JOB = -3 => Unable to parse matrix "CODE" from input file
                    to determine if only the lower triangle of matrix
                    is stored.
        JOB = -2 => Number of non-zeros (NELT) too large.
        JOB = -1 => System size (N) too large.
        JOB =  0 => Read in only the matrix.
        JOB =  1 => Read in the matrix and RHS.
        JOB =  2 => Read in the matrix and SOLN.
        JOB =  3 => Read in the matrix, RHS and SOLN.
        JOB = 10 => Read in only the matrix *STRUCTURE*, but no
                    non-zero entries.  Hence, A(*) is not referenced
                    and has the return values the same as the input.
        JOB = 11 => Read in the matrix *STRUCTURE* and RHS.
        JOB = 12 => Read in the matrix *STRUCTURE* and SOLN.
        JOB = 13 => Read in the matrix *STRUCTURE*, RHS and SOLN.

*Description:
      The format for the input is as follows.  The first line contains
      a title to identify the data file.  On the second line (5I4) are
      counters: NLINE, NPLS, NRILS, NNVLS, NRHSLS.
       NLINE  Number of data lines (after the header) in the file.
       NPLS   Number of lines for the Column Pointer data in the file.
       NRILS  Number of lines for the Row indices in the file.
       NNVLS  Number of lines for the Matrix elements in the file.
       NRHSLS Number of lines for the RHS in the file.
      The third line (A3,11X,4I4) contains a symmetry code and some
      additional counters: CODE, NROW, NCOL, NIND, NELE.
      On the fourth line (2A16,2A20) are formats to be used to read
      the following data: PNTFNT, RINFMT, NVLFMT, RHSFMT.
      Following that are the blocks of data in the order indicated.

      ==================== S L A P Triad format ====================
      This routine requires that the  matrix A be   stored in  the
      SLAP  Triad format.  In  this format only the non-zeros  are
      stored.  They may appear in  *ANY* order.  The user supplies
      three arrays of  length NELT, where  NELT is  the number  of
      non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)).  For
      each non-zero the user puts the row and column index of that
      matrix element  in the IA and  JA arrays.  The  value of the
      non-zero  matrix  element is  placed   in  the corresponding
      location of the A array.   This is  an  extremely  easy data
      structure to generate.  On  the  other hand it   is  not too
      efficient on vector computers for  the iterative solution of
      linear systems.  Hence,   SLAP changes   this  input    data
      structure to the SLAP Column format  for  the iteration (but
      does not change it back).

      Here is an example of the  SLAP Triad   storage format for a
      5x5 Matrix.  Recall that the entries may appear in any order.

          5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
```

```
                                 1  2  3  4  5  6  7  8  9 10 11
        |11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
        |21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
        | 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
        | 0  0  0 44  0|
        |51  0 53  0 55|
```

```
 *Portability:
         You must make sure that IUNIT is a valid Fortran logical
         I/O device unit number and that the unit number has been
         associated with a file or the console.  This is a system
         dependent function.

 *Implementation note:
         SOLN is not read by this version.  It will simply be
         zeroed out if JOB = 2 or 3 and the returned value of
         JOB will indicate SOLN has not been read.
***REFERENCES  (NONE)
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   881107  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   911122  Added loop to zero out RHS if user wants to read RHS, but
           it's not in the input file. (MKS)
   911125  Minor improvements to prologue.  (FNF)
   920511  Added complete declaration section.  (WRB)
   921007  Corrected description of input format.  (FNF)
   921208  Added Implementation Note and code to zero out SOLN.  (FNF)
   930701  Updated CATEGORY section.  (FNF, WRB)
   END PROLOGUE
```

SBOCLS

```
      SUBROUTINE SBOCLS (W, MDW, MCON, MROWS, NCOLS, BL, BU, IND, IOPT,
     +   X, RNORMC, RNORM, MODE, RW, IW)
***BEGIN PROLOGUE  SBOCLS
***PURPOSE  Solve the bounded and constrained least squares
            problem consisting of solving the equation
                      E*X = F  (in the least squares sense)
             subject to the linear constraints
                            C*X = Y.
***LIBRARY   SLATEC
***CATEGORY  K1A2A, G2E, G2H1, G2H2
***TYPE      SINGLE PRECISION (SBOCLS-S, DBOCLS-D)
***KEYWORDS  BOUNDS, CONSTRAINTS, INEQUALITY, LEAST SQUARES, LINEAR
***AUTHOR  Hanson, R. J., (SNLA)
***DESCRIPTION

      This subprogram solves the bounded and constrained least squares
      problem. The problem statement is:

      Solve E*X = F (least squares sense), subject to constraints
      C*X=Y.

      In this formulation both X and Y are unknowns, and both may
      have bounds on any of their components.  This formulation
      of the problem allows the user to have equality and inequality
      constraints as well as simple bounds on the solution components.

      This constrained linear least squares subprogram solves E*X=F
      subject to C*X=Y, where E is MROWS by NCOLS, C is MCON by NCOLS.

       The user must have dimension statements of the form

       DIMENSION W(MDW,NCOLS+MCON+1), BL(NCOLS+MCON), BU(NCOLS+MCON),
      * X(2*(NCOLS+MCON)+2+NX), RW(6*NCOLS+5*MCON)
        INTEGER IND(NCOLS+MCON), IOPT(17+NI), IW(2*(NCOLS+MCON))

      (here NX=number of extra locations required for the options; NX=0
      if no options are in use. Also NI=number of extra locations
      for options 1-9.)

     INPUT
     -----

     -------------------------
     W(MDW,*),MCON,MROWS,NCOLS
     -------------------------
      The array W contains the (possibly null) matrix [C:*] followed by
      [E:F].  This must be placed in W as follows:
           [C  :  *]
      W  = [       ]
           [E  :  F]
      The (*) after C indicates that this data can be undefined. The
      matrix [E:F] has MROWS rows and NCOLS+1 columns. The matrix C is
      placed in the first MCON rows of W(*,*) while [E:F]
      follows in rows MCON+1 through MCON+MROWS of W(*,*). The vector F
      is placed in rows MCON+1 through MCON+MROWS, column NCOLS+1. The
      values of MDW and NCOLS must be positive; the value of MCON must
```

be nonnegative. An exception to this occurs when using option 1 for accumulation of blocks of equations. In that case MROWS is an OUTPUT variable only, and the matrix data for [E:F] is placed in W(*,*), one block of rows at a time. See IOPT(*) contents, option number 1, for further details. The row dimension, MDW, of the array W(*,*) must satisfy the inequality:

```
If using option 1,
                MDW .ge. MCON + max(max. number of
                rows accumulated, NCOLS) + 1.
If using option 8,
                MDW .ge. MCON + MROWS.
Else
                MDW .ge. MCON + max(MROWS, NCOLS).
```

Other values are errors, but this is checked only when using option=2. The value of MROWS is an output parameter when using option number 1 for accumulating large blocks of least squares equations before solving the problem.
See IOPT(*) contents for details about option 1.

BL(*),BU(*),IND(*)

These arrays contain the information about the bounds that the solution values are to satisfy. The value of IND(J) tells the type of bound and BL(J) and BU(J) give the explicit values for the respective upper and lower bounds on the unknowns X and Y. The first NVARS entries of IND(*), BL(*) and BU(*) specify bounds on X; the next MCON entries specify bounds on Y.

```
1.    For IND(J)=1, require X(J) .ge. BL(J);
      IF J.gt.NCOLS,        Y(J-NCOLS) .ge. BL(J).
      (the value of BU(J) is not used.)
2.    For IND(J)=2, require X(J) .le. BU(J);
      IF J.gt.NCOLS,        Y(J-NCOLS) .le. BU(J).
      (the value of BL(J) is not used.)
3.    For IND(J)=3, require X(J) .ge. BL(J) and
                            X(J) .le. BU(J);
      IF J.gt.NCOLS,        Y(J-NCOLS) .ge. BL(J) and
                            Y(J-NCOLS) .le. BU(J).
      (to impose equality constraints have BL(J)=BU(J)=
      constraining value.)
4.    For IND(J)=4, no bounds on X(J) or Y(J-NCOLS) are required.
      (the values of BL(J) and BU(J) are not used.)
```

Values other than 1,2,3 or 4 for IND(J) are errors. In the case IND(J)=3 (upper and lower bounds) the condition BL(J) .gt. BU(J) is an error. The values BL(J), BU(J), J .gt. NCOLS, will be changed. Significant changes mean that the constraints are infeasible. (Users must make this decision themselves.)
The new values for BL(J), BU(J), J .gt. NCOLS, define a region such that the perturbed problem is feasible. If users know that their problem is feasible, this step can be skipped by using option number 8 described below.

See IOPT(*) description.

IOPT(*)

This is the array where the user can specify nonstandard options for SBOCLS( ). Most of the time this feature can be ignored by setting the input value IOPT(1)=99. Occasionally users may have needs that require use of the following subprogram options. For details about how to use the options see below: IOPT(*) CONTENTS.

Option Number	Brief Statement of Purpose
1	Return to user for accumulation of blocks of least squares equations. The values of IOPT(*) are changed with this option. The changes are updates to pointers for placing the rows of equations into position for processing.
2	Check lengths of all arrays used in the subprogram.
3	Column scaling of the data matrix, [C]. [E]
4	User provides column scaling for matrix [C]. [E]
5	Provide option array to the low-level subprogram SBOLS( ).
6	Provide option array to the low-level subprogram SBOLSM( ).
7	Move the IOPT(*) processing pointer.
8	Do not preprocess the constraints to resolve infeasibilities.
9	Do not pretriangularize the least squares matrix.
99	No more options to change.

X(*)

This array is used to pass data associated with options 4,5 and 6. Ignore this parameter (on input) if no options are used. Otherwise see below: IOPT(*) CONTENTS.

OUTPUT

X(*),RNORMC,RNORM

The array X(*) contains a solution (if MODE .ge.0 or .eq.-22) for the constrained least squares problem. The value RNORMC is the minimum residual vector length for the constraints C*X - Y = 0. The value RNORM is the minimum residual vector length for the least squares equations. Normally RNORMC=0, but in the case of inconsistent constraints this value will be nonzero.
The values of X are returned in the first NVARS entries of X(*).
The values of Y are returned in the last MCON entries of X(*).

MODE

The sign of MODE determines whether the subprogram has completed normally, or encountered an error condition or abnormal status. A

value of MODE .ge. 0 signifies that the subprogram has completed normally. The value of mode (.ge. 0) is the number of variables in an active status: not at a bound nor at the value zero, for the case of free variables. A negative value of MODE will be one of the cases (-57)-(-41), (-37)-(-22), (-19)-(-2). Values .lt. -1 correspond to an abnormal completion of the subprogram. These error messages are in groups for the subprograms SBOCLS(), SBOLSM(), and SBOLS(). An approximate solution will be returned to the user only when max. iterations is reached, MODE=-22.

RW(*),IW(*)

These are working arrays. (normally the user can ignore the contents of these arrays.)

IOPT(*) CONTENTS
--- ---

The option array allows a user to modify some internal variables in the subprogram without recompiling the source code. A central goal of the initial software design was to do a good job for most people. Thus the use of options will be restricted to a select group of users. The processing of the option array proceeds as follows: a pointer, here called LP, is initially set to the value 1. At the pointer position the option number is extracted and used for locating other information that allows for options to be changed. The portion of the array IOPT(*) that is used for each option is fixed; the user and the subprogram both know how many locations are needed for each option. The value of LP is updated for each option based on the amount of storage in IOPT(*) that is required. A great deal of error checking is done by the subprogram on the contents of the option array. Nevertheless it is still possible to give the subprogram optional input that is meaningless. For example option 4 uses the locations X(NCOLS+IOFF),...,X(NCOLS+IOFF+NCOLS-1) for passing scaling data. The user must manage the allocation of these locations.

1
-

This option allows the user to solve problems with a large number of rows compared to the number of variables. The idea is that the subprogram returns to the user (perhaps many times) and receives new least squares equations from the calling program unit. Eventually the user signals "that's all" and a solution is then computed. The value of MROWS is an output variable when this option is used. Its value is always in the range 0 .le. MROWS .le. NCOLS+1. It is the number of rows after the triangularization of the entire set of equations. If LP is the processing pointer for IOPT(*), the usage for the sequential processing of blocks of equations is

```
IOPT(LP)=1
 Move block of equations to W(*,*) starting at
 the first row of W(*,*).
IOPT(LP+3)=# of rows in the block; user defined
```

The user now calls SBOCLS( ) in a loop. The value of IOPT(LP+1) directs the user's action. The value of IOPT(LP+2) points to where the subsequent rows are to be placed in W(*,*). Both of

these values are first defined in the subprogram. The user changes the value of IOPT(LP+1) (to 2) as a signal that all of the rows have been processed.

```
 .<LOOP
 . CALL SBOCLS( )
 . IF(IOPT(LP+1) .EQ. 1) THEN
 .    IOPT(LP+3)=# OF ROWS IN THE NEW BLOCK; USER DEFINED
 .    PLACE NEW BLOCK OF IOPT(LP+3) ROWS IN
 .    W(*,*) STARTING AT ROW MCON + IOPT(LP+2).
 .
 .    IF( THIS IS THE LAST BLOCK OF EQUATIONS ) THEN
 .       IOPT(LP+1)=2
 .<------CYCLE LOOP
 .    ELSE IF (IOPT(LP+1) .EQ. 2) THEN
<-------EXIT LOOP SOLUTION COMPUTED IF MODE .GE. 0
 . ELSE
 . ERROR CONDITION; SHOULD NOT HAPPEN.
 .<END LOOP
```

Use of this option adds 4 to the required length of IOPT(*).

2
-

This option is useful for checking the lengths of all arrays used by SBOCLS( ) against their actual requirements for this problem. The idea is simple: the user's program unit passes the declared dimension information of the arrays. These values are compared against the problem-dependent needs within the subprogram. If any of the dimensions are too small an error message is printed and a negative value of MODE is returned, -41 to -47. The printed error message tells how long the dimension should be. If LP is the processing pointer for IOPT(*),

```
IOPT(LP)=2
IOPT(LP+1)=Row dimension of W(*,*)
IOPT(LP+2)=Col. dimension of W(*,*)
IOPT(LP+3)=Dimensions of BL(*),BU(*),IND(*)
IOPT(LP+4)=Dimension of X(*)
IOPT(LP+5)=Dimension of RW(*)
IOPT(LP+6)=Dimension of IW(*)
IOPT(LP+7)=Dimension of IOPT(*)
 .
CALL SBOCLS( )
```

Use of this option adds 8 to the required length of IOPT(*).

3
-

This option can change the type of scaling for the data matrix. Nominally each nonzero column of the matrix is scaled so that the magnitude of its largest entry is equal to the value ONE. If LP is the processing pointer for IOPT(*),

```
IOPT(LP)=3
IOPT(LP+1)=1,2 or 3
    1= Nominal scaling as noted;
    2= Each nonzero column scaled to have length ONE;
    3= Identity scaling; scaling effectively suppressed.
```

```
     .
   CALL SBOCLS( )
```

Use of this option adds 2 to the required length of IOPT(*).

4
-

This options allows the user to provide arbitrary (positive) column scaling for the matrix. If LP is the processing pointer for IOPT(*),

```
   IOPT(LP)=4
   IOPT(LP+1)=IOFF
   X(NCOLS+IOFF),...,X(NCOLS+IOFF+NCOLS-1)
   = Positive scale factors for cols. of E.
     .
   CALL SBOCLS( )
```

Use of this option adds 2 to the required length of IOPT(*) and NCOLS to the required length of X(*).

5
-

This option allows the user to provide an option array to the low-level subprogram SBOLS( ). If LP is the processing pointer for IOPT(*),

```
   IOPT(LP)=5
   IOPT(LP+1)= Position in IOPT(*) where option array
               data for SBOLS( ) begins.
     .
   CALL SBOCLS( )
```

Use of this option adds 2 to the required length of IOPT(*).

6
-

This option allows the user to provide an option array to the low-level subprogram SBOLSM( ). If LP is the processing pointer for IOPT(*),

```
   IOPT(LP)=6
   IOPT(LP+1)= Position in IOPT(*) where option array
               data for SBOLSM( ) begins.
     .
   CALL SBOCLS( )
```

Use of this option adds 2 to the required length of IOPT(*).

7
-

Move the processing pointer (either forward or backward) to the location IOPT(LP+1). The processing pointer moves to locations LP+2 if option number 7 is used with the value -7. For example to skip over locations 3,...,NCOLS+2,

```
  IOPT(1)=7
  IOPT(2)=NCOLS+3
  (IOPT(I), I=3,...,NCOLS+2 are not defined here.)
  IOPT(NCOLS+3)=99
```

```
      CALL SBOCLS( )
```

CAUTION: Misuse of this option can yield some very hard-to-find bugs. Use it with care. It is intended to be used for passing option arrays to other subprograms.

8
-

This option allows the user to suppress the algorithmic feature of SBOCLS( ) that processes the constraint equations C*X = Y and resolves infeasibilities. The steps normally done are to solve C*X - Y = 0 in a least squares sense using the stated bounds on both X and Y. Then the "reachable" vector Y = C*X is computed using the solution X obtained. Finally the stated bounds for Y are enlarged to include C*X. To suppress the feature:

```
      IOPT(LP)=8
        .
      CALL SBOCLS( )
```

Use of this option adds 1 to the required length of IOPT(*).

9
-

This option allows the user to suppress the pretriangularizing step of the least squares matrix that is done within SBOCLS( ). This is primarily a means of enhancing the subprogram efficiency and has little effect on accuracy. To suppress the step, set:

```
      IOPT(LP)=9
        .
      CALL SBOCLS( )
```

Use of this option adds 1 to the required length of IOPT(*).

99
--

There are no more options to change.

Only option numbers -99, -9,-8,...,-1, 1,2,...,9, and 99 are permitted. Other values are errors. Options -99,-1,...,-9 mean that the respective options 99,1,...,9 are left at their default values. An example is the option to suppress the preprocessing of constraints:

```
      IOPT(1)=-8 Option is recognized but not changed
      IOPT(2)=99
      CALL SBOCLS( )
```

Error Messages for SBOCLS()
----- -------- --- --------

```
WARNING in...
SBOCLS(). THE ROW DIMENSION OF W(,)=(I1) MUST BE .GE. THE NUMBER
OF EFFECTIVE ROWS=(I2).
          IN ABOVE MESSAGE, I1=         1
          IN ABOVE MESSAGE, I2=         2
ERROR NUMBER =        41
```

```
WARNING IN...
SBOCLS(). THE COLUMN DIMENSION OF W(,)=(I1) MUST BE .GE. NCOLS+
MCON+1=(I2).
          IN ABOVE MESSAGE, I1=         2
          IN ABOVE MESSAGE, I2=         3
ERROR NUMBER =        42

WARNING IN...
SBOCLS(). THE DIMENSIONS OF THE ARRAYS BL(),BU(), AND IND()=(I1)
MUST BE .GE. NCOLS+MCON=(I2).
          IN ABOVE MESSAGE, I1=         1
          IN ABOVE MESSAGE, I2=         2
ERROR NUMBER =        43

WARNING IN...
SBOCLS(). THE DIMENSION OF X()=(I1) MUST BE
.GE. THE REQD.LENGTH=(I2).
          IN ABOVE MESSAGE, I1=         1
          IN ABOVE MESSAGE, I2=         2
ERROR NUMBER =        44

WARNING IN...
SBOCLS(). THE .
SBOCLS() THE DIMENSION OF IW()=(I1) MUST BE .GE. 2*NCOLS+2*MCON=(I2).
          IN ABOVE MESSAGE, I1=         1
          IN ABOVE MESSAGE, I2=         4
ERROR NUMBER =        46

WARNING IN...
SBOCLS(). THE DIMENSION OF IOPT()=(I1) MUST BE .GE. THE REQD.
LEN.=(I2).
          IN ABOVE MESSAGE, I1=        16
          IN ABOVE MESSAGE, I2=        18
ERROR NUMBER =        47

WARNING IN...
SBOCLS(). ISCALE OPTION=(I1) MUST BE 1-3.
          IN ABOVE MESSAGE, I1=         0
ERROR NUMBER =        48

WARNING IN...
SBOCLS(). OFFSET PAST X(NCOLS) (I1) FOR USER-PROVIDED COLUMN SCALING
MUST BE POSITIVE.
          IN ABOVE MESSAGE, I1=         0
ERROR NUMBER =        49

WARNING IN...
SBOCLS(). EACH PROVIDED COL. SCALE FACTOR MUST BE POSITIVE.
 COMPONENT (I1) NOW = (R1).
          IN ABOVE MESSAGE, I1=         1
          IN ABOVE MESSAGE, R1=    0.
ERROR NUMBER =        50

WARNING IN...
SBOCLS(). THE OPTION NUMBER=(I1) IS NOT DEFINED.
          IN ABOVE MESSAGE, I1=      1001
ERROR NUMBER =        51

WARNING IN...
SBOCLS(). NO. OF ROWS=(I1) MUST BE .GE. 0 .AND. .LE. MDW-MCON=(I2).
```

```
          IN ABOVE MESSAGE, I1=         2
          IN ABOVE MESSAGE, I2=         1
 ERROR NUMBER =        52

 WARNING IN...
 SBOCLS(). MDW=(I1) MUST BE POSITIVE.
          IN ABOVE MESSAGE, I1=         0
 ERROR NUMBER =        53

 WARNING IN...
 SBOCLS(). MCON=(I1) MUST BE NONNEGATIVE.
          IN ABOVE MESSAGE, I1=        -1
 ERROR NUMBER =        54

 WARNING IN...
 SBOCLS(). NCOLS=(I1) THE NO. OF VARIABLES MUST BE POSITIVE.
          IN ABOVE MESSAGE, I1=         0
 ERROR NUMBER =        55

 WARNING IN...
 SBOCLS(). FOR J=(I1), IND(J)=(I2) MUST BE 1-4.
          IN ABOVE MESSAGE, I1=         1
          IN ABOVE MESSAGE, I2=         0
 ERROR NUMBER =        56

 WARNING IN...
 SBOCLS(). FOR J=(I1), BOUND BL(J)=(R1) IS .GT. BU(J)=(R2).
          IN ABOVE MESSAGE, I1=         1
          IN ABOVE MESSAGE, R1=     .1000000000E+01
          IN ABOVE MESSAGE, R2=    0.
 ERROR NUMBER =        57
          LINEAR CONSTRAINTS, SNLA REPT. SAND82-1517, AUG. (1982).

***REFERENCES  R. J. Hanson, Linear least squares with bounds and
                linear constraints, Report SAND82-1517, Sandia
                Laboratories, August 1982.
***ROUTINES CALLED  R1MACH, SASUM, SBOLS, SCOPY, SDOT, SNRM2, SSCAL,
                    XERMSG
***REVISION HISTORY  (YYMMDD)
   821220  DATE WRITTEN
   870803  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900510  Convert XERRWV calls to XERMSG calls.  (RWC)
   910819  Added variable M for MOUT+MCON in reference to SBOLS.  (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SBOLS

```
      SUBROUTINE SBOLS (W, MDW, MROWS, NCOLS, BL, BU, IND, IOPT, X,
     +   RNORM, MODE, RW, IW)
***BEGIN PROLOGUE  SBOLS
***PURPOSE  Solve the problem
                 E*X = F (in the least  squares  sense)
            with bounds on selected X values.
***LIBRARY   SLATEC
***CATEGORY  K1A2A, G2E, G2H1, G2H2
***TYPE      SINGLE PRECISION (SBOLS-S, DBOLS-D)
***KEYWORDS  BOUNDS, CONSTRAINTS, INEQUALITY, LEAST SQUARES, LINEAR
***AUTHOR  Hanson, R. J., (SNLA)
***DESCRIPTION

      The user must have dimension statements of the form:

        DIMENSION W(MDW,NCOLS+1), BL(NCOLS), BU(NCOLS),
       * X(NCOLS+NX), RW(5*NCOLS)
        INTEGER IND(NCOLS), IOPT(1+NI), IW(2*NCOLS)

      (here NX=number of extra locations required for option 4; NX=0
      for no options; NX=NCOLS if this option is in use. Here NI=number
      of extra locations required for options 1-6; NI=0 for no
      options.)

    INPUT
    -----

     --------------------
     W(MDW,*),MROWS,NCOLS
     --------------------
      The array W(*,*) contains the matrix [E:F] on entry. The matrix
      [E:F] has MROWS rows and NCOLS+1 columns. This data is placed in
      the array W(*,*) with E occupying the first NCOLS columns and the
      right side vector F in column NCOLS+1. The row dimension, MDW, of
      the array W(*,*) must satisfy the inequality MDW .ge. MROWS.
      Other values of MDW are errors. The values of MROWS and NCOLS
      must be positive. Other values are errors. There is an exception
      to this when using option 1 for accumulation of blocks of
      equations. In that case MROWS is an OUTPUT variable ONLY, and the
      matrix data for [E:F] is placed in W(*,*), one block of rows at a
      time.  MROWS contains the number of rows in the matrix after
      triangularizing several blocks of equations. This is an OUTPUT
      parameter ONLY when option 1 is used. See IOPT(*) CONTENTS
      for details about option 1.

     ------------------
     BL(*),BU(*),IND(*)
     ------------------
      These arrays contain the information about the bounds that the
      solution values are to satisfy. The value of IND(J) tells the
      type of bound and BL(J) and BU(J) give the explicit values for
      the respective upper and lower bounds.

     1.    For IND(J)=1, require X(J) .ge. BL(J).
           (the value of BU(J) is not used.)
     2.    For IND(J)=2, require X(J) .le. BU(J).
```

```
      (the value of BL(J) is not used.)
3.    For IND(J)=3, require X(J) .ge. BL(J) and
                            X(J) .le. BU(J).
4.    For IND(J)=4, no bounds on X(J) are required.
      (the values of BL(J) and BU(J) are not used.)
```

Values other than 1,2,3 or 4 for IND(J) are errors. In the case IND(J)=3 (upper and lower bounds) the condition BL(J) .gt. BU(J) is an error.

IOPT(*)

This is the array where the user can specify nonstandard options for SBOLSM( ). Most of the time this feature can be ignored by setting the input value IOPT(1)=99. Occasionally users may have needs that require use of the following subprogram options. For details about how to use the options see below: IOPT(*) CONTENTS.

Option Number	Brief Statement of Purpose
1	Return to user for accumulation of blocks of least squares equations.
2	Check lengths of all arrays used in the subprogram.
3	Standard scaling of the data matrix, E.
4	User provides column scaling for matrix E.
5	Provide option array to the low-level subprogram SBOLSM( ).
6	Move the IOPT(*) processing pointer.
99	No more options to change.

X(*)

This array is used to pass data associated with option 4. Ignore this parameter if this option is not used. Otherwise see below: IOPT(*) CONTENTS.

OUTPUT

X(*),RNORM

The array X(*) contains a solution (if MODE .ge.0 or .eq.-22) for the constrained least squares problem. The value RNORM is the minimum residual vector length.

MODE

The sign of MODE determines whether the subprogram has completed normally, or encountered an error condition or abnormal status. A value of MODE .ge. 0 signifies that the subprogram has completed normally. The value of MODE (.GE. 0) is the number of variables in an active status: not at a bound nor at the value ZERO, for the case of free variables. A negative value of MODE will be one of the cases -37,-36,...,-22, or -17,...,-2. Values .lt. -1 correspond to an abnormal completion of the subprogram. To

understand the abnormal completion codes see below: ERROR MESSAGES for SBOLS( ). AN approximate solution will be returned to the user only when max. iterations is reached, MODE=-22. Values for MODE=-37,...,-22 come from the low-level subprogram SBOLSM(). See the section ERROR MESSAGES for SBOLSM() in the documentation for SBOLSM().

RW(*),IW(*)

These are working arrays with 5*NCOLS and 2*NCOLS entries. (normally the user can ignore the contents of these arrays, but they must be dimensioned properly.)

IOPT(*) CONTENTS
--- ---

The option array allows a user to modify internal variables in the subprogram without recompiling the source code. A central goal of the initial software design was to do a good job for most people. Thus the use of options will be restricted to a select group of users. The processing of the option array proceeds as follows: a pointer, here called LP, is initially set to the value 1. This value is updated as each option is processed. At the pointer position the option number is extracted and used for locating other information that allows for options to be changed. The portion of the array IOPT(*) that is used for each option is fixed; the user and the subprogram both know how many locations are needed for each option. A great deal of error checking is done by the subprogram on the contents of the option array. Nevertheless it is still possible to give the subprogram optional input that is meaningless. For example option 4 uses the locations X(NCOLS+IOFF),...,X(NCOLS+IOFF+NCOLS-1) for passing scaling data. The user must manage the allocation of these locations.

1
-

This option allows the user to solve problems with a large number of rows compared to the number of variables. The idea is that the subprogram returns to the user (perhaps many times) and receives new least squares equations from the calling program unit. Eventually the user signals "that's all" and then computes the solution with one final call to subprogram SBOLS( ). The value of MROWS is an OUTPUT variable when this option is used. Its value is always in the range 0 .le. MROWS .le. NCOLS+1. It is equal to the number of rows after the triangularization of the entire set of equations. If LP is the processing pointer for IOPT(*), the usage for the sequential processing of blocks of equations is

```
IOPT(LP)=1
Move block of equations to W(*,*) starting at
the first row of W(*,*).
IOPT(LP+3)=# of rows in the block; user defined
```

The user now calls SBOLS( ) in a loop. The value of IOPT(LP+1) directs the user's action. The value of IOPT(LP+2) points to where the subsequent rows are to be placed in W(*,*).

```
.<LOOP
. CALL SBOLS()
```

```
. IF(IOPT(LP+1) .EQ. 1) THEN
.    IOPT(LP+3)=# OF ROWS IN THE NEW BLOCK; USER DEFINED
.    PLACE NEW BLOCK OF IOPT(LP+3) ROWS IN
.    W(*,*) STARTING AT ROW IOPT(LP+2).
.
.    IF( THIS IS THE LAST BLOCK OF EQUATIONS ) THEN
.       IOPT(LP+1)=2
.<------CYCLE LOOP
.    ELSE IF (IOPT(LP+1) .EQ. 2) THEN
<-------EXIT LOOP SOLUTION COMPUTED IF MODE .GE. 0
. ELSE
. ERROR CONDITION; SHOULD NOT HAPPEN.
.<END LOOP
```

Use of this option adds 4 to the required length of IOPT(*).

2
-

This option is useful for checking the lengths of all arrays used by SBOLS() against their actual requirements for this problem. The idea is simple: the user's program unit passes the declared dimension information of the arrays. These values are compared against the problem-dependent needs within the subprogram. If any of the dimensions are too small an error message is printed and a negative value of MODE is returned, -11 to -17. The printed error message tells how long the dimension should be. If LP is the processing pointer for IOPT(*),

```
IOPT(LP)=2
IOPT(LP+1)=Row dimension of W(*,*)
IOPT(LP+2)=Col. dimension of W(*,*)
IOPT(LP+3)=Dimensions of BL(*),BU(*),IND(*)
IOPT(LP+4)=Dimension of X(*)
IOPT(LP+5)=Dimension of RW(*)
IOPT(LP+6)=Dimension of IW(*)
IOPT(LP+7)=Dimension of IOPT(*)
 .
CALL SBOLS()
```

Use of this option adds 8 to the required length of IOPT(*).

3
-

This option changes the type of scaling for the data matrix E. Nominally each nonzero column of E is scaled so that the magnitude of its largest entry is equal to the value ONE. If LP is the processing pointer for IOPT(*),

```
IOPT(LP)=3
IOPT(LP+1)=1,2 or 3
    1= Nominal scaling as noted;
    2= Each nonzero column scaled to have length ONE;
    3= Identity scaling; scaling effectively suppressed.
 .
CALL SBOLS()
```

Use of this option adds 2 to the required length of IOPT(*).

4

-

This option allows the user to provide arbitrary (positive) column scaling for the matrix E. If LP is the processing pointer for IOPT(*),

```
IOPT(LP)=4
IOPT(LP+1)=IOFF
X(NCOLS+IOFF),...,X(NCOLS+IOFF+NCOLS-1)
= Positive scale factors for cols. of E.
 .
CALL SBOLS()
```

Use of this option adds 2 to the required length of IOPT(*) and NCOLS to the required length of X(*).

5
-

This option allows the user to provide an option array to the low-level subprogram SBOLSM(). If LP is the processing pointer for IOPT(*),

```
IOPT(LP)=5
IOPT(LP+1)= Position in IOPT(*) where option array
            data for SBOLSM() begins.
 .
CALL SBOLS()
```

Use of this option adds 2 to the required length of IOPT(*).

6
-

Move the processing pointer (either forward or backward) to the location IOPT(LP+1). The processing point is moved to entry LP+2 of IOPT(*) if the option is left with -6 in IOPT(LP). For example to skip over locations 3,...,NCOLS+2 of IOPT(*),

```
IOPT(1)=6
IOPT(2)=NCOLS+3
(IOPT(I), I=3,...,NCOLS+2 are not defined here.)
IOPT(NCOLS+3)=99
CALL SBOLS()
```

CAUTION: Misuse of this option can yield some very hard -to-find bugs. Use it with care.

99
--

There are no more options to change.

Only option numbers -99, -6,-5,...,-1, 1,2,...,6, and 99 are permitted. Other values are errors. Options -99,-1,...,-6 mean that the respective options 99,1,...,6 are left at their default values. An example is the option to modify the (rank) tolerance:

```
IOPT(1)=-3 Option is recognized but not changed
IOPT(2)=2  Scale nonzero cols. to have length ONE
IOPT(3)=99
```

ERROR MESSAGES for SBOLS()

```
WARNING IN...
SBOLS(). MDW=(I1) MUST BE POSITIVE.
          IN ABOVE MESSAGE, I1=         0
ERROR NUMBER =         2
(NORMALLY A RETURN TO THE USER TAKES PLACE FOLLOWING THIS MESSAGE.)

WARNING IN...
SBOLS(). NCOLS=(I1) THE NO. OF VARIABLES MUST BE POSITIVE.
          IN ABOVE MESSAGE, I1=         0
ERROR NUMBER =         3
(NORMALLY A RETURN TO THE USER TAKES PLACE FOLLOWING THIS MESSAGE.)

WARNING IN...
SBOLS(). FOR J=(I1), IND(J)=(I2) MUST BE 1-4.
          IN ABOVE MESSAGE, I1=         1
          IN ABOVE MESSAGE, I2=         0
ERROR NUMBER =         4
(NORMALLY A RETURN TO THE USER TAKES PLACE FOLLOWING THIS MESSAGE.)

WARNING IN...
SBOLS(). FOR J=(I1), BOUND BL(J)=(R1) IS .GT. BU(J)=(R2).
          IN ABOVE MESSAGE, I1=         1
          IN ABOVE MESSAGE, R1=    0.
          IN ABOVE MESSAGE, R2=    ABOVE MESSAGE, I1=         0
ERROR NUMBER =         6
(NORMALLY A RETURN TO THE USER TAKES PLACE FOLLOWING THIS MESSAGE.)

WARNING IN...
SBOLS(). ISCALE OPTION=(I1) MUST BE 1-3.
          IN ABOVE MESSAGE, I1=         0
ERROR NUMBER =         7
(NORMALLY A RETURN TO THE USER TAKES PLACE FOLLOWING THIS MESSAGE.)

WARNING IN...
SBOLS(). OFFSET PAST X(NCOLS) (I1) FOR USER-PROVIDED  COLUMN SCALING
MUST BE POSITIVE.
          IN ABOVE MESSAGE, I1=         0
ERROR NUMBER =         8
(NORMALLY A RETURN TO THE USER TAKES PLACE FOLLOWING THIS MESSAGE.)

WARNING IN...
SBOLS(). EACH PROVIDED COL. SCALE FACTOR MUST BE POSITIVE.
COMPONENT (I1) NOW = (R1).
          IN ABOVE MESSAGE, I1=        ND. .LE. MDW=(I2).
          IN ABOVE MESSAGE, I1=         1
          IN ABOVE MESSAGE, I2=         0
ERROR NUMBER =        10
(NORMALLY A RETURN TO THE USER TAKES PLACE FOLLOWING THIS MESSAGE.)

WARNING IN...
SBOLS().THE ROW DIMENSION OF W(,)=(I1) MUST BE .GE.THE NUMBER OF ROWS=
(I2).
          IN ABOVE MESSAGE, I1=         0
          IN ABOVE MESSAGE, I2=         1
ERROR NUMBER =        11
(NORMALLY A RETURN TO THE USER TAKES PLACE FOLLOWING THIS MESSAGE.)

WARNING IN...
SBOLS(). THE COLUMN DIMENSION OF W(,)=(I1) MUST BE .GE. NCOLS+1=(I2).
```

```
          IN ABOVE MESSAGE, I1=         0
          IN ABOVE MESSAGE, I2=         2
 ERROR NUMBER =        12
 (NORMALLY A RETURN TO THE USER TAKES PLACE FOLLOWING THIS MESSAGE.)

 WARNING IN...
 SBOLS().THE DIMENSIONS OF THE ARRAYS BL(),BU(), AND IND()=(I1) MUST BE
 .GE. NCOLS=(I2).
          IN ABOVE MESSAGE, I1=         0
          IN ABOVE MESSAGE, I2=         1
 ERROR NUMBER =        13
 (NORMALLY A RETURN TO THE USER TAKES PLACE FOLLOWING THIS MESSAGE.)

 WARNING IN...
 SBOLS(). THE DIMENSION OF X()=(I1) MUST BE .GE. THE REQD. LENGTH=(I2).
          IN ABOVE MESSAGE, I1=         0
          IN ABOVE MESSAGE, I2=         2
 ERROR NUMBER =        14
 (NORMALLY A RETURN TO THE USER TAKES PLACE FOLLOWING THIS MESSAGE.)

 WARNING IN...
 SBOLS(). THE DIMENSION OF RW()=(I1) MUST BE .GE. 5*NCOLS=(I2).
          IN ABOVE MESSAGE, I1=         0
          IN ABOVE MESSAGE, I2=         3
 ERROR NUMBER =        15
 (NORMALLY A RETURN TO THE USER TAKES PLACE FOLLOWING THIS MESSAGE.)

 WARNING IN...
 SBOLS() THE DIMENSION OF IW()=(I1) MUST BE .GE. 2*NCOLS=(I2).
          IN ABOVE MESSAGE, I1=         0
          IN ABOVE MESSAGE, I2=         2
 ERROR NUMBER =        16
 (NORMALLY A RETURN TO THE USER TAKES PLACE FOLLOWING THIS MESSAGE.)

 WARNING IN...
 SBOLS() THE DIMENSION OF IOPT()=(I1) MUST BE .GE. THE REQD. LEN.=(I2).
          IN ABOVE MESSAGE, I1=         0
          IN ABOVE MESSAGE, I2=         1
 ERROR NUMBER =        17
 (NORMALLY A RETURN TO THE USER TAKES PLACE FOLLOWING THIS MESSAGE.)

***REFERENCES  R. J. Hanson, Linear least squares with bounds and
                 linear constraints, Report SAND82-1517, Sandia
                 Laboratories, August 1982.
***ROUTINES CALLED  ISAMAX, SBOLSM, SCOPY, SNRM2, SROT, SROTG, XERMSG
***REVISION HISTORY  (YYMMDD)
   821220  DATE WRITTEN
   861211  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900510  Convert XERRWV calls to XERMSG calls.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SCASUM

```
      FUNCTION SCASUM (N, CX, INCX)
***BEGIN PROLOGUE  SCASUM
***PURPOSE  Compute the sum of the magnitudes of the real and
            imaginary elements of a complex vector.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A3A
***TYPE      COMPLEX (SASUM-S, DASUM-D, SCASUM-C)
***KEYWORDS  BLAS, LINEAR ALGEBRA, SUM OF MAGNITUDES OF A VECTOR
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
           Kincaid, D. R., (U. of Texas)
           Krogh, F. T., (JPL)
***DESCRIPTION

                B L A S  Subprogram
    Description of Parameters

     --Input--
        N  number of elements in input vector(s)
       CX  complex vector with N elements
     INCX  storage spacing between elements of CX

     --Output--
   SCASUM  single precision result (zero if N .LE. 0)

     Returns sums of magnitudes of real and imaginary parts of
     components of CX.  Note that this is not the L1 norm of CX.
     CASUM = sum from 0 to N-1 of ABS(REAL(CX(IX+I*INCX))) +
             ABS(IMAG(CX(IX+I*INCX))),
     where IX = 1 if INCX .GE. 0, else IX = 1+(1-N)*INCX.

***REFERENCES  C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T.
                 Krogh, Basic linear algebra subprograms for Fortran
                 usage, Algorithm No. 539, Transactions on Mathematical
                 Software 5, 3 (September 1979), pp. 308-323.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   791001  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900821  Modified to correct problem with a negative increment.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SCG

```
      SUBROUTINE SCG(N, B, X, NELT, IA, JA, A, ISYM, MATVEC, MSOLVE,
     $     ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT, R, Z, P, DZ,
     $     RWORK, IWORK )
***BEGIN PROLOGUE  SCG
***PURPOSE  Preconditioned Conjugate Gradient Sparse Ax=b Solver.
            Routine to solve a symmetric positive definite linear
            system  Ax = b  using the Preconditioned Conjugate
            Gradient method.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2B4
***TYPE      SINGLE PRECISION (SCG-S, DCG-D)
***KEYWORDS  ITERATIVE PRECONDITION, SLAP, SPARSE,
             SYMMETRIC LINEAR SYSTEM
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER  N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
     INTEGER  ITER, IERR, IUNIT, IWORK(USER DEFINED)
     REAL     B(N), X(N), A(NELT), TOL, ERR, R(N), Z(N)
     REAL     P(N), DZ(N), RWORK(USER DEFINED)
     EXTERNAL MATVEC, MSOLVE

     CALL SCG(N, B, X, NELT, IA, JA, A, ISYM, MATVEC, MSOLVE,
     $     ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT, R, Z, P, DZ,
     $     RWORK, IWORK )

 *Arguments:
 N      :IN       Integer.
         Order of the Matrix.
 B      :IN       Real B(N).
         Right-hand side vector.
 X      :INOUT    Real X(N).
         On input X is your initial guess for solution vector.
         On output X is the final approximate solution.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :IN       Integer IA(NELT).
 JA     :IN       Integer JA(NELT).
 A      :IN       Real A(NELT).
         These arrays contain the matrix data structure for A.
         It could take any form.  See "Description", below,
         for more details.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 MATVEC :EXT      External.
         Name of a routine which performs the matrix vector multiply
         Y = A*X given A and X.  The name of the MATVEC routine must
```

```
        be declared external in the calling program.  The calling
        sequence to MATVEC is:

            CALL MATVEC( N, X, Y, NELT, IA, JA, A, ISYM )

        Where N is the number of unknowns, Y is the product A*X
        upon return X is an input vector, NELT is the number of
        non-zeros in the SLAP IA, JA, A storage for the matrix A.
        ISYM is a flag which, if non-zero, denotest that A is
        symmetric and only the lower or upper triangle is stored.
MSOLVE :EXT      External.
        Name of a routine which solves a linear system MZ = R for
        Z given R with the preconditioning matrix M (M is supplied via
        RWORK and IWORK arrays).  The name of the MSOLVE routine must
        be declared external in the calling program.  The calling
        sequence to MSOLVE is:

            CALL MSOLVE(N, R, Z, NELT, IA, JA, A, ISYM, RWORK, IWORK)

        Where N is the number of unknowns, R is the right-hand side
        vector and Z is the solution upon return.  NELT, IA, JA, A and
        ISYM are defined as above.  RWORK is a real array that can
        be used to pass necessary preconditioning information and/or
        workspace to MSOLVE.  IWORK is an integer work array for
        the same purpose as RWORK.
ITOL   :IN       Integer.
        Flag to indicate type of convergence criterion.
        If ITOL=1, iteration stops when the 2-norm of the residual
        divided by the 2-norm of the right-hand side is less than TOL.
        If ITOL=2, iteration stops when the 2-norm of M-inv times the
        residual divided by the 2-norm of M-inv times the right hand
        side is less than TOL, where M-inv is the inverse of the
        diagonal of A.
        ITOL=11 is often useful for checking and comparing different
        routines.  For this case, the user must supply the "exact"
        solution or a very accurate approximation (one with an error
        much less than TOL) through a common block,
            COMMON /SSLBLK/ SOLN( )
        If ITOL=11, iteration stops when the 2-norm of the difference
        between the iterative approximation and the user-supplied
        solution divided by the 2-norm of the user-supplied solution
        is less than TOL.  Note that this requires the user to set up
        the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine.
        The routine with this declaration should be loaded before the
        stop test so that the correct length is used by the loader.
        This procedure is not standard Fortran and may not work
        correctly on your system (although it has worked on every
        system the authors have tried).  If ITOL is not 11 then this
        common block is indeed standard Fortran.
TOL    :INOUT    Real.
        Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX  :IN       Integer.
        Maximum number of iterations.
ITER   :OUT      Integer.
        Number of iterations required to reach convergence, or
        ITMAX+1 if convergence criterion could not be achieved in
        ITMAX iterations.
ERR    :OUT      Real.
        Error estimate of error in final approximate solution, as
        defined by ITOL.
```

```
IERR   :OUT      Integer.
        Return error flag.
          IERR = 0 => All went well.
          IERR = 1 => Insufficient space allocated for WORK or IWORK.
          IERR = 2 => Method failed to converge in ITMAX steps.
          IERR = 3 => Error in user input.
                      Check input values of N, ITOL.
          IERR = 4 => User error tolerance set too tight.
                      Reset to 500*R1MACH(3).  Iteration proceeded.
          IERR = 5 => Preconditioning matrix, M, is not positive
                      definite.  (r,z) < 0.
          IERR = 6 => Matrix A is not positive definite.  (p,Ap) < 0.
IUNIT  :IN       Integer.
        Unit number on which to write the error at each iteration,
        if this is desired for monitoring convergence.  If unit
        number is 0, no writing will occur.
R      :WORK     Real R(N).
Z      :WORK     Real Z(N).
P      :WORK     Real P(N).
DZ     :WORK     Real DZ(N).
        Real arrays used for workspace.
RWORK  :WORK     Real RWORK(USER DEFINED).
        Real array that can be used by  MSOLVE.
IWORK  :WORK     Integer IWORK(USER DEFINED).
        Integer array that can be used by  MSOLVE.
```

*Description

This routine does not care what matrix data structure is used for A and M. It simply calls the MATVEC and MSOLVE routines, with the arguments as described above. The user could write any type of structure and the appropriate MATVEC and MSOLVE routines. It is assumed that A is stored in the IA, JA, A arrays in some fashion and that M (or INV(M)) is stored in IWORK and RWORK in some fashion. The SLAP routines SSDCG and SSICCG are examples of this procedure.

Two examples of matrix data structures are the: 1) SLAP Triad format and 2) SLAP Column format.

========== S L A P Triad format ==========

In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
```

```
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Cautions:

This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO SSDCG, SSICCG

***REFERENCES

1. Louis Hageman and David Young, Applied Iterative Methods, Academic Press, New York, 1981.
2. Concus, Golub and O'Leary, A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations, in Sparse Matrix Computations, Bunch and Rose, Eds., Academic Press, New York, 1979.
3. Mark K. Seager, A SLAP for the Masses, in G. F. Carey, Ed., Parallel Supercomputing: Methods, Algorithms and Applications, Wiley, 1989, pp.135-155.

***ROUTINES CALLED ISSCG, R1MACH, SAXPY, SCOPY, SDOT

***REVISION HISTORY (YYMMDD)

```
871119  DATE WRITTEN
881213  Previous REVISION DATE
890915  Made changes requested at July 1989 CML Meeting.  (MKS)
890921  Removed TeX from comments.  (FNF)
890922  Numerous changes to prologue to make closer to SLATEC
```

```
        standard.  (FNF)
890929  Numerous changes to reduce SP/DP differences.  (FNF)
891004  Added new reference.
910411  Prologue converted to Version 4.0 format.  (BAB)
910502  Removed MATVEC and MSOLVE from ROUTINES CALLED list.  (FNF)
920407  COMMON BLOCK renamed SSLBLK.  (WRB)
920511  Added complete declaration section.  (WRB)
920929  Corrected format of references.  (FNF)
921019  Changed 500.0 to 500 to reduce SP/DP differences.  (FNF)
END PROLOGUE
```

SCGN

```
      SUBROUTINE SCGN(N, B, X, NELT, IA, JA, A, ISYM, MATVEC, MTTVEC,
     $     MSOLVE, ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT, R,
     $     Z, P, ATP, ATZ, DZ, ATDZ, RWORK, IWORK)
***BEGIN PROLOGUE  SCGN
***PURPOSE  Preconditioned CG Sparse Ax=b Solver for Normal Equations.
            Routine to solve a general linear system  Ax = b  using the
            Preconditioned Conjugate Gradient method applied to the
            normal equations  AA'y = b, x=A'y.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SCGN-S, DCGN-D)
***KEYWORDS  ITERATIVE PRECONDITION, NON-SYMMETRIC LINEAR SYSTEM SOLVE,
             NORMAL EQUATIONS., SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER  N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
     INTEGER  ITER, IERR, IUNIT, IWORK(USER DEFINED)
     REAL     B(N), X(N), A(NELT), TOL, ERR, R(N), Z(N)
     REAL     P(N), ATP(N), ATZ(N), DZ(N), ATDZ(N)
     REAL     RWORK(USER DEFINED)
     EXTERNAL MATVEC, MTTVEC, MSOLVE

     CALL SCGN(N, B, X, NELT, IA, JA, A, ISYM, MATVEC, MTTVEC,
     $     MSOLVE, ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT, R,
     $     Z, P, ATP, ATZ, DZ, ATDZ, RWORK, IWORK)

 *Arguments:
 N      :IN       Integer
         Order of the Matrix.
 B      :IN       Real B(N).
         Right-hand side vector.
 X      :INOUT    Real X(N).
         On input X is your initial guess for solution vector.
         On output X is the final approximate solution.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :IN       Integer IA(NELT).
 JA     :IN       Integer JA(NELT).
 A      :IN       Real A(NELT).
         These arrays contain the matrix data structure for A.
         It could take any form.  See "Description", below,
         for more details.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 MATVEC :EXT      External.
         Name of a routine which performs the matrix vector multiply
```

y = A*X given A and X. The name of the MATVEC routine must be declared external in the calling program. The calling sequence to MATVEC is:

```
    CALL MATVEC( N, X, Y, NELT, IA, JA, A, ISYM )
```

Where N is the number of unknowns, Y is the product A*X upon return X is an input vector, NELT is the number of non-zeros in the SLAP-Column IA, JA, A storage for the matrix A. ISYM is a flag which, if non-zero, denotes that A is symmetric and only the lower or upper triangle is stored.

MTTVEC :EXT External.

Name of a routine which performs the matrix transpose vector multiply y = A'*X given A and X (where ' denotes transpose). The name of the MTTVEC routine must be declared external in the calling program. The calling sequence to MTTVEC is the same as that for MATVEC, viz.:

```
    CALL MTTVEC( N, X, Y, NELT, IA, JA, A, ISYM )
```

Where N is the number of unknowns, Y is the product A'*X upon return X is an input vector, NELT is the number of non-zeros in the SLAP-Column IA, JA, A storage for the matrix A. ISYM is a flag which, if non-zero, denotes that A is symmetric and only the lower or upper triangle is stored.

MSOLVE :EXT External.

Name of a routine which solves a linear system MZ = R for Z given R with the preconditioning matrix M (M is supplied via RWORK and IWORK arrays). The name of the MSOLVE routine must be declared external in the calling program. The calling sequence to MSOLVE is:

```
    CALL MSOLVE(N, R, Z, NELT, IA, JA, A, ISYM, RWORK, IWORK)
```

Where N is the number of unknowns, R is the right-hand side vector and Z is the solution upon return. NELT, IA, JA, A and ISYM are defined as above. RWORK is a real array that can be used to pass necessary preconditioning information and/or workspace to MSOLVE. IWORK is an integer work array for the same purpose as RWORK.

ITOL :IN Integer.

Flag to indicate type of convergence criterion.
If ITOL=1, iteration stops when the 2-norm of the residual divided by the 2-norm of the right-hand side is less than TOL.
If ITOL=2, iteration stops when the 2-norm of M-inv times the residual divided by the 2-norm of M-inv times the right hand side is less than TOL, where M-inv is the inverse of the diagonal of A.
ITOL=11 is often useful for checking and comparing different routines. For this case, the user must supply the "exact" solution or a very accurate approximation (one with an error much less than TOL) through a common block,

```
    COMMON /SSLBLK/ SOLN( )
```

If ITOL=11, iteration stops when the 2-norm of the difference between the iterative approximation and the user-supplied solution divided by the 2-norm of the user-supplied solution is less than TOL. Note that this requires the user to set up the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine. The routine with this declaration should be loaded before the stop test so that the correct length is used by the loader. This procedure is not standard Fortran and may not work correctly on your system (although it has worked on every system the authors have tried). If ITOL is not 11 then this common block is indeed standard Fortran.

TOL :INOUT Real.

Convergence criterion, as described above. (Reset if IERR=4.)

```
ITMAX  :IN       Integer.
        Maximum number of iterations.
ITER   :OUT      Integer.
        Number of iterations required to reach convergence, or
        ITMAX+1 if convergence criterion could not be achieved in
        ITMAX iterations.
ERR    :OUT      Real.
        Error estimate of error in final approximate solution, as
        defined by ITOL.
IERR   :OUT      Integer.
        Return error flag.
          IERR = 0 => All went well.
          IERR = 1 => Insufficient space allocated for WORK or IWORK.
          IERR = 2 => Method failed to converge in ITMAX steps.
          IERR = 3 => Error in user input.
                      Check input values of N, ITOL.
          IERR = 4 => User error tolerance set too tight.
                      Reset to 500*R1MACH(3).  Iteration proceeded.
          IERR = 5 => Preconditioning matrix, M, is not positive
                      definite.  (r,z) < 0.
          IERR = 6 => Matrix A is not positive definite.  (p,Ap) < 0.
IUNIT  :IN       Integer.
        Unit number on which to write the error at each iteration,
        if this is desired for monitoring convergence.  If unit
        number is 0, no writing will occur.
R      :WORK     Real R(N).
Z      :WORK     Real Z(N).
P      :WORK     Real P(N).
ATP    :WORK     Real ATP(N).
ATZ    :WORK     Real ATZ(N).
DZ     :WORK     Real DZ(N).
ATDZ   :WORK     Real ATDZ(N).
        Real arrays used for workspace.
RWORK  :WORK     Real RWORK(USER DEFINED).
        Real array that can be used by  MSOLVE.
IWORK  :WORK     Integer IWORK(USER DEFINED).
        Integer array that can be used by  MSOLVE.
```

*Description:

This routine applies the preconditioned conjugate gradient (PCG) method to a non-symmetric system of equations Ax=b. To do this the normal equations are solved:

$$AA'\,y = b, \text{ where } x = A'y.$$

In PCG method the iteration count is determined by condition number of the matrix $(M^{-1}A)$. In the situation where the normal equations are used to solve a non-symmetric system the condition number depends on AA' and should therefore be much worse than that of A. This is the conventional wisdom. When one has a good preconditioner for AA' this may not hold. The latter is the situation when SCGN should be tried.

If one is trying to solve a symmetric system, SCG should be used instead.

This routine does not care what matrix data structure is used for A and M. It simply calls MATVEC, MTTVEC and MSOLVE routines, with arguments as described above. The user could write any type of structure, and appropriate MATVEC, MTTVEC and MSOLVE routines. It is assumed that A is stored in the

IA, JA, A arrays in some fashion and that M (or INV(M)) is stored in IWORK and RWORK) in some fashion. The SLAP routines SSDCGN and SSLUCN are examples of this procedure.

Two examples of matrix data structures are the: 1) SLAP Triad format and 2) SLAP Column format.

========== S L A P Triad format ==========

In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
```

```
       | 0  0  0 44  0|
       |51  0 53  0 55|

 *Cautions:
     This routine will attempt to write to the Fortran logical output
     unit IUNIT, if IUNIT .ne. 0.  Thus, the user must make sure that
     this logical unit is attached to a file or terminal before calling
     this routine with a non-zero value for IUNIT.  This routine does
     not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO  SSDCGN, SSLUCN, ISSCGN
***REFERENCES  1. Mark K. Seager, A SLAP for the Masses, in
                  G. F. Carey, Ed., Parallel Supercomputing: Methods,
                  Algorithms and Applications, Wiley, 1989, pp.135-155.
***ROUTINES CALLED  ISSCGN, R1MACH, SAXPY, SCOPY, SDOT
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890921  Removed TeX from comments.  (FNF)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   891004  Added new reference.
   910411  Prologue converted to Version 4.0 format.  (BAB)
   910502  Removed MATVEC, MTTVEC and MSOLVE from ROUTINES CALLED
           list.  (FNF)
   920407  COMMON BLOCK renamed SSLBLK.  (WRB)
   920511  Added complete declaration section.  (WRB)
   920929  Corrected format of reference.  (FNF)
   921019  Changed 500.0 to 500 to reduce SP/DP differences.  (FNF)
   921113  Corrected C***CATEGORY line.  (FNF)
   END PROLOGUE
```

SCGS

```
      SUBROUTINE SCGS(N, B, X, NELT, IA, JA, A, ISYM, MATVEC,
     $     MSOLVE, ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT,
     $     R, R0, P, Q, U, V1, V2, RWORK, IWORK)
***BEGIN PROLOGUE  SCGS
***PURPOSE  Preconditioned BiConjugate Gradient Squared Ax=b Solver.
            Routine to solve a Non-Symmetric linear system  Ax = b
            using the Preconditioned BiConjugate Gradient Squared
            method.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SCGS-S, DCGS-D)
***KEYWORDS  BICONJUGATE GRADIENT, ITERATIVE PRECONDITION,
             NON-SYMMETRIC LINEAR SYSTEM, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
      INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
      INTEGER ITER, IERR, IUNIT, IWORK(USER DEFINED)
      REAL    B(N), X(N), A(NELT), TOL, ERR, R(N), R0(N), P(N)
      REAL    Q(N), U(N), V1(N), V2(N), RWORK(USER DEFINED)
      EXTERNAL MATVEC, MSOLVE

      CALL SCGS(N, B, X, NELT, IA, JA, A, ISYM, MATVEC,
     $     MSOLVE, ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT,
     $     R, R0, P, Q, U, V1, V2, RWORK, IWORK)

 *Arguments:
 N      :IN       Integer
         Order of the Matrix.
 B      :IN       Real B(N).
         Right-hand side vector.
 X      :INOUT    Real X(N).
         On input X is your initial guess for solution vector.
         On output X is the final approximate solution.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :IN       Integer IA(NELT).
 JA     :IN       Integer JA(NELT).
 A      :IN       Real A(NELT).
         These arrays contain the matrix data structure for A.
         It could take any form.  See "Description", below,
         for more details.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 MATVEC :EXT      External.
         Name of a routine which  performs the matrix vector multiply
         operation  Y = A*X  given A and X.  The  name of  the MATVEC
```

```
        routine must  be declared external  in the  calling program.
        The calling sequence of MATVEC is:
            CALL MATVEC( N, X, Y, NELT, IA, JA, A, ISYM )
        Where N is the number of unknowns, Y is the product A*X upon
        return,  X is an input  vector.  NELT, IA,  JA,  A and  ISYM
        define the SLAP matrix data structure: see Description,below.
MSOLVE :EXT      External.
        Name of a routine which solves a linear system MZ = R  for Z
        given R with the preconditioning matrix M (M is supplied via
        RWORK  and IWORK arrays).   The name  of  the MSOLVE routine
        must be declared  external  in the  calling   program.   The
        calling sequence of MSOLVE is:
            CALL MSOLVE(N, R, Z, NELT, IA, JA, A, ISYM, RWORK, IWORK)
        Where N is the number of unknowns, R is  the right-hand side
        vector, and Z is the solution upon return.  NELT,  IA, JA, A
        and  ISYM define the SLAP  matrix  data structure: see
        Description, below.  RWORK is a  real array that can be used
        to  pass   necessary  preconditioning     information and/or
        workspace to MSOLVE.  IWORK is an integer work array for the
        same purpose as RWORK.
ITOL   :IN       Integer.
        Flag to indicate type of convergence criterion.
        If ITOL=1, iteration stops when the 2-norm of the residual
        divided by the 2-norm of the right-hand side is less than TOL.
        This routine must calculate the residual from R = A*X - B.
        This is unnatural and hence expensive for this type of iter-
        ative method.  ITOL=2 is *STRONGLY* recommended.
        If ITOL=2, iteration stops when the 2-norm of M-inv times the
        residual divided by the 2-norm of M-inv times the right hand
        side is less than TOL, where M-inv time a vector is the pre-
        conditioning step.  This is the *NATURAL* stopping for this
        iterative method and is *STRONGLY* recommended.
        ITOL=11 is often useful for checking and comparing different
        routines.  For this case, the user must supply the "exact"
        solution or a very accurate approximation (one with an error
        much less than TOL) through a common block,
            COMMON /SSLBLK/ SOLN( )
        If ITOL=11, iteration stops when the 2-norm of the difference
        between the iterative approximation and the user-supplied
        solution divided by the 2-norm of the user-supplied solution
        is less than TOL.
TOL    :INOUT    Real.
        Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX  :IN       Integer.
        Maximum number of iterations.
ITER   :OUT      Integer.
        Number of iterations required to reach convergence, or
        ITMAX+1 if convergence criterion could not be achieved in
        ITMAX iterations.
ERR    :OUT      Real.
        Error estimate of error in final approximate solution, as
        defined by ITOL.
IERR   :OUT      Integer.
        Return error flag.
          IERR = 0 => All went well.
          IERR = 1 => Insufficient space allocated for WORK or IWORK.
          IERR = 2 => Method failed to converge in ITMAX steps.
          IERR = 3 => Error in user input.
                      Check input values of N, ITOL.
          IERR = 4 => User error tolerance set too tight.
```

```
                    Reset to 500*R1MACH(3).  Iteration proceeded.
          IERR = 5 => Breakdown of the method detected.
                      (r0,r) approximately 0.
          IERR = 6 => Stagnation of the method detected.
                      (r0,v) approximately 0.
IUNIT  :IN       Integer.
        Unit number on which to write the error at each iteration,
        if this is desired for monitoring convergence.  If unit
        number is 0, no writing will occur.
R      :WORK     Real R(N).
R0     :WORK     Real R0(N).
P      :WORK     Real P(N).
Q      :WORK     Real Q(N).
U      :WORK     Real U(N).
V1     :WORK     Real V1(N).
V2     :WORK     Real V2(N).
        Real arrays used for workspace.
RWORK  :WORK     Real RWORK(USER DEFINED).
        Real array that can be used for workspace in MSOLVE.
IWORK  :WORK     Integer IWORK(USER DEFINED).
        Integer array that can be used for workspace in MSOLVE.
```

*Description

This routine does not care what matrix data structure is used for A and M. It simply calls the MATVEC and MSOLVE routines, with the arguments as described above. The user could write any type of structure and the appropriate MATVEC and MSOLVE routines. It is assumed that A is stored in the IA, JA, A arrays in some fashion and that M (or INV(M)) is stored in IWORK and RWORK in some fashion. The SLAP routines SSDBCG and SSLUCS are examples of this procedure.

Two examples of matrix data structures are the: 1) SLAP Triad format and 2) SLAP Column format.

========== S L A P Triad format ==========

In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
```

```
|51  0 53  0 55|

=================== S L A P Column format ==================

In  this format   the non-zeros are    stored counting  down
columns (except  for the diagonal  entry, which must  appear
first in each "column") and are  stored in the real array A.
In other words,  for  each column    in the matrix   put the
diagonal  entry  in A.   Then   put  in the  other  non-zero
elements going   down the  column (except  the  diagonal) in
order.  The IA array holds the row index  for each non-zero.
The JA array holds the offsets into the IA, A arrays for the
beginning   of   each  column.      That is,   IA(JA(ICOL)),
A(JA(ICOL)) points to the beginning of the ICOL-th column in
IA and  A.  IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1)  points to the
end of the ICOL-th column.  Note that we always have JA(N+1)
= NELT+1, where N is the number of columns in the matrix and
NELT is the number of non-zeros in the matrix.

Here is an example of the  SLAP Column  storage format for a
5x5 Matrix (in the A and IA arrays '|'  denotes the end of a
column):

    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Cautions:

This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO SSDCGS, SSLUCS

***REFERENCES 1. P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric linear systems, Delft University of Technology Report 84-16, Department of Mathematics and Informatics, Delft, The Netherlands.

2. E. F. Kaasschieter, The solution of non-symmetric linear systems by biconjugate gradients or conjugate gradients squared, Delft University of Technology Report 86-21, Department of Mathematics and Informatics, Delft, The Netherlands.

3. Mark K. Seager, A SLAP for the Masses, in G. F. Carey, Ed., Parallel Supercomputing: Methods, Algorithms and Applications, Wiley, 1989, pp.135-155.

***ROUTINES CALLED ISSCGS, R1MACH, SAXPY, SDOT

***REVISION HISTORY (YYMMDD)

871119	DATE WRITTEN
881213	Previous REVISION DATE
890915	Made changes requested at July 1989 CML Meeting. (MKS)
890921	Removed TeX from comments. (FNF)
890922	Numerous changes to prologue to make closer to SLATEC standard. (FNF)
890929	Numerous changes to reduce SP/DP differences. (FNF)

```
891004  Added new reference.
910411  Prologue converted to Version 4.0 format.  (BAB)
910502  Removed MATVEC and MSOLVE from ROUTINES CALLED list.  (FNF)
920407  COMMON BLOCK renamed SSLBLK.  (WRB)
920511  Added complete declaration section.  (WRB)
920929  Corrected format of references.  (FNF)
921019  Changed 500.0 to 500 to reduce SP/DP differences.  (FNF)
921113  Corrected C***CATEGORY line.  (FNF)
END PROLOGUE
```

SCHDC

```
      SUBROUTINE SCHDC (A, LDA, P, WORK, JPVT, JOB, INFO)
***BEGIN PROLOGUE  SCHDC
***PURPOSE  Compute the Cholesky decomposition of a positive definite
            matrix.  A pivoting option allows the user to estimate the
            condition number of a positive definite matrix or determine
            the rank of a positive semidefinite matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B
***TYPE      SINGLE PRECISION (SCHDC-S, DCHDC-D, CCHDC-C)
***KEYWORDS  CHOLESKY DECOMPOSITION, LINEAR ALGEBRA, LINPACK, MATRIX,
             POSITIVE DEFINITE
***AUTHOR  Dongarra, J., (ANL)
           Stewart, G. W., (U. of Maryland)
***DESCRIPTION

     SCHDC computes the Cholesky decomposition of a positive definite
     matrix.  A pivoting option allows the user to estimate the
     condition of a positive definite matrix or determine the rank
     of a positive semidefinite matrix.

     On Entry

         A      REAL(LDA,P).
                A contains the matrix whose decomposition is to
                be computed.  Only the upper half of A need be stored.
                The lower part of the array A is not referenced.

         LDA    INTEGER.
                LDA is the leading dimension of the array A.

         P      INTEGER.
                P is the order of the matrix.

         WORK   REAL.
                WORK is a work array.

         JPVT   INTEGER(P).
                JPVT contains integers that control the selection
                of the pivot elements, if pivoting has been requested.
                Each diagonal element A(K,K)
                is placed in one of three classes according to the
                value of JPVT(K).

                   If JPVT(K) .GT. 0, then X(K) is an initial
                                      element.

                   If JPVT(K) .EQ. 0, then X(K) is a free element.

                   If JPVT(K) .LT. 0, then X(K) is a final element.

                Before the decomposition is computed, initial elements
                are moved by symmetric row and column interchanges to
                the beginning of the array A and final
                elements to the end.  Both initial and final elements
                are frozen in place during the computation and only
                free elements are moved.  At the K-th stage of the
```

```
                reduction, if A(K,K) is occupied by a free element
                it is interchanged with the largest free element
                A(L,L) with L .GE. K.  JPVT is not referenced if
                JOB .EQ. 0.

         JOB    INTEGER.
                JOB is an integer that initiates column pivoting.
                If JOB .EQ. 0, no pivoting is done.
                If JOB .NE. 0, pivoting is done.

      On Return

          A     A contains in its upper half the Cholesky factor
                of the matrix A as it has been permuted by pivoting.

          JPVT  JPVT(J) contains the index of the diagonal element
                of a that was moved into the J-th position,
                provided pivoting was requested.

          INFO  contains the index of the last positive diagonal
                element of the Cholesky factor.

      For positive definite matrices INFO = P is the normal return.
      For pivoting with positive semidefinite matrices INFO will
      in general be less than P.  However, INFO may be greater than
      the rank of A, since rounding error can cause an otherwise zero
      element to be positive.  Indefinite systems will always cause
      INFO to be less than P.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SSWAP
***REVISION HISTORY  (YYMMDD)
   790319  DATE WRITTEN
   890313  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SCHDD

```
      SUBROUTINE SCHDD (R, LDR, P, X, Z, LDZ, NZ, Y, RHO, C, S, INFO)
***BEGIN PROLOGUE  SCHDD
***PURPOSE  Downdate an augmented Cholesky decomposition or the
            triangular factor of an augmented QR decomposition.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D7B
***TYPE      SINGLE PRECISION (SCHDD-S, DCHDD-D, CCHDD-C)
***KEYWORDS  CHOLESKY DECOMPOSITION, DOWNDATE, LINEAR ALGEBRA, LINPACK,
             MATRIX
***AUTHOR  Stewart, G. W., (U. of Maryland)
***DESCRIPTION

      SCHDD downdates an augmented Cholesky decomposition or the
      triangular factor of an augmented QR decomposition.
      Specifically, given an upper triangular matrix R of order P, a
      row vector X, a column vector Z, and a scalar Y, SCHDD
      determines an orthogonal matrix U and a scalar ZETA such that

                          (R   Z )     (RR  ZZ)
                     U * (       )  =  (       ) ,
                          (0 ZETA)     ( X   Y)

      where RR is upper triangular.  If R and Z have been obtained
      from the factorization of a least squares problem, then
      RR and ZZ are the factors corresponding to the problem
      with the observation (X,Y) removed.  In this case, if RHO
      is the norm of the residual vector, then the norm of
      the residual vector of the downdated problem is
      SQRT(RHO**2 - ZETA**2). SCHDD will simultaneously downdate
      several triplets (Z,Y,RHO) along with R.
      For a less terse description of what SCHDD does and how
      it may be applied, see the LINPACK guide.

      The matrix U is determined as the product U(1)*...*U(P)
      where U(I) is a rotation in the (P+1,I)-plane of the
      form

                        ( C(I)     -S(I)     )
                        (                    ) .
                        ( S(I)       C(I)    )

      The rotations are chosen so that C(I) is real.

      The user is warned that a given downdating problem may
      be impossible to accomplish or may produce
      inaccurate results.  For example, this can happen
      if X is near a vector whose removal will reduce the
      rank of R.  Beware.

      On Entry

          R      REAL(LDR,P), where LDR .GE. P.
                 R contains the upper triangular matrix
                 that is to be downdated.  The part of  R
                 below the diagonal is not referenced.
```

```
         LDR    INTEGER.
                LDR is the leading dimension of the array R.

         P      INTEGER.
                P is the order of the matrix R.

         X      REAL(P).
                X contains the row vector that is to
                be removed from R.  X is not altered by SCHDD.

         Z      REAL(LDZ,NZ), where LDZ .GE. P.
                Z is an array of NZ P-vectors which
                are to be downdated along with R.

         LDZ    INTEGER.
                LDZ is the leading dimension of the array Z.

         NZ     INTEGER.
                NZ is the number of vectors to be downdated
                NZ may be zero, in which case Z, Y, and RHO
                are not referenced.

         Y      REAL(NZ).
                Y contains the scalars for the downdating
                of the vectors Z.  Y is not altered by SCHDD.

         RHO    REAL(NZ).
                RHO contains the norms of the residual
                vectors that are to be downdated.

     On Return

         R
         Z      contain the downdated quantities.
         RHO

         C      REAL(P).
                C contains the cosines of the transforming
                rotations.

         S      REAL(P).
                S contains the sines of the transforming
                rotations.

         INFO   INTEGER.
                INFO is set as follows.

                   INFO = 0  if the entire downdating
                             was successful.

                   INFO =-1  if R could not be downdated.
                             In this case, all quantities
                             are left unaltered.

                   INFO = 1  if some RHO could not be
                             downdated.  The offending RHOs are
                             set to -1.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
```

```
***ROUTINES CALLED  SDOT, SNRM2
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SCHEX

```
      SUBROUTINE SCHEX (R, LDR, P, K, L, Z, LDZ, NZ, C, S, JOB)
***BEGIN PROLOGUE  SCHEX
***PURPOSE  Update the Cholesky factorization  A=TRANS(R)*R  of A
            positive definite matrix A of order P under diagonal
            permutations of the form TRANS(E)*A*E, where E is a
            permutation matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D7B
***TYPE      SINGLE PRECISION (SCHEX-S, DCHEX-D, CCHEX-C)
***KEYWORDS  CHOLESKY DECOMPOSITION, EXCHANGE, LINEAR ALGEBRA, LINPACK,
             MATRIX, POSITIVE DEFINITE
***AUTHOR  Stewart, G. W., (U. of Maryland)
***DESCRIPTION

     SCHEX updates the Cholesky factorization

                  A = TRANS(R)*R

     of a positive definite matrix A of order P under diagonal
     permutations of the form

                  TRANS(E)*A*E

     where E is a permutation matrix.  Specifically, given
     an upper triangular matrix R and a permutation matrix
     E (which is specified by K, L, and JOB), SCHEX determines
     an orthogonal matrix U such that

                          U*R*E = RR,

     where RR is upper triangular.  At the users option, the
     transformation U will be multiplied into the array Z.
     If A = TRANS(X)*X, so that R is the triangular part of the
     QR factorization of X, then RR is the triangular part of the
     QR factorization of X*E, i.e., X with its columns permuted.
     For a less terse description of what SCHEX does and how
     it may be applied, see the LINPACK guide.

     The matrix Q is determined as the product U(L-K)*...*U(1)
     of plane rotations of the form

                          (    C(I)       S(I) )
                          (                    ) ,
                          (    -S(I)      C(I) )

     where C(I) is real.  The rows these rotations operate on
     are described below.

     There are two types of permutations, which are determined
     by the value of JOB.

     1. Right circular shift (JOB = 1).

         The columns are rearranged in the following order.

                 1,...,K-1,L,K,K+1,...,L-1,L+1,...,P.
```

U is the product of L-K rotations U(I), where U(I) acts in the (L-I,L-I+1)-plane.

2. Left circular shift (JOB = 2).
   The columns are rearranged in the following order

```
1,...,K-1,K+1,K+2,...,L,K,L+1,...,P.
```

   U is the product of L-K rotations U(I), where U(I) acts in the (K+I-1,K+I)-plane.

On Entry

```
R      REAL(LDR,P), where LDR .GE. P.
       R contains the upper triangular factor
       that is to be updated.  Elements of R
       below the diagonal are not referenced.

LDR    INTEGER.
       LDR is the leading dimension of the array R.

P      INTEGER.
       P is the order of the matrix R.

K      INTEGER.
       K is the first column to be permuted.

L      INTEGER.
       L is the last column to be permuted.
       L must be strictly greater than K.

Z      REAL(LDZ,NZ), where LDZ.GE.P.
       Z is an array of NZ P-vectors into which the
       transformation U is multiplied.  Z is
       not referenced if NZ = 0.

LDZ    INTEGER.
       LDZ is the leading dimension of the array Z.

NZ     INTEGER.
       NZ is the number of columns of the matrix Z.

JOB    INTEGER.
       JOB determines the type of permutation.
              JOB = 1  right circular shift.
              JOB = 2  left circular shift.
```

On Return

```
R      contains the updated factor.

Z      contains the updated matrix Z.

C      REAL(P).
       C contains the cosines of the transforming rotations.

S      REAL(P).
       S contains the sines of the transforming rotations.
```

```
***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SROTG
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890531  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SCHUD

```
      SUBROUTINE SCHUD (R, LDR, P, X, Z, LDZ, NZ, Y, RHO, C, S)
***BEGIN PROLOGUE  SCHUD
***PURPOSE  Update an augmented Cholesky decomposition of the
            triangular part of an augmented QR decomposition.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D7B
***TYPE      SINGLE PRECISION (SCHUD-S, DCHUD-D, CCHUD-C)
***KEYWORDS  CHOLESKY DECOMPOSITION, LINEAR ALGEBRA, LINPACK, MATRIX,
             UPDATE
***AUTHOR  Stewart, G. W., (U. of Maryland)
***DESCRIPTION

      SCHUD updates an augmented Cholesky decomposition of the
      triangular part of an augmented QR decomposition.  Specifically,
      given an upper triangular matrix R of order P, a row vector
      X, a column vector Z, and a scalar Y, SCHUD determines a
      unitary matrix U and a scalar ZETA such that


                           (R  Z)     (RR   ZZ )
                      U  * (    )  =  (         ) ,
                           (X  Y)     ( 0  ZETA)

      where RR is upper triangular.  If R and Z have been
      obtained from the factorization of a least squares
      problem, then RR and ZZ are the factors corresponding to
      the problem with the observation (X,Y) appended.  In this
      case, if RHO is the norm of the residual vector, then the
      norm of the residual vector of the updated problem is
      SQRT(RHO**2 + ZETA**2).  SCHUD will simultaneously update
      several triplets (Z,Y,RHO).
      For a less terse description of what SCHUD does and how
      it may be applied, see the LINPACK guide.

      The matrix U is determined as the product U(P)*...*U(1),
      where U(I) is a rotation in the (I,P+1) plane of the
      form

                        (     C(I)      S(I) )
                        (                    ) .
                        (    -S(I)      C(I) )

      The rotations are chosen so that C(I) is real.

      On Entry

          R      REAL(LDR,P), where LDR .GE. P.
                 R contains the upper triangular matrix
                 that is to be updated.  The part of R
                 below the diagonal is not referenced.

          LDR    INTEGER.
                 LDR is the leading dimension of the array R.

          P      INTEGER.
                 P is the order of the matrix R.
```

```
          X      REAL(P).
                 X contains the row to be added to R.  X is
                 not altered by SCHUD.

          Z      REAL(LDZ,NZ), where LDZ .GE. P.
                 Z is an array containing NZ P-vectors to
                 be updated with R.

          LDZ    INTEGER.
                 LDZ is the leading dimension of the array Z.

          NZ     INTEGER.
                 NZ is the number of vectors to be updated.
                 NZ may be zero, in which case Z, Y, and RHO
                 are not referenced.

          Y      REAL(NZ).
                 Y contains the scalars for updating the vectors
                 Z.  Y is not altered by SCHUD.

          RHO    REAL(NZ).
                 RHO contains the norms of the residual
                 vectors that are to be updated.  If RHO(J)
                 is negative, it is left unaltered.

      On Return

          RC
          RHO    contain the updated quantities.
          Z

          C      REAL(P).
                 C contains the cosines of the transforming
                 rotations.

          S      REAL(P).
                 S contains the sines of the transforming
                 rotations.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SROTG
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SCNRM2

```
      REAL FUNCTION SCNRM2 (N, CX, INCX)
***BEGIN PROLOGUE  SCNRM2
***PURPOSE  Compute the unitary norm of a complex vector.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A3B
***TYPE      COMPLEX (SNRM2-S, DNRM2-D, SCNRM2-C)
***KEYWORDS  BLAS, EUCLIDEAN LENGTH, EUCLIDEAN NORM, L2,
             LINEAR ALGEBRA, UNITARY, VECTOR
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
           Kincaid, D. R., (U. of Texas)
           Krogh, F. T., (JPL)
***DESCRIPTION

                B L A S  Subprogram
    Description of Parameters

      --Input--
         N  number of elements in input vector(s)
        CX  complex vector with N elements
      INCX  storage spacing between elements of CX

      --Output--
    SCNRM2  single precision result (zero if N .LE. 0)

      Unitary norm of the complex N-vector stored in CX with storage
      increment INCX.
      If N .LE. 0, return with result = 0.
      If N .GE. 1, then INCX must be .GE. 1

      Four phase method using two built-in constants that are
      hopefully applicable to all machines.
          CUTLO = maximum of  SQRT(U/EPS)  over all known machines.
          CUTHI = minimum of  SQRT(V)      over all known machines.
      where
          EPS = smallest no. such that EPS + 1. .GT. 1.
          U   = smallest positive no.   (underflow limit)
          V   = largest  no.            (overflow  limit)

      Brief outline of algorithm.

      Phase 1 scans zero components.
      Move to phase 2 when a component is nonzero and .LE. CUTLO
      Move to phase 3 when a component is .GT. CUTLO
      Move to phase 4 when a component is .GE. CUTHI/M
      where M = N for X() real and M = 2*N for complex.

      Values for CUTLO and CUTHI.
      From the environmental parameters listed in the IMSL converter
      document the limiting values are as follows:
      CUTLO, S.P.   U/EPS = 2**(-102) for  Honeywell.  Close seconds are
                    Univac and DEC at 2**(-103)
                    Thus CUTLO = 2**(-51) = 4.44089E-16
      CUTHI, S.P.   V = 2**127 for Univac, Honeywell, and DEC.
                    Thus CUTHI = 2**(63.5) = 1.30438E19
      CUTLO, D.P.   U/EPS = 2**(-67) for Honeywell and DEC.
```

```
                    Thus CUTLO = 2**(-33.5) = 8.23181D-11
     CUTHI, D.P.   same as S.P.  CUTHI = 1.30438D19
     DATA CUTLO, CUTHI /8.232D-11,  1.304D19/
     DATA CUTLO, CUTHI /4.441E-16,  1.304E19/

***REFERENCES  C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T.
                 Krogh, Basic linear algebra subprograms for Fortran
                 usage, Algorithm No. 539, Transactions on Mathematical
                 Software 5, 3 (September 1979), pp. 308-323.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   791001  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SCOPY

```
      SUBROUTINE SCOPY (N, SX, INCX, SY, INCY)
***BEGIN PROLOGUE  SCOPY
***PURPOSE  Copy a vector.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A5
***TYPE      SINGLE PRECISION (SCOPY-S, DCOPY-D, CCOPY-C, ICOPY-I)
***KEYWORDS  BLAS, COPY, LINEAR ALGEBRA, VECTOR
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
           Kincaid, D. R., (U. of Texas)
           Krogh, F. T., (JPL)
***DESCRIPTION

                B L A S  Subprogram
    Description of Parameters

     --Input--
        N  number of elements in input vector(s)
       SX  single precision vector with N elements
     INCX  storage spacing between elements of SX
       SY  single precision vector with N elements
     INCY  storage spacing between elements of SY

     --Output--
       SY  copy of vector SX (unchanged if N .LE. 0)

     Copy single precision SX to single precision SY.
     For I = 0 to N-1, copy  SX(LX+I*INCX) to SY(LY+I*INCY),
     where LX = 1 if INCX .GE. 0, else LX = 1+(1-N)*INCX, and LY is
     defined in a similar way using INCY.

***REFERENCES  C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T.
                Krogh, Basic linear algebra subprograms for Fortran
                usage, Algorithm No. 539, Transactions on Mathematical
                Software 5, 3 (September 1979), pp. 308-323.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   791001  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920310  Corrected definition of LX in DESCRIPTION.  (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SCOPYM

```
      SUBROUTINE SCOPYM (N, SX, INCX, SY, INCY)
***BEGIN PROLOGUE  SCOPYM
***PURPOSE  Copy the negative of a vector to a vector.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A5
***TYPE      SINGLE PRECISION (SCOPYM-S, DCOPYM-D)
***KEYWORDS  BLAS, COPY, VECTOR
***AUTHOR  Kahaner, D. K., (NBS)
***DESCRIPTION

       Description of Parameters
           The * Flags Output Variables

       N   Number of elements in vector(s)
      SX   Real vector with N elements
    INCX   Storage spacing between elements of SX
      SY*  Real negative copy of SX
    INCY   Storage spacing between elements of SY

      ***  Note that SY = -SX  ***

     Copy negative of real SX to real SY.  For I=0 to N-1,
     copy  -SX(LX+I*INCX) to SY(LY+I*INCY), where LX=1 if
     INCX .GE. 0, else LX = 1+(1-N)*INCX, and LY is defined
     in a similar way using INCY.

***REFERENCES  (NONE)
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   801001  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920310  Corrected definition of LX in DESCRIPTION.  (WRB)
   END PROLOGUE
```

SCOV

```
      SUBROUTINE SCOV (FCN, IOPT, M, N, X, FVEC, R, LDR, INFO, WA1, WA2,
     +   WA3, WA4)
***BEGIN PROLOGUE  SCOV
***PURPOSE  Calculate the covariance matrix for a nonlinear data
            fitting problem.  It is intended to be used after a
            successful return from either SNLS1 or SNLS1E.
***LIBRARY   SLATEC
***CATEGORY  K1B1
***TYPE      SINGLE PRECISION (SCOV-S, DCOV-D)
***KEYWORDS  COVARIANCE MATRIX, NONLINEAR DATA FITTING,
             NONLINEAR LEAST SQUARES
***AUTHOR  Hiebert, K. L., (SNLA)
***DESCRIPTION

  1. Purpose.

     SCOV calculates the covariance matrix for a nonlinear data
     fitting problem.  It is intended to be used after a
     successful return from either SNLS1 or SNLS1E. SCOV
     and SNLS1 (and SNLS1E) have compatible parameters.  The
     required external subroutine, FCN, is the same
     for all three codes, SCOV, SNLS1, and SNLS1E.

  2. Subroutine and Type Statements.

     SUBROUTINE SCOV(FCN,IOPT,M,N,X,FVEC,R,LDR,INFO,
                     WA1,WA2,WA3,WA4)
     INTEGER IOPT,M,N,LDR,INFO
     REAL X(N),FVEC(M),R(LDR,N),WA1(N),WA2(N),WA3(N),WA4(M)
     EXTERNAL FCN

  3. Parameters.

       FCN is the name of the user-supplied subroutine which calculates
         the functions.  If the user wants to supply the Jacobian
         (IOPT=2 or 3), then FCN must be written to calculate the
         Jacobian, as well as the functions.  See the explanation
         of the IOPT argument below.  FCN must be declared in an
         EXTERNAL statement in the calling program and should be
         written as follows.

         SUBROUTINE FCN(IFLAG,M,N,X,FVEC,FJAC,LDFJAC)
         INTEGER IFLAG,LDFJAC,M,N
         REAL X(N),FVEC(M)
         ----------
         FJAC and LDFJAC may be ignored     , if IOPT=1.
         REAL FJAC(LDFJAC,N)                , if IOPT=2.
         REAL FJAC(N)                       , if IOPT=3.
         ----------
           IFLAG will never be zero when FCN is called by SCOV.
         RETURN
         ----------
           If IFLAG=1, calculate the functions at X and return
           this vector in FVEC.
         RETURN
         ----------
```

```
      If IFLAG=2, calculate the full Jacobian at X and return
      this matrix in FJAC.  Note that IFLAG will never be 2 unless
      IOPT=2.  FVEC contains the function values at X and must
      not be altered.  FJAC(I,J) must be set to the derivative
      of FVEC(I) with respect to X(J).
    RETURN
    ----------
      If IFLAG=3, calculate the LDFJAC-th row of the Jacobian
      and return this vector in FJAC.  Note that IFLAG will
      never be 3 unless IOPT=3.  FJAC(J) must be set to
      the derivative of FVEC(LDFJAC) with respect to X(J).
    RETURN
    ----------
    END
```

The value of IFLAG should not be changed by FCN unless the user wants to terminate execution of SCOV. In this case, set IFLAG to a negative integer.

IOPT is an input variable which specifies how the Jacobian will be calculated. If IOPT=2 or 3, then the user must supply the Jacobian, as well as the function values, through the subroutine FCN. If IOPT=2, the user supplies the full Jacobian with one call to FCN. If IOPT=3, the user supplies one row of the Jacobian with each call. (In this manner, storage can be saved because the full Jacobian is not stored.) If IOPT=1, the code will approximate the Jacobian by forward differencing.

M is a positive integer input variable set to the number of functions.

N is a positive integer input variable set to the number of variables. N must not exceed M.

X is an array of length N. On input X must contain the value at which the covariance matrix is to be evaluated. This is usually the value for X returned from a successful run of SNLS1 (or SNLS1E). The value of X will not be changed.

FVEC is an output array of length M which contains the functions evaluated at X.

R is an output array. For IOPT=1 and 2, R is an M by N array. For IOPT=3, R is an N by N array. On output, if INFO=1, the upper N by N submatrix of R contains the covariance matrix evaluated at X.

LDR is a positive integer input variable which specifies the leading dimension of the array R. For IOPT=1 and 2, LDR must not be less than M. For IOPT=3, LDR must not be less than N.

INFO is an integer output variable. If the user has terminated execution, INFO is set to the (negative) value of IFLAG. See description of FCN. Otherwise, INFO is set as follows.

INFO = 0 Improper input parameters (M.LE.0 or N.LE.0).

```
          INFO = 1 Successful return.  The covariance matrix has been
                   calculated and stored in the upper N by N
                   submatrix of R.

          INFO = 2 The Jacobian matrix is singular for the input value
                   of X.  The covariance matrix cannot be calculated.
                   The upper N by N submatrix of R contains the QR
                   factorization of the Jacobian (probably not of
                   interest to the user).

      WA1 is a work array of length N.
      WA2 is a work array of length N.
      WA3 is a work array of length N.
      WA4 is a work array of length M.

***REFERENCES  (NONE)
***ROUTINES CALLED  ENORM, FDJAC3, QRFAC, RWUPDT, XERMSG
***REVISION HISTORY  (YYMMDD)
   810522  DATE WRITTEN
   890505  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900510  Fixed an error message.  (RWC)
   END PROLOGUE
```

SCPPLT

```
      SUBROUTINE SCPPLT (N, NELT, IA, JA, A, ISYM, IUNIT)
***BEGIN PROLOGUE  SCPPLT
***PURPOSE  Printer Plot of SLAP Column Format Matrix.
            Routine to print out a SLAP Column format matrix in a
            "printer plot" graphical representation.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  N1
***TYPE      SINGLE PRECISION (SCPPLT-S, DCPPLT-D)
***KEYWORDS  DIAGNOSTICS, LINEAR SYSTEM, SLAP SPARSE
***AUTHOR  Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, IUNIT
     REAL    A(NELT)

     CALL SCPPLT( N, NELT, IA, JA, A, ISYM, IUNIT )

 *Arguments:
 N       :IN       Integer
          Order of the Matrix.
          If N.gt.MAXORD, only the leading MAXORD x MAXORD
          submatrix will be printed.  (Currently MAXORD = 225.)
 NELT    :IN       Integer.
          Number of non-zeros stored in A.
 IA      :IN       Integer IA(NELT).
 JA      :IN       Integer JA(NELT).
 A       :IN       Real A(NELT).
          These arrays should hold the matrix A in the SLAP
          Column format.  See "Description", below.
 ISYM    :IN       Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the lower
          triangle of the matrix is stored.
 IUNIT   :IN       Integer.
          Fortran logical I/O device unit number to write the matrix
          to.  This unit must be connected in a system dependent fashion
          to a file or the console or you will get a nasty message
          from the Fortran I/O libraries.

 *Description:
       This routine prints out a SLAP  Column format matrix  to the
       Fortran logical I/O unit   number  IUNIT.  The  numbers them
       selves  are not printed  out, but   rather  a one  character
       representation of the numbers.   Elements of the matrix that
       are not represented in the (IA,JA,A)  arrays are  denoted by
       ' ' character (a blank).  Elements of A that are *ZERO* (and
       hence  should  really not be  stored) are  denoted  by a '0'
       character.  Elements of A that are *POSITIVE* are denoted by
       'D' if they are Diagonal elements  and '#' if  they are off
       Diagonal  elements.  Elements of  A that are *NEGATIVE* are
```

denoted by 'N' if they are Diagonal elements and '*' if they are off Diagonal elements.

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Cautions:

This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

*Portability:

This routine, as distributed, can generate lines up to 229 characters long. Some Fortran systems have more restricted line lengths. Change parameter MAXORD and the large number in FORMAT 1010 to reduce this line length.

***REFERENCES (NONE)
***ROUTINES CALLED (NONE)
***REVISION HISTORY (YYMMDD)

871119	DATE WRITTEN
881213	Previous REVISION DATE
890915	Made changes requested at July 1989 CML Meeting. (MKS)
890922	Numerous changes to prologue to make closer to SLATEC standard. (FNF)
890929	Numerous changes to reduce SP/DP differences. (FNF)
910411	Prologue converted to Version 4.0 format. (BAB)
920511	Added complete declaration section. (WRB)
921007	Replaced hard-wired 225 with parameter MAXORD. (FNF)
921021	Corrected syntax of CHARACTER declaration. (FNF)

```
930701  Updated CATEGORY section.  (FNF, WRB)
END PROLOGUE
```

SDASSL

```
      SUBROUTINE SDASSL (RES, NEQ, T, Y, YPRIME, TOUT, INFO, RTOL, ATOL,
     *   IDID, RWORK, LRW, IWORK, LIW, RPAR, IPAR, JAC)
***BEGIN PROLOGUE  SDASSL
***PURPOSE  This code solves a system of differential/algebraic
            equations of the form G(T,Y,YPRIME) = 0.
***LIBRARY   SLATEC (DASSL)
***CATEGORY  I1A2
***TYPE      SINGLE PRECISION (SDASSL-S, DDASSL-D)
***KEYWORDS  BACKWARD DIFFERENTIATION FORMULAS, DASSL,
             DIFFERENTIAL/ALGEBRAIC, IMPLICIT DIFFERENTIAL SYSTEMS
***AUTHOR  Petzold, Linda R., (LLNL)
             Computing and Mathematics Research Division
             Lawrence Livermore National Laboratory
             L - 316, P.O. Box 808,
             Livermore, CA.    94550
***DESCRIPTION

 *Usage:

      EXTERNAL RES, JAC
      INTEGER NEQ, INFO(N), IDID, LRW, LIW, IWORK(LIW), IPAR
      REAL T, Y(NEQ), YPRIME(NEQ), TOUT, RTOL, ATOL,
     *   RWORK(LRW), RPAR

      CALL SDASSL (RES, NEQ, T, Y, YPRIME, TOUT, INFO, RTOL, ATOL,
     *   IDID, RWORK, LRW, IWORK, LIW, RPAR, IPAR, JAC)


 *Arguments:

  RES:EXT     This is a subroutine which you provide to define the
              differential/algebraic system.

  NEQ:IN      This is the number of equations to be solved.

  T:INOUT     This is the current value of the independent variable.

  Y(*):INOUT  This array contains the solution components at T.

  YPRIME(*):INOUT  This array contains the derivatives of the solution
              components at T.

  TOUT:IN     This is a point at which a solution is desired.

  INFO(N):IN  The basic task of the code is to solve the system from T
              to TOUT and return an answer at TOUT.  INFO is an integer
              array which is used to communicate exactly how you want
              this task to be carried out.  (See below for details.)
              N must be greater than or equal to 15.

  RTOL,ATOL:INOUT  These quantities represent relative and absolute
              error tolerances which you provide to indicate how
              accurately you wish the solution to be computed.  You
              may choose them to be both scalars or else both vectors.
              Caution:  In Fortran 77, a scalar is not the same as an
                        array of length 1.  Some compilers may object
```

to using scalars for RTOL,ATOL.

IDID:OUT This scalar quantity is an indicator reporting what the code did. You must monitor this integer variable to decide what action to take next.

RWORK:WORK A real work array of length LRW which provides the code with needed storage space.

LRW:IN The length of RWORK. (See below for required length.)

IWORK:WORK An integer work array of length LIW which provides the code with needed storage space.

LIW:IN The length of IWORK. (See below for required length.)

RPAR,IPAR:IN These are real and integer parameter arrays which you can use for communication between your calling program and the RES subroutine (and the JAC subroutine)

JAC:EXT This is the name of a subroutine which you may choose to provide for defining a matrix of partial derivatives described below.

Quantities which may be altered by SDASSL are:
T, Y(*), YPRIME(*), INFO(1), RTOL, ATOL,
IDID, RWORK(*) AND IWORK(*)

*Description

Subroutine SDASSL uses the backward differentiation formulas of orders one through five to solve a system of the above form for Y and YPRIME. Values for Y and YPRIME at the initial time must be given as input. These values must be consistent, (that is, if T,Y,YPRIME are the given initial values, they must satisfy G(T,Y,YPRIME) = 0.). The subroutine solves the system from T to TOUT. It is easy to continue the solution to get results at additional TOUT. This is the interval mode of operation. Intermediate results can also be obtained easily by using the intermediate-output capability.

The following detailed description is divided into subsections:
1. Input required for the first call to SDASSL.
2. Output after any return from SDASSL.
3. What to do to continue the integration.
4. Error messages.

-------- INPUT -- WHAT TO DO ON THE FIRST CALL TO SDASSL ----------

The first call of the code is defined to be the start of each new problem. Read through the descriptions of all the following items, provide sufficient storage space for designated arrays, set appropriate variables for the initialization of the problem, and give information about how you want the problem to be solved.

RES -- Provide a subroutine of the form
```
      SUBROUTINE RES(T,Y,YPRIME,DELTA,IRES,RPAR,IPAR)
```
to define the system of differential/algebraic equations which is to be solved. For the given values

of T,Y and YPRIME, the subroutine should
return the residual of the differential/algebraic
system
    DELTA = G(T,Y,YPRIME)
(DELTA(*) is a vector of length NEQ which is
output for RES.)

Subroutine RES must not alter T,Y or YPRIME.
You must declare the name RES in an external
statement in your program that calls SDASSL.
You must dimension Y,YPRIME and DELTA in RES.

IRES is an integer flag which is always equal to
zero on input. Subroutine RES should alter IRES
only if it encounters an illegal value of Y or
a stop condition. Set IRES = -1 if an input value
is illegal, and SDASSL will try to solve the problem
without getting IRES = -1. If IRES = -2, SDASSL
will return control to the calling program
with IDID = -11.

RPAR and IPAR are real and integer parameter arrays which
you can use for communication between your calling program
and subroutine RES. They are not altered by SDASSL. If you
do not need RPAR or IPAR, ignore these parameters by treat-
ing them as dummy arguments. If you do choose to use them,
dimension them in your calling program and in RES as arrays
of appropriate length.

NEQ -- Set it to the number of differential equations.
(NEQ .GE. 1)

T -- Set it to the initial point of the integration.
T must be defined as a variable.

Y(*) -- Set this vector to the initial values of the NEQ solution
components at the initial point. You must dimension Y of
length at least NEQ in your calling program.

YPRIME(*) -- Set this vector to the initial values of the NEQ
first derivatives of the solution components at the initial
point. You must dimension YPRIME at least NEQ in your
calling program. If you do not know initial values of some
of the solution components, see the explanation of INFO(11).

TOUT -- Set it to the first point at which a solution
is desired. You can not take TOUT = T.
integration either forward in T (TOUT .GT. T) or
backward in T (TOUT .LT. T) is permitted.

The code advances the solution from T to TOUT using
step sizes which are automatically selected so as to
achieve the desired accuracy. If you wish, the code will
return with the solution and its derivative at
intermediate steps (intermediate-output mode) so that
you can monitor them, but you still must provide TOUT in
accord with the basic aim of the code.

The first step taken by the code is a critical one
because it must reflect how fast the solution changes near

```
        the initial point. The code automatically selects an
        initial step size which is practically always suitable for
        the problem. By using the fact that the code will not step
        past TOUT in the first step, you could, if necessary,
        restrict the length of the initial step size.

        For some problems it may not be permissible to integrate
        past a point TSTOP because a discontinuity occurs there
        or the solution or its derivative is not defined beyond
        TSTOP. When you have declared a TSTOP point (SEE INFO(4)
        and RWORK(1)), you have told the code not to integrate
        past TSTOP. In this case any TOUT beyond TSTOP is invalid
        input.

INFO(*) -- Use the INFO array to give the code more details about
        how you want your problem solved.  This array should be
        dimensioned of length 15, though SDASSL uses only the first
        eleven entries.  You must respond to all of the following
        items, which are arranged as questions.  The simplest use
        of the code corresponds to answering all questions as yes,
        i.e. setting all entries of INFO to 0.

     INFO(1) - This parameter enables the code to initialize
            itself. You must set it to indicate the start of every
            new problem.

        **** Is this the first call for this problem ...
              Yes - Set INFO(1) = 0
               No - Not applicable here.
                    See below for continuation calls.  ****

     INFO(2) - How much accuracy you want of your solution
            is specified by the error tolerances RTOL and ATOL.
            The simplest use is to take them both to be scalars.
            To obtain more flexibility, they can both be vectors.
            The code must be told your choice.

        **** Are both error tolerances RTOL, ATOL scalars ...
              Yes - Set INFO(2) = 0
                    and input scalars for both RTOL and ATOL
               No - Set INFO(2) = 1
                    and input arrays for both RTOL and ATOL ****

     INFO(3) - The code integrates from T in the direction
            of TOUT by steps. If you wish, it will return the
            computed solution and derivative at the next
            intermediate step (the intermediate-output mode) or
            TOUT, whichever comes first. This is a good way to
            proceed if you want to see the behavior of the solution.
            If you must have solutions at a great many specific
            TOUT points, this code will compute them efficiently.

        **** Do you want the solution only at
              TOUT (and not at the next intermediate step) ...
               Yes - Set INFO(3) = 0
                No - Set INFO(3) = 1 ****

     INFO(4) - To handle solutions at a great many specific
            values TOUT efficiently, this code may integrate past
            TOUT and interpolate to obtain the result at TOUT.
```

Sometimes it is not possible to integrate beyond some point TSTOP because the equation changes there or it is not defined past TSTOP. Then you must tell the code not to go past.

```
**** Can the integration be carried out without any
     restrictions on the independent variable T ...
      Yes - Set INFO(4)=0
       No - Set INFO(4)=1
            and define the stopping point TSTOP by
            setting RWORK(1)=TSTOP ****
```

INFO(5) - To solve differential/algebraic problems it is necessary to use a matrix of partial derivatives of the system of differential equations. If you do not provide a subroutine to evaluate it analytically (see description of the item JAC in the call list), it will be approximated by numerical differencing in this code. although it is less trouble for you to have the code compute partial derivatives by numerical differencing, the solution will be more reliable if you provide the derivatives via JAC. Sometimes numerical differencing is cheaper than evaluating derivatives in JAC and sometimes it is not - this depends on your problem.

```
**** Do you want the code to evaluate the partial
     derivatives automatically by numerical differences ...
        Yes - Set INFO(5)=0
         No - Set INFO(5)=1
       and provide subroutine JAC for evaluating the
       matrix of partial derivatives ****
```

INFO(6) - SDASSL will perform much better if the matrix of partial derivatives, DG/DY + CJ*DG/DYPRIME, (here CJ is a scalar determined by SDASSL) is banded and the code is told this. In this case, the storage needed will be greatly reduced, numerical differencing will be performed much cheaper, and a number of important algorithms will execute much faster. The differential equation is said to have half-bandwidths ML (lower) and MU (upper) if equation i involves only unknowns Y(J) with

```
                I-ML .LE. J .LE. I+MU
```

for all I=1,2,...,NEQ. Thus, ML and MU are the widths of the lower and upper parts of the band, respectively, with the main diagonal being excluded. If you do not indicate that the equation has a banded matrix of partial derivatives, the code works with a full matrix of NEQ**2 elements (stored in the conventional way). Computations with banded matrices cost less time and storage than with full matrices if 2*ML+MU .LT. NEQ. If you tell the code that the matrix of partial derivatives has a banded structure and you want to provide subroutine JAC to compute the partial derivatives, then you must be careful to store the elements of the matrix in the special form indicated in the description of JAC.

```
**** Do you want to solve the problem using a full
     (dense) matrix (and not a special banded
     structure) ...
```

```
        Yes - Set INFO(6)=0
         No - Set INFO(6)=1
              and provide the lower (ML) and upper (MU)
              bandwidths by setting
              IWORK(1)=ML
              IWORK(2)=MU ****


INFO(7) -- You can specify a maximum (absolute value of)
      stepsize, so that the code
      will avoid passing over very
      large regions.

  ****  Do you want the code to decide
        on its own maximum stepsize?
        Yes - Set INFO(7)=0
         No - Set INFO(7)=1
              and define HMAX by setting
              RWORK(2)=HMAX ****

INFO(8) -- Differential/algebraic problems
      may occasionally suffer from
      severe scaling difficulties on the
      first step. If you know a great deal
      about the scaling of your problem, you can
      help to alleviate this problem by
      specifying an initial stepsize HO.

  ****  Do you want the code to define
        its own initial stepsize?
        Yes - Set INFO(8)=0
         No - Set INFO(8)=1
              and define HO by setting
              RWORK(3)=HO ****

INFO(9) -- If storage is a severe problem,
      you can save some locations by
      restricting the maximum order MAXORD.
      the default value is 5. for each
      order decrease below 5, the code
      requires NEQ fewer locations, however
      it is likely to be slower. In any
      case, you must have 1 .LE. MAXORD .LE. 5
  ****  Do you want the maximum order to
        default to 5?
        Yes - Set INFO(9)=0
         No - Set INFO(9)=1
              and define MAXORD by setting
              IWORK(3)=MAXORD ****

INFO(10) --If you know that the solutions to your equations
       will always be nonnegative, it may help to set this
       parameter. However, it is probably best to
       try the code without using this option first,
       and only to use this option if that doesn't
       work very well.
   ****  Do you want the code to solve the problem without
         invoking any special nonnegativity constraints?
          Yes - Set INFO(10)=0
           No - Set INFO(10)=1
```

```
INFO(11) --SDASSL normally requires the initial T,
       Y, and YPRIME to be consistent. That is,
       you must have G(T,Y,YPRIME) = 0 at the initial
       time. If you do not know the initial
       derivative precisely, you can let SDASSL try
       to compute it.
 ****   Are the initial T, Y, YPRIME consistent?
        Yes - Set INFO(11) = 0
         No - Set INFO(11) = 1,
              and set YPRIME to an initial approximation
              to YPRIME.  (If you have no idea what
              YPRIME should be, set it to zero. Note
              that the initial Y should be such
              that there must exist a YPRIME so that
              G(T,Y,YPRIME) = 0.)

RTOL, ATOL -- You must assign relative (RTOL) and absolute (ATOL
       error tolerances to tell the code how accurately you
       want the solution to be computed.  They must be defined
       as variables because the code may change them.  You
       have two choices --
             Both RTOL and ATOL are scalars. (INFO(2)=0)
             Both RTOL and ATOL are vectors. (INFO(2)=1)
       in either case all components must be non-negative.

       The tolerances are used by the code in a local error
       test at each step which requires roughly that
             ABS(LOCAL ERROR) .LE. RTOL*ABS(Y)+ATOL
       for each vector component.
       (More specifically, a root-mean-square norm is used to
       measure the size of vectors, and the error test uses the
       magnitude of the solution at the beginning of the step.)

       The true (global) error is the difference between the
       true solution of the initial value problem and the
       computed approximation.  Practically all present day
       codes, including this one, control the local error at
       each step and do not even attempt to control the global
       error directly.
       Usually, but not always, the true accuracy of the
       computed Y is comparable to the error tolerances. This
       code will usually, but not always, deliver a more
       accurate solution if you reduce the tolerances and
       integrate again.  By comparing two such solutions you
       can get a fairly reliable idea of the true error in the
       solution at the bigger tolerances.

       Setting ATOL=0. results in a pure relative error test on
       that component.  Setting RTOL=0. results in a pure
       absolute error test on that component.  A mixed test
       with non-zero RTOL and ATOL corresponds roughly to a
       relative error test when the solution component is much
       bigger than ATOL and to an absolute error test when the
       solution component is smaller than the threshhold ATOL.

       The code will not attempt to compute a solution at an
       accuracy unreasonable for the machine being used.  It will
       advise you if you ask for too much accuracy and inform
       you as to the maximum accuracy it believes possible.
```

```
RWORK(*) --  Dimension this real work array of length LRW in your
       calling program.

LRW -- Set it to the declared length of the RWORK array.
             You must have
                  LRW .GE. 40+(MAXORD+4)*NEQ+NEQ**2
             for the full (dense) JACOBIAN case (when INFO(6)=0), or
                  LRW .GE. 40+(MAXORD+4)*NEQ+(2*ML+MU+1)*NEQ
             for the banded user-defined JACOBIAN case
             (when INFO(5)=1 and INFO(6)=1), or
                   LRW .GE. 40+(MAXORD+4)*NEQ+(2*ML+MU+1)*NEQ
                         +2*(NEQ/(ML+MU+1)+1)
             for the banded finite-difference-generated JACOBIAN case
             (when INFO(5)=0 and INFO(6)=1)

IWORK(*) --  Dimension this integer work array of length LIW in
       your calling program.

LIW -- Set it to the declared length of the IWORK array.
             You must have LIW .GE. 20+NEQ

RPAR, IPAR -- These are parameter arrays, of real and integer
       type, respectively.  You can use them for communication
       between your program that calls SDASSL and the
       RES subroutine (and the JAC subroutine).  They are not
       altered by SDASSL.  If you do not need RPAR or IPAR,
       ignore these parameters by treating them as dummy
       arguments.  If you do choose to use them, dimension
       them in your calling program and in RES (and in JAC)
       as arrays of appropriate length.

JAC -- If you have set INFO(5)=0, you can ignore this parameter
       by treating it as a dummy argument.  Otherwise, you must
       provide a subroutine of the form
           SUBROUTINE JAC(T,Y,YPRIME,PD,CJ,RPAR,IPAR)
       to define the matrix of partial derivatives
           PD=DG/DY+CJ*DG/DYPRIME
       CJ is a scalar which is input to JAC.
       For the given values of T,Y,YPRIME, the
       subroutine must evaluate the non-zero partial
       derivatives for each equation and each solution
       component, and store these values in the
       matrix PD.  The elements of PD are set to zero
       before each call to JAC so only non-zero elements
       need to be defined.

       Subroutine JAC must not alter T,Y,(*),YPRIME(*), or CJ.
       You must declare the name JAC in an EXTERNAL statement in
       your program that calls SDASSL.  You must dimension Y,
       YPRIME and PD in JAC.

       The way you must store the elements into the PD matrix
       depends on the structure of the matrix which you
       indicated by INFO(6).
             *** INFO(6)=0 -- Full (dense) matrix ***
                 Give PD a first dimension of NEQ.
                 When you evaluate the (non-zero) partial derivative
                 of equation I with respect to variable J, you must
                 store it in PD according to
```

```
                PD(I,J) = "DG(I)/DY(J)+CJ*DG(I)/DYPRIME(J)"
            *** INFO(6)=1 -- Banded JACOBIAN with ML lower and MU
                upper diagonal bands (refer to INFO(6) description
                of ML and MU) ***
                Give PD a first dimension of 2*ML+MU+1.
                when you evaluate the (non-zero) partial derivative
                of equation I with respect to variable J, you must
                store it in PD according to
                IROW = I - J + ML + MU + 1
                PD(IROW,J) = "DG(I)/DY(J)+CJ*DG(I)/DYPRIME(J)"
```

RPAR and IPAR are real and integer parameter arrays which you can use for communication between your calling program and your JACOBIAN subroutine JAC. They are not altered by SDASSL. If you do not need RPAR or IPAR, ignore these parameters by treating them as dummy arguments. If you do choose to use them, dimension them in your calling program and in JAC as arrays of appropriate length.

OPTIONALLY REPLACEABLE NORM ROUTINE:

SDASSL uses a weighted norm SDANRM to measure the size of vectors such as the estimated error in each step. A FUNCTION subprogram

```
  REAL FUNCTION SDANRM(NEQ,V,WT,RPAR,IPAR)
  DIMENSION V(NEQ),WT(NEQ)
```

is used to define this norm. Here, V is the vector whose norm is to be computed, and WT is a vector of weights. A SDANRM routine has been included with SDASSL which computes the weighted root-mean-square norm given by

```
  SDANRM=SQRT((1/NEQ)*SUM(V(I)/WT(I))**2)
```

this norm is suitable for most problems. In some special cases, it may be more convenient and/or efficient to define your own norm by writing a function subprogram to be called instead of SDANRM. This should, however, be attempted only after careful thought and consideration.

-------- OUTPUT -- AFTER ANY RETURN FROM SDASSL ----------

The principal aim of the code is to return a computed solution at TOUT, although it is also possible to obtain intermediate results along the way. To find out whether the code achieved its goal or if the integration process was interrupted before the task was completed, you must check the IDID parameter.

```
T -- The solution was successfully advanced to the
             output value of T.

Y(*) -- Contains the computed solution approximation at T.

YPRIME(*) -- Contains the computed derivative
             approximation at T.

IDID -- Reports what the code did.
```

```
          *** Task completed ***
     Reported by positive values of IDID

IDID = 1 -- A step was successfully taken in the
        intermediate-output mode. The code has not
        yet reached TOUT.

IDID = 2 -- The integration to TSTOP was successfully
        completed (T=TSTOP) by stepping exactly to TSTOP.

IDID = 3 -- The integration to TOUT was successfully
        completed (T=TOUT) by stepping past TOUT.
        Y(*) is obtained by interpolation.
        YPRIME(*) is obtained by interpolation.

         *** Task interrupted ***
     Reported by negative values of IDID

IDID = -1 -- A large amount of work has been expended.
        (About 500 steps)

IDID = -2 -- The error tolerances are too stringent.

IDID = -3 -- The local error test cannot be satisfied
        because you specified a zero component in ATOL
        and the corresponding computed solution
        component is zero. Thus, a pure relative error
        test is impossible for this component.

IDID = -6 -- SDASSL had repeated error test
        failures on the last attempted step.

IDID = -7 -- The corrector could not converge.

IDID = -8 -- The matrix of partial derivatives
        is singular.

IDID = -9 -- The corrector could not converge.
        there were repeated error test failures
        in this step.

IDID =-10 -- The corrector could not converge
        because IRES was equal to minus one.

IDID =-11 -- IRES equal to -2 was encountered
        and control is being returned to the
        calling program.

IDID =-12 -- SDASSL failed to compute the initial
        YPRIME.



IDID = -13,..,-32 -- Not applicable for this code

         *** Task terminated ***
     Reported by the value of IDID=-33

IDID = -33 -- The code has encountered trouble from which
```

```
                it cannot recover. A message is printed
                explaining the trouble and control is returned
                to the calling program. For example, this occurs
                when invalid input is detected.

RTOL, ATOL -- These quantities remain unchanged except when
              IDID = -2. In this case, the error tolerances have been
              increased by the code to values which are estimated to
              be appropriate for continuing the integration. However,
              the reported solution at T was obtained using the input
              values of RTOL and ATOL.

RWORK, IWORK -- Contain information which is usually of no
              interest to the user but necessary for subsequent calls.
              However, you may find use for

              RWORK(3)--Which contains the step size H to be
                      attempted on the next step.

              RWORK(4)--Which contains the current value of the
                      independent variable, i.e., the farthest point
                      integration has reached. This will be different
                      from T only when interpolation has been
                      performed (IDID=3).

              RWORK(7)--Which contains the stepsize used
                      on the last successful step.

              IWORK(7)--Which contains the order of the method to
                      be attempted on the next step.

              IWORK(8)--Which contains the order of the method used
                      on the last step.

              IWORK(11)--Which contains the number of steps taken so
                       far.

              IWORK(12)--Which contains the number of calls to RES
                       so far.

              IWORK(13)--Which contains the number of evaluations of
                       the matrix of partial derivatives needed so
                       far.

              IWORK(14)--Which contains the total number
                       of error test failures so far.

              IWORK(15)--Which contains the total number
                       of convergence test failures so far.
                       (includes singular iteration matrix
                       failures.)


-------- INPUT -- WHAT TO DO TO CONTINUE THE INTEGRATION ------------
                  (CALLS AFTER THE FIRST)

This code is organized so that subsequent calls to continue the
integration involve little (if any) additional effort on your
part. You must monitor the IDID parameter in order to determine
what to do next.
```

Recalling that the principal task of the code is to integrate from T to TOUT (the interval mode), usually all you will need to do is specify a new TOUT upon reaching the current TOUT.

Do not alter any quantity not specifically permitted below, in particular do not alter NEQ,T,Y(*),YPRIME(*),RWORK(*),IWORK(*) or the differential equation in subroutine RES. Any such alteration constitutes a new problem and must be treated as such, i.e., you must start afresh.

You cannot change from vector to scalar error control or vice versa (INFO(2)), but you can change the size of the entries of RTOL, ATOL. Increasing a tolerance makes the equation easier to integrate. Decreasing a tolerance will make the equation harder to integrate and should generally be avoided.

You can switch from the intermediate-output mode to the interval mode (INFO(3)) or vice versa at any time.

If it has been necessary to prevent the integration from going past a point TSTOP (INFO(4), RWORK(1)), keep in mind that the code will not integrate to any TOUT beyond the currently specified TSTOP. Once TSTOP has been reached you must change the value of TSTOP or set INFO(4)=0. You may change INFO(4) or TSTOP at any time but you must supply the value of TSTOP in RWORK(1) whenever you set INFO(4)=1.

Do not change INFO(5), INFO(6), IWORK(1), or IWORK(2) unless you are going to restart the code.

```
               *** Following a completed task ***
If
   IDID = 1, call the code again to continue the integration
                another step in the direction of TOUT.

   IDID = 2 or 3, define a new TOUT and call the code again.
                TOUT must be different from T. You cannot change
                the direction of integration without restarting.

               *** Following an interrupted task ***
             To show the code that you realize the task was
             interrupted and that you want to continue, you
             must take appropriate action and set INFO(1) = 1
If
  IDID = -1, The code has taken about 500 steps.
                If you want to continue, set INFO(1) = 1 and
                call the code again. An additional 500 steps
                will be allowed.

  IDID = -2, The error tolerances RTOL, ATOL have been
                increased to values the code estimates appropriate
                for continuing. You may want to change them
                yourself. If you are sure you want to continue
                with relaxed error tolerances, set INFO(1)=1 and
                call the code again.

  IDID = -3, A solution component is zero and you set the
                corresponding component of ATOL to zero. If you
                are sure you want to continue, you must first
```

```
              alter the error criterion to use positive values
              for those components of ATOL corresponding to zero
              solution components, then set INFO(1)=1 and call
              the code again.

IDID = -4,-5  --- Cannot occur with this code.

IDID = -6, Repeated error test failures occurred on the
              last attempted step in SDASSL. A singularity in the
              solution may be present. If you are absolutely
              certain you want to continue, you should restart
              the integration. (Provide initial values of Y and
              YPRIME which are consistent)

IDID = -7, Repeated convergence test failures occurred
              on the last attempted step in SDASSL. An inaccurate
              or ill-conditioned JACOBIAN may be the problem. If
              you are absolutely certain you want to continue, you
              should restart the integration.

IDID = -8, The matrix of partial derivatives is singular.
              Some of your equations may be redundant.
              SDASSL cannot solve the problem as stated.
              It is possible that the redundant equations
              could be removed, and then SDASSL could
              solve the problem. It is also possible
              that a solution to your problem either
              does not exist or is not unique.

IDID = -9, SDASSL had multiple convergence test
              failures, preceded by multiple error
              test failures, on the last attempted step.
              It is possible that your problem
              is ill-posed, and cannot be solved
              using this code. Or, there may be a
              discontinuity or a singularity in the
              solution. If you are absolutely certain
              you want to continue, you should restart
              the integration.

IDID =-10, SDASSL had multiple convergence test failures
              because IRES was equal to minus one.
              If you are absolutely certain you want
              to continue, you should restart the
              integration.

IDID =-11, IRES=-2 was encountered, and control is being
              returned to the calling program.

IDID =-12, SDASSL failed to compute the initial YPRIME.
              This could happen because the initial
              approximation to YPRIME was not very good, or
              if a YPRIME consistent with the initial Y
              does not exist. The problem could also be caused
              by an inaccurate or singular iteration matrix.

IDID = -13,..,-32  --- Cannot occur with this code.

             *** Following a terminated task ***
```

If IDID= -33, you cannot continue the solution of this problem.
An attempt to do so will result in your
run being terminated.

-------- ERROR MESSAGES ----------

The SLATEC error print routine XERMSG is called in the event of unsuccessful completion of a task. Most of these are treated as "recoverable errors", which means that (unless the user has directed otherwise) control will be returned to the calling program for possible action after the message has been printed.

In the event of a negative value of IDID other than -33, an appropriate message is printed and the "error number" printed by XERMSG is the value of IDID. There are quite a number of illegal input errors that can lead to a returned value IDID=-33. The conditions and their printed "error numbers" are as follows:

Error number	Condition
1	Some element of INFO vector is not zero or one.
2	NEQ .le. 0
3	MAXORD not in range.
4	LRW is less than the required length for RWORK.
5	LIW is less than the required length for IWORK.
6	Some element of RTOL is .lt. 0
7	Some element of ATOL is .lt. 0
8	All elements of RTOL and ATOL are zero.
9	INFO(4)=1 and TSTOP is behind TOUT.
10	HMAX .lt. 0.0
11	TOUT is behind T.
12	INFO(8)=1 and H0=0.0
13	Some element of WT is .le. 0.0
14	TOUT is too close to T to start integration.
15	INFO(4)=1 and TSTOP is behind T.
16	--(Not used in this version)--
17	ML illegal. Either .lt. 0 or .gt. NEQ
18	MU illegal. Either .lt. 0 or .gt. NEQ
19	TOUT = T.

If SDASSL is called again without any action taken to remove the cause of an unsuccessful return, XERMSG will be called with a fatal error flag, which will cause unconditional termination of the program. There are two such fatal errors:

Error number -998: The last step was terminated with a negative value of IDID other than -33, and no appropriate action was taken.

Error number -999: The previous call was terminated because of illegal input (IDID=-33) and there is illegal input in the present call, as well. (Suspect infinite loop.)

***REFERENCES A DESCRIPTION OF DASSL: A DIFFERENTIAL/ALGEBRAIC
SYSTEM SOLVER, L. R. PETZOLD, SAND82-8637,
SANDIA NATIONAL LABORATORIES, SEPTEMBER 1982.

```
***ROUTINES CALLED  R1MACH, SDAINI, SDANRM, SDASTP, SDATRP, SDAWTS,
                    XERMSG
***REVISION HISTORY  (YYMMDD)
   830315  DATE WRITTEN
   880387  Code changes made.  All common statements have been
           replaced by a DATA statement, which defines pointers into
           RWORK, and PARAMETER statements which define pointers
           into IWORK.  As well the documentation has gone through
           grammatical changes.
   881005  The prologue has been changed to mixed case.
           The subordinate routines had revision dates changed to
           this date, although the documentation for these routines
           is all upper case.  No code changes.
   890511  Code changes made.  The DATA statement in the declaration
           section of SDASSL was replaced with a PARAMETER
           statement.  Also the statement S = 100.E0 was removed
           from the top of the Newton iteration in SDASTP.
           The subordinate routines had revision dates changed to
           this date.
   890517  The revision date syntax was replaced with the revision
           history syntax.  Also the "DECK" comment was added to
           the top of all subroutines.  These changes are consistent
           with new SLATEC guidelines.
           The subordinate routines had revision dates changed to
           this date.  No code changes.
   891013  Code changes made.
           Removed all occurrences of FLOAT.  All operations
           are now performed with "mixed-mode" arithmetic.
           Also, specific function names were replaced with generic
           function names to be consistent with new SLATEC guidelines.
           In particular:
              Replaced AMIN1 with MIN everywhere.
              Replaced MIN0 with MIN everywhere.
              Replaced AMAX1 with MAX everywhere.
              Replaced MAX0 with MAX everywhere.
           Also replaced REVISION DATE with REVISION HISTORY in all
           subordinate routines.
   901004  Miscellaneous changes to prologue to complete conversion
           to SLATEC 4.0 format.  No code changes.  (F.N.Fritsch)
   901009  Corrected GAMS classification code and converted subsidiary
           routines to 4.0 format.  No code changes.  (F.N.Fritsch)
   901010  Converted XERRWV calls to XERMSG calls.  (R.Clemens, AFWL)
   901019  Code changes made.
           Merged SLATEC 4.0 changes with previous changes made
           by C. Ulrich.  Below is a history of the changes made by
           C. Ulrich. (Changes in subsidiary routines are implied
           by this history)
           891228  Bug was found and repaired inside the SDASSL
                   and SDAINI routines.  SDAINI was incorrectly
                   returning the initial T with Y and YPRIME
                   computed at T+H.  The routine now returns T+H
                   rather than the initial T.
                   Cosmetic changes made to SDASTP.
           900904  Three modifications were made to fix a bug (inside
                   SDASSL) re interpolation for continuation calls and
                   cases where TN is very close to TSTOP:

                   1) In testing for whether H is too large, just
                      compare H to (TSTOP - TN), rather than
                      (TSTOP - TN) * (1-4*UROUND), and set H to
```

```
                    TSTOP - TN.  This will force SDASTP to step
                    exactly to TSTOP under certain situations
                    (i.e. when H returned from SDASTP would otherwise
                    take TN beyond TSTOP).

                 2) Inside the SDASTP loop, interpolate exactly to
                    TSTOP if TN is very close to TSTOP (rather than
                    interpolating to within roundoff of TSTOP).

                 3) Modified IDID description for IDID = 2 to say
                    that the solution is returned by stepping exactly
                    to TSTOP, rather than TOUT.  (In some cases the
                    solution is actually obtained by extrapolating
                    over a distance near unit roundoff to TSTOP,
                    but this small distance is deemed acceptable in
                    these circumstances.)
901026  Added explicit declarations for all variables and minor
        cosmetic changes to prologue, removed unreferenced labels,
        and improved XERMSG calls.  (FNF)
901030  Added ERROR MESSAGES section and reworked other sections to
        be of more uniform format.  (FNF)
910624  Fixed minor bug related to HMAX (six lines after label
        525).  (LRP)
END PROLOGUE
```

SDOT

```
      REAL FUNCTION SDOT (N, SX, INCX, SY, INCY)
***BEGIN PROLOGUE  SDOT
***PURPOSE  Compute the inner product of two vectors.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A4
***TYPE      SINGLE PRECISION (SDOT-S, DDOT-D, CDOTU-C)
***KEYWORDS  BLAS, INNER PRODUCT, LINEAR ALGEBRA, VECTOR
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
           Kincaid, D. R., (U. of Texas)
           Krogh, F. T., (JPL)
***DESCRIPTION

                B L A S  Subprogram
    Description of Parameters

     --Input--
        N  number of elements in input vector(s)
       SX  single precision vector with N elements
     INCX  storage spacing between elements of SX
       SY  single precision vector with N elements
     INCY  storage spacing between elements of SY

     --Output--
     SDOT  single precision dot product (zero if N .LE. 0)

     Returns the dot product of single precision SX and SY.
     SDOT = sum for I = 0 to N-1 of  SX(LX+I*INCX) * SY(LY+I*INCY),
     where LX = 1 if INCX .GE. 0, else LX = 1+(1-N)*INCX, and LY is
     defined in a similar way using INCY.

***REFERENCES  C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T.
                Krogh, Basic linear algebra subprograms for Fortran
                usage, Algorithm No. 539, Transactions on Mathematical
                Software 5, 3 (September 1979), pp. 308-323.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   791001  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920310  Corrected definition of LX in DESCRIPTION.  (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SDRIV1

```
      SUBROUTINE SDRIV1 (N, T, Y, F, TOUT, MSTATE, EPS, LENW,
     8   IERFLG)
***BEGIN PROLOGUE  SDRIV1
***PURPOSE  The function of SDRIV1 is to solve N (200 or fewer)
            ordinary differential equations of the form
            dY(I)/dT = F(Y(I),T), given the initial conditions
            Y(I) = YI.  SDRIV1 uses single precision arithmetic.
***LIBRARY   SLATEC (SDRIVE)
***CATEGORY  I1A2, I1A1B
***TYPE      SINGLE PRECISION (SDRIV1-S, DDRIV1-D, CDRIV1-C)
***KEYWORDS  GEAR'S METHOD, INITIAL VALUE PROBLEMS, ODE,
             ORDINARY DIFFERENTIAL EQUATIONS, SDRIVE, SINGLE PRECISION,
             STIFF
***AUTHOR  Kahaner, D. K., (NIST)
             National Institute of Standards and Technology
             Gaithersburg, MD  20899
           Sutherland, C. D., (LANL)
             Mail Stop D466
             Los Alamos National Laboratory
             Los Alamos, NM  87545
***DESCRIPTION

   Version 92.1

  I.  CHOOSING THE CORRECT ROUTINE  ..................................

      SDRIV
      DDRIV
      CDRIV
            These are the generic names for three packages for solving
            initial value problems for ordinary differential equations.
            SDRIV uses single precision arithmetic.  DDRIV uses double
            precision arithmetic.  CDRIV allows complex-valued
            differential equations, integrated with respect to a single,
            real, independent variable.

    As an aid in selecting the proper program, the following is a
    discussion of the important options or restrictions associated with
    each program:

      A. SDRIV1 should be tried first for those routine problems with
         no more than 200 differential equations (SDRIV2 and SDRIV3
         have no such restriction.)  Internally this routine has two
         important technical defaults:
           1. Numerical approximation of the Jacobian matrix of the
              right hand side is used.
           2. The stiff solver option is used.
         Most users of SDRIV1 should not have to concern themselves
         with these details.

      B. SDRIV2 should be considered for those problems for which
         SDRIV1 is inadequate.  For example, SDRIV1 may have difficulty
         with problems having zero initial conditions and zero
         derivatives.  In this case SDRIV2, with an appropriate value
         of the parameter EWT, should perform more efficiently.  SDRIV2
         provides three important additional options:
```

1. The nonstiff equation solver (as well as the stiff solver) is available.
2. The root-finding option is available.
3. The program can dynamically select either the non-stiff or the stiff methods.

Internally this routine also defaults to the numerical approximation of the Jacobian matrix of the right hand side.

C. SDRIV3 is the most flexible, and hence the most complex, of the programs. Its important additional features include:

1. The ability to exploit band structure in the Jacobian matrix.
2. The ability to solve some implicit differential equations, i.e., those having the form:

```
A(Y,T)*dY/dT = F(Y,T).
```

3. The option of integrating in the one step mode.
4. The option of allowing the user to provide a routine which computes the analytic Jacobian matrix of the right hand side.
5. The option of allowing the user to provide a routine which does all the matrix algebra associated with corrections to the solution components.

II. PARAMETERS .....

The user should use parameter names in the call sequence of SDRIV1 for those quantities whose value may be altered by SDRIV1. The parameters in the call sequence are:

N = (Input) The number of differential equations, N .LE. 200

T = The independent variable. On input for the first call, T is the initial point. On output, T is the point at which the solution is given.

Y = The vector of dependent variables. Y is used as input on the first call, to set the initial values. On output, Y is the computed solution vector. This array Y is passed in the call sequence of the user-provided routine F. Thus parameters required by F can be stored in this array in components N+1 and above. (Note: Changes by the user to the first N components of this array will take effect only after a restart, i.e., after setting MSTATE to +1(-1).)

F = A subroutine supplied by the user. The name must be declared EXTERNAL in the user's calling program. This subroutine is of the form:

```
      SUBROUTINE F (N, T, Y, YDOT)
      REAL Y(*), YDOT(*)
        .
        .
      YDOT(1) = ...
        .
        .
      YDOT(N) = ...
      END (Sample)
```

This computes YDOT = F(Y,T), the right hand side of the differential equations. Here Y is a vector of length at least N. The actual length of Y is determined by the user's declaration in the program which calls SDRIV1.

Thus the dimensioning of Y in F, while required by FORTRAN convention, does not actually allocate any storage. When this subroutine is called, the first N components of Y are intermediate approximations to the solution components. The user should not alter these values. Here YDOT is a vector of length N. The user should only compute YDOT(I) for I from 1 to N. Normally a return from F passes control back to SDRIV1. However, if the user would like to abort the calculation, i.e., return control to the program which calls SDRIV1, he should set N to zero. SDRIV1 will signal this by returning a value of MSTATE equal to +5(-5). Altering the value of N in F has no effect on the value of N in the call sequence of SDRIV1.

TOUT = (Input) The point at which the solution is desired.

MSTATE = An integer describing the status of integration. The user must initialize MSTATE to +1 or -1. If MSTATE is positive, the routine will integrate past TOUT and interpolate the solution. This is the most efficient mode. If MSTATE is negative, the routine will adjust its internal step to reach TOUT exactly (useful if a singularity exists beyond TOUT.) The meaning of the magnitude of MSTATE:

1 (Input) Means the first call to the routine. This value must be set by the user. On all subsequent calls the value of MSTATE should be tested by the user. Unless SDRIV1 is to be reinitialized, only the sign of MSTATE may be changed by the user. (As a convenience to the user who may wish to put out the initial conditions, SDRIV1 can be called with MSTATE=+1(-1), and TOUT=T. In this case the program will return with MSTATE unchanged, i.e., MSTATE=+1(-1).)

2 (Output) Means a successful integration. If a normal continuation is desired (i.e., a further integration in the same direction), simply advance TOUT and call again. All other parameters are automatically set.

3 (Output)(Unsuccessful) Means the integrator has taken 1000 steps without reaching TOUT. The user can continue the integration by simply calling SDRIV1 again.

4 (Output)(Unsuccessful) Means too much accuracy has been requested. EPS has been increased to a value the program estimates is appropriate. The user can continue the integration by simply calling SDRIV1 again.

5 (Output)(Unsuccessful) N has been set to zero in SUBROUTINE F.

6 (Output)(Successful) For MSTATE negative, T is beyond TOUT. The solution was obtained by interpolation. The user can continue the integration by simply advancing TOUT and calling SDRIV1 again.

7 (Output)(Unsuccessful) The solution could not be obtained. The value of IERFLG (see description below) for a "Recoverable" situation indicates the type of difficulty encountered: either an illegal value for a parameter or an inability to continue the solution. For this condition the user should take corrective action and reset MSTATE to +1(-1) before

calling SDRIV1 again. Otherwise the program will terminate the run.

EPS = On input, the requested relative accuracy in all solution components. On output, the adjusted relative accuracy if the input value was too small. The value of EPS should be set as large as is reasonable, because the amount of work done by SDRIV1 increases as EPS decreases.

WORK
LENW = (Input)
WORK is an array of LENW real words used internally for temporary storage. The user must allocate space for this array in the calling program by a statement such as

```
REAL WORK(...)
```

The length of WORK should be at least N*N + 11*N + 300 and LENW should be set to the value used. The contents of WORK should not be disturbed between calls to SDRIV1.

IERFLG = An error flag. The error number associated with a diagnostic message (see Section IV-A below) is the same as the corresponding value of IERFLG. The meaning of IERFLG:

- 0 The routine completed successfully. (No message is issued.)
- 3 (Warning) The number of steps required to reach TOUT exceeds 1000 .
- 4 (Warning) The value of EPS is too small.
- 11 (Warning) For MSTATE negative, T is beyond TOUT. The solution was obtained by interpolation.
- 15 (Warning) The integration step size is below the roundoff level of T. (The program issues this message as a warning but does not return control to the user.)
- 21 (Recoverable) N is greater than 200 .
- 22 (Recoverable) N is not positive.
- 26 (Recoverable) The magnitude of MSTATE is either 0 or greater than 7 .
- 27 (Recoverable) EPS is less than zero.
- 32 (Recoverable) Insufficient storage has been allocated for the WORK array.
- 41 (Recoverable) The integration step size has gone to zero.
- 42 (Recoverable) The integration step size has been reduced about 50 times without advancing the solution. The problem setup may not be correct.
- 999 (Fatal) The magnitude of MSTATE is 7 .

III. USAGE .....

```
      PROGRAM SAMPLE
      EXTERNAL F
      REAL ALFA, EPS, T, TOUT
C                                N is the number of equations
      PARAMETER(ALFA = 1.E0, N = 3, LENW = N*N + 11*N + 300)
      REAL WORK(LENW), Y(N+1)
C                                                Initial point
      T = 0.00001E0
C                                      Set initial conditions
      Y(1) = 10.E0
```

```
      Y(2) = 0.E0
      Y(3) = 10.E0
C                                                       Pass parameter
      Y(4) = ALFA
      TOUT = T
      MSTATE = 1
      EPS = .001E0
 10   CALL SDRIV1 (N, T, Y, F, TOUT, MSTATE, EPS, WORK, LENW,
     8             IERFLG)
      IF (MSTATE .GT. 2) STOP
      WRITE(*, '(4E12.3)') TOUT, (Y(I), I=1,3)
      TOUT = 10.E0*TOUT
      IF (TOUT .LT. 50.E0) GO TO 10
      END

      SUBROUTINE F (N, T, Y, YDOT)
      REAL ALFA, T, Y(*), YDOT(*)
      ALFA = Y(N+1)
      YDOT(1) = 1.E0 + ALFA*(Y(2) - Y(1)) - Y(1)*Y(3)
      YDOT(2) = ALFA*(Y(1) - Y(2)) - Y(2)*Y(3)
      YDOT(3) = 1.E0 - Y(3)*(Y(1) + Y(2))
      END
```

IV. OTHER COMMUNICATION TO THE USER .....

A. The solver communicates to the user through the parameters above. In addition it writes diagnostic messages through the standard error handling program XERMSG. A complete description of XERMSG is given in "Guide to the SLATEC Common Mathematical Library" by Kirby W. Fong et al.. At installations which do not have this error handling package the short but serviceable routine, XERMSG, available with this package, can be used. That program uses the file named OUTPUT to transmit messages.

B. The number of evaluations of the right hand side can be found in the WORK array in the location determined by:
LENW - (N + 50) + 4

V. REMARKS .....

For other information, see Section IV of the writeup for SDRIV3.

```
***REFERENCES  C. W. Gear, Numerical Initial Value Problems in
                 Ordinary Differential Equations, Prentice-Hall, 1971.
***ROUTINES CALLED  SDRIV3, XERMSG
***REVISION HISTORY  (YYMMDD)
   790601  DATE WRITTEN
   900329  Initial submission to SLATEC.
   END PROLOGUE
```

SDRIV2

```
      SUBROUTINE SDRIV2 (N, T, Y, F, TOUT, MSTATE, NROOT, EPS, EWT,
     8   MINT, WORK, LENW, IWORK, LENIW, G, IERFLG)
***BEGIN PROLOGUE  SDRIV2
***PURPOSE  The function of SDRIV2 is to solve N ordinary differential
            equations of the form dY(I)/dT = F(Y(I),T), given the
            initial conditions Y(I) = YI.  The program has options to
            allow the solution of both stiff and non-stiff differential
            equations.  SDRIV2 uses single precision arithmetic.
***LIBRARY   SLATEC (SDRIVE)
***CATEGORY  I1A2, I1A1B
***TYPE      SINGLE PRECISION (SDRIV2-S, DDRIV2-D, CDRIV2-C)
***KEYWORDS  GEAR'S METHOD, INITIAL VALUE PROBLEMS, ODE,
             ORDINARY DIFFERENTIAL EQUATIONS, SDRIVE, SINGLE PRECISION,
             STIFF
***AUTHOR  Kahaner, D. K., (NIST)
             National Institute of Standards and Technology
             Gaithersburg, MD  20899
           Sutherland, C. D., (LANL)
             Mail Stop D466
             Los Alamos National Laboratory
             Los Alamos, NM  87545
***DESCRIPTION

  I.  PARAMETERS  ...................................................

    The user should use parameter names in the call sequence of SDRIV2
    for those quantities whose value may be altered by SDRIV2.  The
    parameters in the call sequence are:

    N      = (Input) The number of differential equations.

    T      = The independent variable.  On input for the first call, T
             is the initial point.  On output, T is the point at which
             the solution is given.

    Y      = The vector of dependent variables.  Y is used as input on
             the first call, to set the initial values.  On output, Y
             is the computed solution vector.  This array Y is passed
             in the call sequence of the user-provided routines F and
             G.  Thus parameters required by F and G can be stored in
             this array in components N+1 and above.  (Note: Changes
             by the user to the first N components of this array will
             take effect only after a restart, i.e., after setting
             MSTATE to +1(-1).)

    F      = A subroutine supplied by the user.  The name must be
             declared EXTERNAL in the user's calling program.  This
             subroutine is of the form:
                   SUBROUTINE F (N, T, Y, YDOT)
                   REAL Y(*), YDOT(*)
                     .
                     .
                   YDOT(1) = ...
                     .
                     .
                   YDOT(N) = ...
```

```
      END (Sample)
```

This computes YDOT = F(Y,T), the right hand side of the differential equations. Here Y is a vector of length at least N. The actual length of Y is determined by the user's declaration in the program which calls SDRIV2. Thus the dimensioning of Y in F, while required by FORTRAN convention, does not actually allocate any storage. When this subroutine is called, the first N components of Y are intermediate approximations to the solution components. The user should not alter these values. Here YDOT is a vector of length N. The user should only compute YDOT(I) for I from 1 to N. Normally a return from F passes control back to SDRIV2. However, if the user would like to abort the calculation, i.e., return control to the program which calls SDRIV2, he should set N to zero. SDRIV2 will signal this by returning a value of MSTATE equal to +6(-6). Altering the value of N in F has no effect on the value of N in the call sequence of SDRIV2.

TOUT = (Input) The point at which the solution is desired.

MSTATE = An integer describing the status of integration. The user must initialize MSTATE to +1 or -1. If MSTATE is positive, the routine will integrate past TOUT and interpolate the solution. This is the most efficient mode. If MSTATE is negative, the routine will adjust its internal step to reach TOUT exactly (useful if a singularity exists beyond TOUT.) The meaning of the magnitude of MSTATE:

1. (Input) Means the first call to the routine. This value must be set by the user. On all subsequent calls the value of MSTATE should be tested by the user. Unless SDRIV2 is to be reinitialized, only the sign of MSTATE may be changed by the user. (As a convenience to the user who may wish to put out the initial conditions, SDRIV2 can be called with MSTATE=+1(-1), and TOUT=T. In this case the program will return with MSTATE unchanged, i.e., MSTATE=+1(-1).)
2. (Output) Means a successful integration. If a normal continuation is desired (i.e., a further integration in the same direction), simply advance TOUT and call again. All other parameters are automatically set.
3. (Output)(Unsuccessful) Means the integrator has taken 1000 steps without reaching TOUT. The user can continue the integration by simply calling SDRIV2 again. Other than an error in problem setup, the most likely cause for this condition is trying to integrate a stiff set of equations with the non-stiff integrator option. (See description of MINT below.)
4. (Output)(Unsuccessful) Means too much accuracy has been requested. EPS has been increased to a value the program estimates is appropriate. The user can continue the integration by simply calling SDRIV2 again.
5. (Output) A root was found at a point less than TOUT. The user can continue the integration toward TOUT by simply calling SDRIV2 again.
6. (Output)(Unsuccessful) N has been set to zero in SUBROUTINE F.

7 (Output)(Unsuccessful) N has been set to zero in FUNCTION G. See description of G below.
8 (Output)(Successful) For MSTATE negative, T is beyond TOUT. The solution was obtained by interpolation. The user can continue the integration by simply advancing TOUT and calling SDRIV2 again.
9 (Output)(Unsuccessful) The solution could not be obtained. The value of IERFLG (see description below) for a "Recoverable" situation indicates the type of difficulty encountered: either an illegal value for a parameter or an inability to continue the solution. For this condition the user should take corrective action and reset MSTATE to +1(-1) before calling SDRIV2 again. Otherwise the program will terminate the run.

NROOT = (Input) The number of equations whose roots are desired. If NROOT is zero, the root search is not active. This option is useful for obtaining output at points which are not known in advance, but depend upon the solution, e.g., when some solution component takes on a specified value. The root search is carried out using the user-written function G (see description of G below.) SDRIV2 attempts to find the value of T at which one of the equations changes sign. SDRIV2 can find at most one root per equation per internal integration step, and will then return the solution either at TOUT or at a root, whichever occurs first in the direction of integration. The initial point is never reported as a root. The index of the equation whose root is being reported is stored in the sixth element of IWORK.
NOTE: NROOT is never altered by this program.

EPS = On input, the requested relative accuracy in all solution components. EPS = 0 is allowed. On output, the adjusted relative accuracy if the input value was too small. The value of EPS should be set as large as is reasonable, because the amount of work done by SDRIV2 increases as EPS decreases.

EWT = (Input) Problem zero, i.e., the smallest physically meaningful value for the solution. This is used internally to compute an array YWT(I) = MAX(ABS(Y(I)), EWT). One step error estimates divided by YWT(I) are kept less than EPS. Setting EWT to zero provides pure relative error control. However, setting EWT smaller than necessary can adversely affect the running time.

MINT = (Input) The integration method flag.
MINT = 1 Means the Adams methods, and is used for non-stiff problems.
MINT = 2 Means the stiff methods of Gear (i.e., the backward differentiation formulas), and is used for stiff problems.
MINT = 3 Means the program dynamically selects the Adams methods when the problem is non-stiff and the Gear methods when the problem is stiff.
MINT may not be changed without restarting, i.e., setting the magnitude of MSTATE to 1.

```
WORK
LENW   = (Input)
         WORK is an array of LENW real words used
         internally for temporary storage.  The user must allocate
         space for this array in the calling program by a statement
         such as
                   REAL WORK(...)
         The length of WORK should be at least
           16*N + 2*NROOT + 250         if MINT is 1, or
           N*N + 10*N + 2*NROOT + 250   if MINT is 2, or
           N*N + 17*N + 2*NROOT + 250   if MINT is 3,
         and LENW should be set to the value used.  The contents of
         WORK should not be disturbed between calls to SDRIV2.

IWORK
LENIW  = (Input)
         IWORK is an integer array of length LENIW used internally
         for temporary storage.  The user must allocate space for
         this array in the calling program by a statement such as
                   INTEGER IWORK(...)
         The length of IWORK should be at least
           50      if MINT is 1, or
           N+50    if MINT is 2 or 3,
         and LENIW should be set to the value used.  The contents
         of IWORK should not be disturbed between calls to SDRIV2.

G      = A real FORTRAN function supplied by the user
         if NROOT is not 0.  In this case, the name must be
         declared EXTERNAL in the user's calling program.  G is
         repeatedly called with different values of IROOT to
         obtain the value of each of the NROOT equations for which
         a root is desired.  G is of the form:
               REAL FUNCTION G (N, T, Y, IROOT)
               REAL Y(*)
               GO TO (10, ...), IROOT
          10   G = ...
                  .
                  .
               END (Sample)
         Here, Y is a vector of length at least N, whose first N
         components are the solution components at the point T.
         The user should not alter these values.  The actual length
         of Y is determined by the user's declaration in the
         program which calls SDRIV2.  Thus the dimensioning of Y in
         G, while required by FORTRAN convention, does not actually
         allocate any storage.  Normally a return from G passes
         control back to  SDRIV2.  However, if the user would like
         to abort the calculation, i.e., return control to the
         program which calls SDRIV2, he should set N to zero.
         SDRIV2 will signal this by returning a value of MSTATE
         equal to +7(-7).  In this case, the index of the equation
         being evaluated is stored in the sixth element of IWORK.
         Altering the value of N in G has no effect on the value of
         N in the call sequence of SDRIV2.

IERFLG = An error flag.  The error number associated with a
         diagnostic message (see Section II-A below) is the same as
         the corresponding value of IERFLG.  The meaning of IERFLG:
           0  The routine completed successfully. (No message is
```

issued.)

3 (Warning) The number of steps required to reach TOUT exceeds MXSTEP.

4 (Warning) The value of EPS is too small.

11 (Warning) For MSTATE negative, T is beyond TOUT. The solution was obtained by interpolation.

15 (Warning) The integration step size is below the roundoff level of T. (The program issues this message as a warning but does not return control to the user.)

22 (Recoverable) N is not positive.

23 (Recoverable) MINT is less than 1 or greater than 3 .

26 (Recoverable) The magnitude of MSTATE is either 0 or greater than 9 .

27 (Recoverable) EPS is less than zero.

32 (Recoverable) Insufficient storage has been allocated for the WORK array.

33 (Recoverable) Insufficient storage has been allocated for the IWORK array.

41 (Recoverable) The integration step size has gone to zero.

42 (Recoverable) The integration step size has been reduced about 50 times without advancing the solution. The problem setup may not be correct.

999 (Fatal) The magnitude of MSTATE is 9 .

II. OTHER COMMUNICATION TO THE USER .....

A. The solver communicates to the user through the parameters above. In addition it writes diagnostic messages through the standard error handling program XERMSG. A complete description of XERMSG is given in "Guide to the SLATEC Common Mathematical Library" by Kirby W. Fong et al.. At installations which do not have this error handling package the short but serviceable routine, XERMSG, available with this package, can be used. That program uses the file named OUTPUT to transmit messages.

B. The first three elements of WORK and the first five elements of IWORK will contain the following statistical data:

AVGH	The average step size used.
HUSED	The step size last used (successfully).
AVGORD	The average order used.
IMXERR	The index of the element of the solution vector that contributed most to the last error test.
NQUSED	The order last used (successfully).
NSTEP	The number of steps taken since last initialization.
NFE	The number of evaluations of the right hand side.
NJE	The number of evaluations of the Jacobian matrix.

III. REMARKS .....

A. On any return from SDRIV2 all information necessary to continue the calculation is contained in the call sequence parameters, including the work arrays. Thus it is possible to suspend one problem, integrate another, and then return to the first.

B. If this package is to be used in an overlay situation, the user must declare in the primary overlay the variables in the call sequence to SDRIV2.

C. When the routine G is not required, difficulties associated with an unsatisfied external can be avoided by using the name of the routine which calculates the right hand side of the differential equations in place of G in the call sequence of SDRIV2.

IV. USAGE .....

```
      PROGRAM SAMPLE
      EXTERNAL F
      PARAMETER(MINT = 1, NROOT = 0, N = ...,
     8          LENW = 16*N + 2*NROOT + 250, LENIW = 50)
C                                   N is the number of equations
      REAL EPS, EWT, T, TOUT, WORK(LENW), Y(N)
      INTEGER IWORK(LENIW)
      OPEN(FILE='TAPE6', UNIT=6, STATUS='NEW')
C                                                  Initial point
      T = 0.
C                                       Set initial conditions
      DO 10 I = 1,N
 10     Y(I) = ...
      TOUT = T
      EWT = ...
      MSTATE = 1
      EPS = ...
 20   CALL SDRIV2 (N, T, Y, F, TOUT, MSTATE, NROOT, EPS, EWT,
     8             MINT, WORK, LENW, IWORK, LENIW, F, IERFLG)
C                                   Next to last argument is not
C                                      F if rootfinding is used.
      IF (MSTATE .GT. 2) STOP
      WRITE(6, 100) TOUT, (Y(I), I=1,N)
      TOUT = TOUT + 1.
      IF (TOUT .LE. 10.) GO TO 20
 100  FORMAT(...)
      END (Sample)
```

***REFERENCES C. W. Gear, Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall, 1971.
***ROUTINES CALLED SDRIV3, XERMSG
***REVISION HISTORY (YYMMDD)
790601 DATE WRITTEN
900329 Initial submission to SLATEC.
END PROLOGUE

SDRIV3

```
      SUBROUTINE SDRIV3 (N, T, Y, F, NSTATE, TOUT, NTASK, NROOT,
     8     EPS, EWT, IERROR, MINT, MITER, IMPL, ML, MU, MXORD, HMAX,
     8   LENW, IWORK, LENIW, JACOBN, FA, NDE, MXSTEP, G, USERS, IERFLG)
***BEGIN PROLOGUE  SDRIV3
***PURPOSE  The function of SDRIV3 is to solve N ordinary differential
            equations of the form dY(I)/dT = F(Y(I),T), given the
            initial conditions Y(I) = YI.  The program has options to
            allow the solution of both stiff and non-stiff differential
            equations.  Other important options are available.  SDRIV3
            uses single precision arithmetic.
***LIBRARY   SLATEC (SDRIVE)
***CATEGORY  I1A2, I1A1B
***TYPE      SINGLE PRECISION (SDRIV3-S, DDRIV3-D, CDRIV3-C)
***KEYWORDS  GEAR'S METHOD, INITIAL VALUE PROBLEMS, ODE,
             ORDINARY DIFFERENTIAL EQUATIONS, SDRIVE, SINGLE PRECISION,
             STIFF
***AUTHOR  Kahaner, D. K., (NIST)
             National Institute of Standards and Technology
             Gaithersburg, MD  20899
           Sutherland, C. D., (LANL)
             Mail Stop D466
             Los Alamos National Laboratory
             Los Alamos, NM  87545
***DESCRIPTION

  I.  ABSTRACT  .......................................................

    The primary function of SDRIV3 is to solve N ordinary differential
    equations of the form dY(I)/dT = F(Y(I),T), given the initial
    conditions Y(I) = YI.  The program has options to allow the
    solution of both stiff and non-stiff differential equations.  In
    addition, SDRIV3 may be used to solve:
      1. The initial value problem, A*dY(I)/dT = F(Y(I),T), where A is
         a non-singular matrix depending on Y and T.
      2. The hybrid differential/algebraic initial value problem,
         A*dY(I)/dT = F(Y(I),T), where A is a vector (whose values may
         depend upon Y and T) some of whose components will be zero
         corresponding to those equations which are algebraic rather
         than differential.
    SDRIV3 is to be called once for each output point of T.

  II.  PARAMETERS  ....................................................

    The user should use parameter names in the call sequence of SDRIV3
    for those quantities whose value may be altered by SDRIV3.  The
    parameters in the call sequence are:

    N      = (Input) The number of dependent functions whose solution
             is desired.  N must not be altered during a problem.

    T      = The independent variable.  On input for the first call, T
             is the initial point.  On output, T is the point at which
             the solution is given.

    Y      = The vector of dependent variables.  Y is used as input on
             the first call, to set the initial values.  On output, Y
```

is the computed solution vector. This array Y is passed in the call sequence of the user-provided routines F, JACOBN, FA, USERS, and G. Thus parameters required by those routines can be stored in this array in components N+1 and above. (Note: Changes by the user to the first N components of this array will take effect only after a restart, i.e., after setting NSTATE to 1 .)

F = A subroutine supplied by the user. The name must be declared EXTERNAL in the user's calling program. This subroutine is of the form:

```
      SUBROUTINE F (N, T, Y, YDOT)
      REAL Y(*), YDOT(*)
        .
        .
      YDOT(1) = ...
        .
        .
      YDOT(N) = ...
      END (Sample)
```

This computes YDOT = F(Y,T), the right hand side of the differential equations. Here Y is a vector of length at least N. The actual length of Y is determined by the user's declaration in the program which calls SDRIV3. Thus the dimensioning of Y in F, while required by FORTRAN convention, does not actually allocate any storage. When this subroutine is called, the first N components of Y are intermediate approximations to the solution components. The user should not alter these values. Here YDOT is a vector of length N. The user should only compute YDOT(I) for I from 1 to N. Normally a return from F passes control back to SDRIV3. However, if the user would like to abort the calculation, i.e., return control to the program which calls SDRIV3, he should set N to zero. SDRIV3 will signal this by returning a value of NSTATE equal to 6 . Altering the value of N in F has no effect on the value of N in the call sequence of SDRIV3.

NSTATE = An integer describing the status of integration. The meaning of NSTATE is as follows:

1 (Input) Means the first call to the routine. This value must be set by the user. On all subsequent calls the value of NSTATE should be tested by the user, but must not be altered. (As a convenience to the user who may wish to put out the initial conditions, SDRIV3 can be called with NSTATE=1, and TOUT=T. In this case the program will return with NSTATE unchanged, i.e., NSTATE=1.)

2 (Output) Means a successful integration. If a normal continuation is desired (i.e., a further integration in the same direction), simply advance TOUT and call again. All other parameters are automatically set.

3 (Output)(Unsuccessful) Means the integrator has taken MXSTEP steps without reaching TOUT. The user can continue the integration by simply calling SDRIV3 again.

4 (Output)(Unsuccessful) Means too much accuracy has been requested. EPS has been increased to a value the program estimates is appropriate. The user can continue the integration by simply calling SDRIV3

again.

5 (Output) A root was found at a point less than TOUT. The user can continue the integration toward TOUT by simply calling SDRIV3 again.

6 (Output)(Unsuccessful) N has been set to zero in SUBROUTINE F.

7 (Output)(Unsuccessful) N has been set to zero in FUNCTION G. See description of G below.

8 (Output)(Unsuccessful) N has been set to zero in SUBROUTINE JACOBN. See description of JACOBN below.

9 (Output)(Unsuccessful) N has been set to zero in SUBROUTINE FA. See description of FA below.

10 (Output)(Unsuccessful) N has been set to zero in SUBROUTINE USERS. See description of USERS below.

11 (Output)(Successful) For NTASK = 2 or 3, T is beyond TOUT. The solution was obtained by interpolation. The user can continue the integration by simply advancing TOUT and calling SDRIV3 again.

12 (Output)(Unsuccessful) The solution could not be obtained. The value of IERFLG (see description below) for a "Recoverable" situation indicates the type of difficulty encountered: either an illegal value for a parameter or an inability to continue the solution. For this condition the user should take corrective action and reset NSTATE to 1 before calling SDRIV3 again. Otherwise the program will terminate the run.

TOUT = (Input) The point at which the solution is desired. The position of TOUT relative to T on the first call determines the direction of integration.

NTASK = (Input) An index specifying the manner of returning the solution, according to the following:

NTASK = 1 Means SDRIV3 will integrate past TOUT and interpolate the solution. This is the most efficient mode.

NTASK = 2 Means SDRIV3 will return the solution after each internal integration step, or at TOUT, whichever comes first. In the latter case, the program integrates exactly to TOUT.

NTASK = 3 Means SDRIV3 will adjust its internal step to reach TOUT exactly (useful if a singularity exists beyond TOUT.)

NROOT = (Input) The number of equations whose roots are desired. If NROOT is zero, the root search is not active. This option is useful for obtaining output at points which are not known in advance, but depend upon the solution, e.g., when some solution component takes on a specified value. The root search is carried out using the user-written function G (see description of G below.) SDRIV3 attempts to find the value of T at which one of the equations changes sign. SDRIV3 can find at most one root per equation per internal integration step, and will then return the solution either at TOUT or at a root, whichever occurs first in the direction of integration. The initial point is never reported as a root. The index of the equation whose root is being reported is stored in the sixth element of IWORK.

```
         NOTE: NROOT is never altered by this program.

EPS    = On input, the requested relative accuracy in all solution
         components.  EPS = 0 is allowed.  On output, the adjusted
         relative accuracy if the input value was too small.  The
         value of EPS should be set as large as is reasonable,
         because the amount of work done by SDRIV3 increases as EPS
         decreases.

EWT    = (Input) Problem zero, i.e., the smallest, nonzero,
         physically meaningful value for the solution.  (Array,
         possibly of length one.  See following description of
         IERROR.)  Setting EWT smaller than necessary can adversely
         affect the running time.

IERROR = (Input) Error control indicator.  A value of 3 is
         suggested for most problems.  Other choices and detailed
         explanations of EWT and IERROR are given below for those
         who may need extra flexibility.

         These last three input quantities EPS, EWT and IERROR
         control the accuracy of the computed solution.  EWT and
         IERROR are used internally to compute an array YWT.  One
         step error estimates divided by YWT(I) are kept less than
         EPS in root mean square norm.
             IERROR (Set by the user) =
             1  Means YWT(I) = 1. (Absolute error control)
                                  EWT is ignored.
             2  Means YWT(I) = ABS(Y(I)),  (Relative error control)
                                  EWT is ignored.
             3  Means YWT(I) = MAX(ABS(Y(I)), EWT(1)).
             4  Means YWT(I) = MAX(ABS(Y(I)), EWT(I)).
                This choice is useful when the solution components
                have differing scales.
             5  Means YWT(I) = EWT(I).
         If IERROR is 3, EWT need only be dimensioned one.
         If IERROR is 4 or 5, the user must dimension EWT at least
         N, and set its values.

MINT   = (Input) The integration method indicator.
           MINT = 1  Means the Adams methods, and is used for
                     non-stiff problems.
           MINT = 2  Means the stiff methods of Gear (i.e., the
                     backward differentiation formulas), and is
                     used for stiff problems.
           MINT = 3  Means the program dynamically selects the
                     Adams methods when the problem is non-stiff
                     and the Gear methods when the problem is
                     stiff.  When using the Adams methods, the
                     program uses a value of MITER=0; when using
                     the Gear methods, the program uses the value
                     of MITER provided by the user.  Only a value
                     of IMPL = 0 and a value of MITER = 1, 2, 4, or
                     5 is allowed for this option.  The user may
                     not alter the value of MINT or MITER without
                     restarting, i.e., setting NSTATE to 1.

MITER  = (Input) The iteration method indicator.
           MITER = 0  Means functional iteration.  This value is
                      suggested for non-stiff problems.
```

```
            MITER = 1  Means chord method with analytic Jacobian.
                       In this case, the user supplies subroutine
                       JACOBN (see description below).
            MITER = 2  Means chord method with Jacobian calculated
                       internally by finite differences.
            MITER = 3  Means chord method with corrections computed
                       by the user-written routine USERS (see
                       description of USERS below.)  This option
                       allows all matrix algebra and storage
                       decisions to be made by the user.  When using
                       a value of MITER = 3, the subroutine FA is
                       not required, even if IMPL is not 0.  For
                       further information on using this option, see
                       Section IV-E below.
            MITER = 4  Means the same as MITER = 1 but the A and
                       Jacobian matrices are assumed to be banded.
            MITER = 5  Means the same as MITER = 2 but the A and
                       Jacobian matrices are assumed to be banded.

IMPL   = (Input) The implicit method indicator.
            IMPL = 0    Means solving dY(I)/dT = F(Y(I),T).
            IMPL = 1    Means solving A*dY(I)/dT = F(Y(I),T), non-
                        singular A (see description of FA below.)
                        Only MINT = 1 or 2, and MITER = 1, 2, 3, 4,
                        or 5 are allowed for this option.
            IMPL = 2,3  Means solving certain systems of hybrid
                        differential/algebraic equations (see
                        description of FA below.)  Only MINT = 2 and
                        MITER = 1, 2, 3, 4, or 5, are allowed for
                        this option.
            The value of IMPL must not be changed during a problem.

ML     = (Input) The lower half-bandwidth in the case of a banded
         A or Jacobian matrix.  (I.e., maximum(R-C) for nonzero
         A(R,C).)

MU     = (Input) The upper half-bandwidth in the case of a banded
         A or Jacobian matrix.  (I.e., maximum(C-R).)

MXORD  = (Input) The maximum order desired. This is .LE. 12 for
         the Adams methods and .LE. 5 for the Gear methods.  Normal
         value is 12 and 5, respectively.  If MINT is 3, the
         maximum order used will be MIN(MXORD, 12) when using the
         Adams methods, and MIN(MXORD, 5) when using the Gear
         methods.  MXORD must not be altered during a problem.

HMAX   = (Input) The maximum magnitude of the step size that will
         be used for the problem.  This is useful for ensuring that
         important details are not missed.  If this is not the
         case, a large value, such as the interval length, is
         suggested.

WORK
LENW   = (Input)
         WORK is an array of LENW real words used
         internally for temporary storage.  The user must allocate
         space for this array in the calling program by a statement
         such as
                   REAL WORK(...)
         The following table gives the required minimum value for
```

the length of WORK, depending on the value of IMPL and MITER. LENW should be set to the value used. The contents of WORK should not be disturbed between calls to SDRIV3.

IMPL =	0	1	2	3
MITER = 0	(MXORD+4)*N + 2*NROOT + 250	Not allowed	Not allowed	Not allowed
1,2	N*N + (MXORD+5)*N + 2*NROOT + 250	2*N*N + (MXORD+5)*N + 2*NROOT + 250	N*N + (MXORD+6)*N + 2*NROOT + 250	N*(N + NDE) + (MXORD+5)*N + 2*NROOT + 250
3	(MXORD+4)*N + 2*NROOT + 250	(MXORD+4)*N + 2*NROOT + 250	(MXORD+4)*N + 2*NROOT + 250	(MXORD+4)*N + 2*NROOT + 250
4,5	(2*ML+MU+1) *N + (MXORD+5)*N + 2*NROOT + 250	2*(2*ML+MU+1) *N + (MXORD+5)*N + 2*NROOT + 250	(2*ML+MU+1) *N + (MXORD+6)*N + 2*NROOT + 250	(2*ML+MU+1)* (N+NDE) + + (MXORD+5)*N + 2*NROOT + 250

IWORK
LENIW = (Input)
IWORK is an integer array of length LENIW used internally for temporary storage. The user must allocate space for this array in the calling program by a statement such as

```
INTEGER IWORK(...)
```

The length of IWORK should be at least

```
  50      if MITER is 0 or 3, or
  N+50    if MITER is 1, 2, 4, or 5, or MINT is 3,
```

and LENIW should be set to the value used. The contents of IWORK should not be disturbed between calls to SDRIV3.

JACOBN = A subroutine supplied by the user, if MITER is 1 or 4. If this is the case, the name must be declared EXTERNAL in the user's calling program. Given a system of N differential equations, it is meaningful to speak about the partial derivative of the I-th right hand side with respect to the J-th dependent variable. In general there are N*N such quantities. Often however the equations can be ordered so that the I-th differential equation only involves dependent variables with index near I, e.g., I+1, I-2. Such a system is called banded. If, for all I, the I-th equation depends on at most the variables

```
  Y(I-ML), Y(I-ML+1), ... , Y(I), Y(I+1), ... , Y(I+MU)
```

then we call ML+MU+1 the bandwidth of the system. In a banded system many of the partial derivatives above are automatically zero. For the cases MITER = 1, 2, 4, and 5, some of these partials are needed. For the cases MITER = 2 and 5 the necessary derivatives are approximated numerically by SDRIV3, and we only ask the user to tell SDRIV3 the value of ML and MU if the system is banded. For the cases MITER = 1 and 4 the user must derive these partials algebraically and encode them in

subroutine JACOBN. By computing these derivatives the user can often save 20-30 per cent of the computing time. Usually, however, the accuracy is not much affected and most users will probably forego this option. The optional user-written subroutine JACOBN has the form:

```
      SUBROUTINE JACOBN (N, T, Y, DFDY, MATDIM, ML, MU)
      REAL Y(*), DFDY(MATDIM,*)
        .
        .
        Calculate values of DFDY
        .
        .
      END (Sample)
```

Here Y is a vector of length at least N. The actual length of Y is determined by the user's declaration in the program which calls SDRIV3. Thus the dimensioning of Y in JACOBN, while required by FORTRAN convention, does not actually allocate any storage. When this subroutine is called, the first N components of Y are intermediate approximations to the solution components. The user should not alter these values. If the system is not banded (MITER=1), the partials of the I-th equation with respect to the J-th dependent function are to be stored in DFDY(I,J). Thus partials of the I-th equation are stored in the I-th row of DFDY. If the system is banded (MITER=4), then the partials of the I-th equation with respect to Y(J) are to be stored in DFDY(K,J), where K=I-J+MU+1 . Normally a return from JACOBN passes control back to SDRIV3. However, if the user would like to abort the calculation, i.e., return control to the program which calls SDRIV3, he should set N to zero. SDRIV3 will signal this by returning a value of NSTATE equal to +8(-8). Altering the value of N in JACOBN has no effect on the value of N in the call sequence of SDRIV3.

FA = A subroutine supplied by the user if IMPL is not zero, and MITER is not 3. If so, the name must be declared EXTERNAL in the user's calling program. This subroutine computes the array A, where A*dY(I)/dT = F(Y(I),T). There are three cases:

IMPL=1.
Subroutine FA is of the form:

```
      SUBROUTINE FA (N, T, Y, A, MATDIM, ML, MU, NDE)
      REAL Y(*), A(MATDIM,*)
        .
        .
        Calculate ALL values of A
        .
        .
      END (Sample)
```

In this case A is assumed to be a nonsingular matrix, with the same structure as DFDY (see JACOBN description above). Programming considerations prevent complete generality. If MITER is 1 or 2, A is assumed to be full and the user must compute and store all values of A(I,J), I,J=1, ... ,N. If MITER is 4 or 5, A is assumed to be banded with lower and upper half bandwidth ML and MU. The left hand side of the I-th equation is a linear combination of dY(I-ML)/dT, dY(I-ML+1)/dT, ... ,

dY(I)/dT, ... , dY(I+MU-1)/dT, dY(I+MU)/dT. Thus in the I-th equation, the coefficient of dY(J)/dT is to be stored in A(K,J), where K=I-J+MU+1.
NOTE: The array A will be altered between calls to FA.

IMPL=2.
Subroutine FA is of the form:

```
    SUBROUTINE FA (N, T, Y, A, MATDIM, ML, MU, NDE)
    REAL Y(*), A(*)
      .
      .
      Calculate non-zero values of A(1),...,A(NDE)
      .
      .
    END (Sample)
```

In this case it is assumed that the system is ordered by the user so that the differential equations appear first, and the algebraic equations appear last. The algebraic equations must be written in the form: 0 = F(Y(I),T). When using this option it is up to the user to provide initial values for the Y(I) that satisfy the algebraic equations as well as possible. It is further assumed that A is a vector of length NDE. All of the components of A, which may depend on T, Y(I), etc., must be set by the user to non-zero values.

IMPL=3.
Subroutine FA is of the form:

```
    SUBROUTINE FA (N, T, Y, A, MATDIM, ML, MU, NDE)
    REAL Y(*), A(MATDIM,*)
      .
      .
      Calculate ALL values of A
      .
      .
    END (Sample)
```

In this case A is assumed to be a nonsingular NDE by NDE matrix with the same structure as DFDY (see JACOBN description above). Programming considerations prevent complete generality. If MITER is 1 or 2, A is assumed to be full and the user must compute and store all values of A(I,J), I,J=1, ... ,NDE. If MITER is 4 or 5, A is assumed to be banded with lower and upper half bandwidths ML and MU. The left hand side of the I-th equation is a linear combination of dY(I-ML)/dT, dY(I-ML+1)/dT, ... , dY(I)/dT, ... , dY(I+MU-1)/dT, dY(I+MU)/dT. Thus in the I-th equation, the coefficient of dY(J)/dT is to be stored in A(K,J), where K=I-J+MU+1. It is assumed that the system is ordered by the user so that the differential equations appear first, and the algebraic equations appear last. The algebraic equations must be written in the form 0 = F(Y(I),T). When using this option it is up to the user to provide initial values for the Y(I) that satisfy the algebraic equations as well as possible.
NOTE: For IMPL = 3, the array A will be altered between calls to FA.

Here Y is a vector of length at least N. The actual length of Y is determined by the user's declaration in the program which calls SDRIV3. Thus the dimensioning of Y in

FA, while required by FORTRAN convention, does not actually allocate any storage. When this subroutine is called, the first N components of Y are intermediate approximations to the solution components. The user should not alter these values. FA is always called immediately after calling F, with the same values of T and Y. Normally a return from FA passes control back to SDRIV3. However, if the user would like to abort the calculation, i.e., return control to the program which calls SDRIV3, he should set N to zero. SDRIV3 will signal this by returning a value of NSTATE equal to +9(-9). Altering the value of N in FA has no effect on the value of N in the call sequence of SDRIV3.

NDE = (Input) The number of differential equations. This is required only for IMPL = 2 or 3, with NDE .LT. N.

MXSTEP = (Input) The maximum number of internal steps allowed on one call to SDRIV3.

G = A real FORTRAN function supplied by the user if NROOT is not 0. In this case, the name must be declared EXTERNAL in the user's calling program. G is repeatedly called with different values of IROOT to obtain the value of each of the NROOT equations for which a root is desired. G is of the form:

```
      REAL FUNCTION G (N, T, Y, IROOT)
      REAL Y(*)
      GO TO (10, ...), IROOT
 10   G = ...
        .
        .
      END (Sample)
```

Here, Y is a vector of length at least N, whose first N components are the solution components at the point T. The user should not alter these values. The actual length of Y is determined by the user's declaration in the program which calls SDRIV3. Thus the dimensioning of Y in G, while required by FORTRAN convention, does not actually allocate any storage. Normally a return from G passes control back to SDRIV3. However, if the user would like to abort the calculation, i.e., return control to the program which calls SDRIV3, he should set N to zero. SDRIV3 will signal this by returning a value of NSTATE equal to +7(-7). In this case, the index of the equation being evaluated is stored in the sixth element of IWORK. Altering the value of N in G has no effect on the value of N in the call sequence of SDRIV3.

USERS = A subroutine supplied by the user, if MITER is 3. If this is the case, the name must be declared EXTERNAL in the user's calling program. The routine USERS is called by SDRIV3 when certain linear systems must be solved. The user may choose any method to form, store and solve these systems in order to obtain the solution result that is returned to SDRIV3. In particular, this allows sparse matrix methods to be used. The call sequence for this routine is:

```
      SUBROUTINE USERS (Y, YH, YWT, SAVE1, SAVE2, T, H, EL,
```

```
  8                  IMPL, N, NDE, IFLAG)
   REAL Y(*), YH(*), YWT(*), SAVE1(*),
  8     SAVE2(*), T, H, EL
```

The input variable IFLAG indicates what action is to be taken. Subroutine USERS should perform the following operations, depending on the value of IFLAG and IMPL.

```
  IFLAG = 0
    IMPL = 0.  USERS is not called.
    IMPL = 1, 2 or 3.  Solve the system A*X = SAVE2,
      returning the result in SAVE2.  The array SAVE1 can
      be used as a work array.  For IMPL = 1, there are N
      components to the system, and for IMPL = 2 or 3,
      there are NDE components to the system.

  IFLAG = 1
    IMPL = 0.  Compute, decompose and store the matrix
      (I - H*EL*J), where I is the identity matrix and J
      is the Jacobian matrix of the right hand side.  The
      array SAVE1 can be used as a work array.
    IMPL = 1, 2 or 3. Compute, decompose and store the
      matrix (A - H*EL*J).  The array SAVE1 can be used as
      a work array.

  IFLAG = 2
    IMPL = 0.   Solve the system
        (I - H*EL*J)*X = H*SAVE2 - YH - SAVE1,
      returning the result in SAVE2.
    IMPL = 1, 2 or 3.  Solve the system
      (A - H*EL*J)*X = H*SAVE2 - A*(YH + SAVE1)
      returning the result in SAVE2.
    The array SAVE1 should not be altered.
```

If IFLAG is 0 and IMPL is 1 or 2 and the matrix A is singular, or if IFLAG is 1 and one of the matrices (I - H*EL*J), (A - H*EL*J) is singular, the INTEGER variable IFLAG is to be set to -1 before RETURNing. Normally a return from USERS passes control back to SDRIV3. However, if the user would like to abort the calculation, i.e., return control to the program which calls SDRIV3, he should set N to zero. SDRIV3 will signal this by returning a value of NSTATE equal to +10(-10). Altering the value of N in USERS has no effect on the value of N in the call sequence of SDRIV3.

IERFLG = An error flag. The error number associated with a diagnostic message (see Section III-A below) is the same as the corresponding value of IERFLG. The meaning of IERFLG:

```
  0  The routine completed successfully. (No message is
     issued.)
  3  (Warning) The number of steps required to reach TOUT
     exceeds MXSTEP.
  4  (Warning) The value of EPS is too small.
 11  (Warning) For NTASK = 2 or 3, T is beyond TOUT.
     The solution was obtained by interpolation.
 15  (Warning) The integration step size is below the
     roundoff level of T.  (The program issues this
     message as a warning but does not return control to
     the user.)
```

22 (Recoverable) N is not positive.
23 (Recoverable) MINT is less than 1 or greater than 3 .
24 (Recoverable) MITER is less than 0 or greater than 5 .
25 (Recoverable) IMPL is less than 0 or greater than 3 .
26 (Recoverable) The value of NSTATE is less than 1 or greater than 12 .
27 (Recoverable) EPS is less than zero.
28 (Recoverable) MXORD is not positive.
29 (Recoverable) For MINT = 3, either MITER = 0 or 3, or IMPL = 0 .
30 (Recoverable) For MITER = 0, IMPL is not 0 .
31 (Recoverable) For MINT = 1, IMPL is 2 or 3 .
32 (Recoverable) Insufficient storage has been allocated for the WORK array.
33 (Recoverable) Insufficient storage has been allocated for the IWORK array.
41 (Recoverable) The integration step size has gone to zero.
42 (Recoverable) The integration step size has been reduced about 50 times without advancing the solution. The problem setup may not be correct.
43 (Recoverable) For IMPL greater than 0, the matrix A is singular.
999 (Fatal) The value of NSTATE is 12 .

III. OTHER COMMUNICATION TO THE USER .....

A. The solver communicates to the user through the parameters above. In addition it writes diagnostic messages through the standard error handling program XERMSG. A complete description of XERMSG is given in "Guide to the SLATEC Common Mathematical Library" by Kirby W. Fong et al.. At installations which do not have this error handling package the short but serviceable routine, XERMSG, available with this package, can be used. That program uses the file named OUTPUT to transmit messages.

B. The first three elements of WORK and the first five elements of IWORK will contain the following statistical data:

AVGH	The average step size used.
HUSED	The step size last used (successfully).
AVGORD	The average order used.
IMXERR	The index of the element of the solution vector that contributed most to the last error test.
NQUSED	The order last used (successfully).
NSTEP	The number of steps taken since last initialization.
NFE	The number of evaluations of the right hand side.
NJE	The number of evaluations of the Jacobian matrix.

IV. REMARKS .....

A. Other routines used:
SDNTP, SDZRO, SDSTP, SDNTL, SDPST, SDCOR, SDCST, SDPSC, and SDSCL;
SGEFA, SGESL, SGBFA, SGBSL, and SNRM2 (from LINPACK)
R1MACH (from the Bell Laboratories Machine Constants Package)
XERMSG (from the SLATEC Common Math Library)
The last seven routines above, not having been written by the present authors, are not explicitly part of this package.

B. On any return from SDRIV3 all information necessary to continue the calculation is contained in the call sequence parameters, including the work arrays. Thus it is possible to suspend one problem, integrate another, and then return to the first.

C. If this package is to be used in an overlay situation, the user must declare in the primary overlay the variables in the call sequence to SDRIV3.

D. Changing parameters during an integration.
The value of NROOT, EPS, EWT, IERROR, MINT, MITER, or HMAX may be altered by the user between calls to SDRIV3. For example, if too much accuracy has been requested (the program returns with NSTATE = 4 and an increased value of EPS) the user may wish to increase EPS further. In general, prudence is necessary when making changes in parameters since such changes are not implemented until the next integration step, which is not necessarily the next call to SDRIV3. This can happen if the program has already integrated to a point which is beyond the new point TOUT.

E. As the price for complete control of matrix algebra, the SDRIV3 USERS option puts all responsibility for Jacobian matrix evaluation on the user. It is often useful to approximate numerically all or part of the Jacobian matrix. However this must be done carefully. The FORTRAN sequence below illustrates the method we recommend. It can be inserted directly into subroutine USERS to approximate Jacobian elements in rows I1 to I2 and columns J1 to J2.

```
      REAL DFDY(N,N), EPSJ, H, R, R1MACH,
     8     SAVE1(N), SAVE2(N), T, UROUND, Y(N), YJ, YWT(N)
      UROUND = R1MACH(4)
      EPSJ = SQRT(UROUND)
      DO 30 J = J1,J2
        R = EPSJ*MAX(ABS(YWT(J)), ABS(Y(J)))
        IF (R .EQ. 0.E0) R = YWT(J)
        YJ = Y(J)
        Y(J) = Y(J) + R
        CALL F (N, T, Y, SAVE1)
        IF (N .EQ. 0) RETURN
        Y(J) = YJ
        DO 20 I = I1,I2
  20      DFDY(I,J) = (SAVE1(I) - SAVE2(I))/R
  30    CONTINUE
```

Many problems give rise to structured sparse Jacobians, e.g., block banded. It is possible to approximate them with fewer function evaluations than the above procedure uses; see Curtis, Powell and Reid, J. Inst. Maths Applics, (1974), Vol. 13, pp. 117-119.

F. When any of the routines JACOBN, FA, G, or USERS, is not required, difficulties associated with unsatisfied externals can be avoided by using the name of the routine which calculates the right hand side of the differential equations in place of the corresponding name in the call sequence of SDRIV3.

***REFERENCES C. W. Gear, Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall, 1971.

***ROUTINES CALLED R1MACH, SDNTP, SDSTP, SDZRO, SGBFA, SGBSL, SGEFA, SGESL, SNRM2, XERMSG

```
***REVISION HISTORY  (YYMMDD)
   790601  DATE WRITTEN
   900329  Initial submission to SLATEC.
   END PROLOGUE
```

SDSDOT

```
      REAL FUNCTION SDSDOT (N, SB, SX, INCX, SY, INCY)
***BEGIN PROLOGUE  SDSDOT
***PURPOSE  Compute the inner product of two vectors with extended
            precision accumulation.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A4
***TYPE      SINGLE PRECISION (SDSDOT-S, CDCDOT-C)
***KEYWORDS  BLAS, DOT PRODUCT, INNER PRODUCT, LINEAR ALGEBRA, VECTOR
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
           Kincaid, D. R., (U. of Texas)
           Krogh, F. T., (JPL)
***DESCRIPTION

                 B L A S  Subprogram
    Description of Parameters

      --Input--
         N  number of elements in input vector(s)
        SB  single precision scalar to be added to inner product
        SX  single precision vector with N elements
      INCX  storage spacing between elements of SX
        SY  single precision vector with N elements
      INCY  storage spacing between elements of SY

      --Output--
   SDSDOT  single precision dot product (SB if N .LE. 0)

      Returns S.P. result with dot product accumulated in D.P.
      SDSDOT = SB + sum for I = 0 to N-1 of SX(LX+I*INCX)*SY(LY+I*INCY),
      where LX = 1 if INCX .GE. 0, else LX = 1+(1-N)*INCX, and LY is
      defined in a similar way using INCY.

***REFERENCES  C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T.
                 Krogh, Basic linear algebra subprograms for Fortran
                 usage, Algorithm No. 539, Transactions on Mathematical
                 Software 5, 3 (September 1979), pp. 308-323.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   791001  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920310  Corrected definition of LX in DESCRIPTION.  (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SEPELI

```
      SUBROUTINE SEPELI (INTL, IORDER, A, B, M, MBDCND, BDA, ALPHA, BDB,
         BETA, C, D, N, NBDCND, BDC, GAMA, BDD, XNU, COFX, COFY, GRHS,
     +   USOL, IDMN, W, PERTRB, IERROR)
***BEGIN PROLOGUE  SEPELI
***PURPOSE  Discretize and solve a second and, optionally, a fourth
            order finite difference approximation on a uniform grid to
            the general separable elliptic partial differential
            equation on a rectangle with any combination of periodic or
            mixed boundary conditions.
***LIBRARY   SLATEC (FISHPACK)
***CATEGORY  I2B1A2
***TYPE      SINGLE PRECISION (SEPELI-S)
***KEYWORDS  ELLIPTIC, FISHPACK, HELMHOLTZ, PDE, SEPARABLE
***AUTHOR  Adams, J., (NCAR)
           Swarztrauber, P. N., (NCAR)
           Sweet, R., (NCAR)
***DESCRIPTION

 Dimension of           BDA(N+1), BDB(N+1), BDC(M+1), BDD(M+1),
 Arguments              USOL(IDMN,N+1), GRHS(IDMN,N+1),
                        W (see argument list)

 Latest Revision        March 1977

 Purpose                SEPELI solves for either the second-order
                        finite difference approximation or a
                        fourth-order approximation to a separable
                        elliptic equation.

                                 2    2
                             AF(X)*d U/dX + BF(X)*dU/dX  + CF(X)*U +
                                 2    2
                             DF(Y)*d U/dY  + EF(Y)*dU/dY + FF(Y)*U

                             = G(X,Y)

                        on a rectangle (X greater than or equal to A
                        and less than or equal to B; Y greater than
                        or equal to C and less than or equal to D).
                        Any combination of periodic or mixed boundary
                        conditions is allowed.

 Purpose                The possible boundary conditions are:
                        in the X-direction:
                         (0) Periodic, U(X+B-A,Y)=U(X,Y) for all Y,X
                         (1) U(A,Y), U(B,Y) are specified for all Y
                         (2) U(A,Y), dU(B,Y)/dX+BETA*U(B,Y) are
                             specified for all Y
                         (3) dU(A,Y)/dX+ALPHA*U(A,Y),dU(B,Y)/dX+
                             BETA*U(B,Y) are specified for all Y
                         (4) dU(A,Y)/dX+ALPHA*U(A,Y),U(B,Y) are
                             specified for all Y

                        in the Y-direction:
                         (0) Periodic, U(X,Y+D-C)=U(X,Y) for all X,Y
                         (1) U(X,C),U(X,D) are specified for all X
```

(2) U(X,C),dU(X,D)/dY+XNU*U(X,D) are specified for all X
(3) dU(X,C)/dY+GAMA*U(X,C),dU(X,D)/dY+XNU*U(X,D) are specified for all X
(4) dU(X,C)/dY+GAMA*U(X,C),U(X,D) are specified for all X

Arguments

On Input

INTL
= 0 On initial entry to SEPELI or if any of the arguments C, D, N, NBDCND, COFY are changed from a previous call
= 1 If C, D, N, NBDCND, COFY are unchanged from the previous call.

IORDER
= 2 If a second-order approximation is sought
= 4 If a fourth-order approximation is sought

A,B
The range of the X-independent variable; i.e., X is greater than or equal to A and less than or equal to B. A must be less than B.

M
The number of panels into which the interval [A,B] is subdivided. Hence, there will be M+1 grid points in the X-direction given by XI=A+(I-1)*DLX for I=1,2,...,M+1 where DLX=(B-A)/M is the panel width. M must be less than IDMN and greater than 5.

MBDCND
Indicates the type of boundary condition at X=A and X=B
= 0 If the solution is periodic in X; i.e., U(X+B-A,Y)=U(X,Y) for all Y,X
= 1 If the solution is specified at X=A and X=B; i.e., U(A,Y) and U(B,Y) are specified for all Y
= 2 If the solution is specified at X=A and the boundary condition is mixed at X=B; i.e., U(A,Y) and dU(B,Y)/dX+BETA*U(B,Y) are specified for all Y
= 3 If the boundary conditions at X=A and X=B are mixed; i.e., dU(A,Y)/dX+ALPHA*U(A,Y) and dU(B,Y)/dX+BETA*U(B,Y) are specified for all Y
= 4 If the boundary condition at X=A is mixed and the solution is specified at X=B; i.e., dU(A,Y)/dX+ALPHA*U(A,Y) and U(B,Y) are specified for all Y

BDA
A one-dimensional array of length N+1 that specifies the values of dU(A,Y)/dX+ALPHA*U(A,Y) at X=A, when MBDCND=3 or 4.
BDA(J) = dU(A,YJ)/dX+ALPHA*U(A,YJ);

```
                    J=1,2,...,N+1
                when MBDCND has any other value, BDA is a
                dummy parameter.

On Input        ALPHA
                  The scalar multiplying the solution in case
                  of a mixed boundary condition at X=A (see
                  argument BDA).  If MBDCND = 3,4 then ALPHA is
                  a dummy parameter.

                BDB
                  A one-dimensional array of length N+1 that
                  specifies the values of dU(B,Y)/dX+
                  BETA*U(B,Y) at X=B.  When MBDCND=2 or 3
                       BDB(J) = dU(B,YJ)/dX+BETA*U(B,YJ);
                       J=1,2,...,N+1
                  When MBDCND has any other value, BDB is a
                  dummy parameter.

                BETA
                  The scalar multiplying the solution in case
                  of a mixed boundary condition at X=B (see
                  argument BDB).  If MBDCND=2,3 then BETA is a
                  dummy parameter.

                C,D
                  The range of the Y-independent variable;
                  i.e., Y is greater than or equal to C and
                  less than or equal to D.  C must be less than
                  D.

                N
                  The number of panels into which the interval
                  [C,D] is subdivided.  Hence, there will be
                  N+1 grid points in the Y-direction given by
                  YJ=C+(J-1)*DLY for J=1,2,...,N+1 where
                  DLY=(D-C)/N is the panel width.  In addition,
                  N must be greater than 4.

                NBDCND
                  Indicates the types of boundary conditions at
                  Y=C and Y=D
                  = 0 If the solution is periodic in Y; i.e.,
                      U(X,Y+D-C)=U(X,Y) for all X,Y
                  = 1 If the solution is specified at Y=C and
                      Y = D, i.e., U(X,C) and U(X,D) are
                      specified for all X
                  = 2 If the solution is specified at Y=C and
                      the boundary condition is mixed at Y=D;
                      i.e., U(X,C) and dU(X,D)/dY+XNU*U(X,D)
                      are specified for all X
                  = 3 If the boundary conditions are mixed at
                      Y=C and Y=D; i.e., dU(X,D)/dY+GAMA*U(X,C)
                      and dU(X,D)/dY+XNU*U(X,D) are specified
                      for all X
                  = 4 If the boundary condition is mixed at Y=C
                      and the solution is specified at Y=D;
                      i.e. dU(X,C)/dY+GAMA*U(X,C) and U(X,D)
                      are specified for all X
```

BDC
A one-dimensional array of length M+1 that specifies the value of dU(X,C)/dY+GAMA*U(X,C) at Y=C. When NBDCND=3 or 4

```
BDC(I) = dU(XI,C)/dY + GAMA*U(XI,C);
I=1,2,...,M+1.
```

When NBDCND has any other value, BDC is a dummy parameter.

GAMA
The scalar multiplying the solution in case of a mixed boundary condition at Y=C (see argument BDC). If NBDCND=3,4 then GAMA is a dummy parameter.

BDD
A one-dimensional array of length M+1 that specifies the value of dU(X,D)/dY + XNU*U(X,D) at Y=C. When NBDCND=2 or 3

```
BDD(I) = dU(XI,D)/dY + XNU*U(XI,D);
I=1,2,...,M+1.
```

When NBDCND has any other value, BDD is a dummy parameter.

XNU
The scalar multiplying the solution in case of a mixed boundary condition at Y=D (see argument BDD). If NBDCND=2 or 3 then XNU is a dummy parameter.

COFX
A user-supplied subprogram with parameters X, AFUN, BFUN, CFUN which returns the values of the X-dependent coefficients AF(X), BF(X), CF(X) in the elliptic equation at X.

COFY
A user-supplied subprogram with parameters Y, DFUN, EFUN, FFUN which returns the values of the Y-dependent coefficients DF(Y), EF(Y), FF(Y) in the elliptic equation at Y.

NOTE: COFX and COFY must be declared external in the calling routine. The values returned in AFUN and DFUN must satisfy AFUN*DFUN greater than 0 for A less than X less than B, C less than Y less than D (see IERROR=10). The coefficients provided may lead to a matrix equation which is not diagonally dominant in which case solution may fail (see IERROR=4).

GRHS
A two-dimensional array that specifies the values of the right-hand side of the elliptic equation; i.e., GRHS(I,J)=G(XI,YI), for I=2,...,M; J=2,...,N. At the boundaries, GRHS is defined by

```
MBDCND   GRHS(1,J)   GRHS(M+1,J)
------   ---------   -----------
  0      G(A,YJ)     G(B,YJ)
  1         *           *
  2         *        G(B,YJ)  J=1,2,...,N+1
  3      G(A,YJ)     G(B,YJ)
  4      G(A,YJ)        *

NBDCND   GRHS(I,1)   GRHS(I,N+1)
------   ---------   -----------
  0      G(XI,C)     G(XI,D)
  1         *           *
  2         *        G(XI,D)  I=1,2,...,M+1
  3      G(XI,C)     G(XI,D)
  4      G(XI,C)        *
```

where * means these quantities are not used. GRHS should be dimensioned IDMN by at least N+1 in the calling routine.

USOL

A two-dimensional array that specifies the values of the solution along the boundaries. At the boundaries, USOL is defined by

```
MBDCND   USOL(1,J)   USOL(M+1,J)
------   ---------   -----------
  0         *           *
  1      U(A,YJ)     U(B,YJ)
  2      U(A,YJ)        *     J=1,2,...,N+1
  3         *           *
  4         *        U(B,YJ)

NBDCND   USOL(I,1)   USOL(I,N+1)
------   ---------   -----------
  0         *           *
  1      U(XI,C)     U(XI,D)
  2      U(XI,C)        *     I=1,2,...,M+1
  3         *           *
  4         *        U(XI,D)
```

where * means the quantities are not used in the solution.

If IORDER=2, the user may equivalence GRHS and USOL to save space. Note that in this case the tables specifying the boundaries of the GRHS and USOL arrays determine the boundaries uniquely except at the corners. If the tables call for both G(X,Y) and U(X,Y) at a corner then the solution must be chosen. For example, if MBDCND=2 and NBDCND=4, then U(A,C), U(A,D), U(B,D) must be chosen at the corners in addition to G(B,C).

If IORDER=4, then the two arrays, USOL and GRHS, must be distinct.

USOL should be dimensioned IDMN by at least N+1 in the calling routine.

IDMN
The row (or first) dimension of the arrays GRHS and USOL as it appears in the program calling SEPELI. This parameter is used to specify the variable dimension of GRHS and USOL. IDMN must be at least 7 and greater than or equal to M+1.

W
A one-dimensional array that must be provided by the user for work space. Let K=INT(log2(N+1))+1 and set L=2**(K+1). then (K-2)*L+K+10*N+12*M+27 will suffice as a length of W. THE actual length of W in the calling routine must be set in W(1) (see IERROR=11).

On Output

USOL
Contains the approximate solution to the elliptic equation. USOL(I,J) is the approximation to U(XI,YJ) for I=1,2...,M+1 and J=1,2,...,N+1. The approximation has error O(DLX**2+DLY**2) if called with IORDER=2 and O(DLX**4+DLY**4) if called with IORDER=4.

W
Contains intermediate values that must not be destroyed if SEPELI is called again with INTL=1. In addition W(1) contains the exact minimal length (in floating point) required for the work space (see IERROR=11).

PERTRB
If a combination of periodic or derivative boundary conditions (i.e., ALPHA=BETA=0 if MBDCND=3; GAMA=XNU=0 if NBDCND=3) is specified and if the coefficients of U(X,Y) in the separable elliptic equation are zero (i.e., CF(X)=0 for X greater than or equal to A and less than or equal to B; FF(Y)=0 for Y greater than or equal to C and less than or equal to D) then a solution may not exist. PERTRB is a constant calculated and subtracted from the right-hand side of the matrix equations generated by SEPELI which insures that a solution exists. SEPELI then computes this solution which is a weighted minimal least squares solution to the original problem.

IERROR
An error flag that indicates invalid input parameters or failure to find a solution
= 0 No error
= 1 If A greater than B or C greater than D
= 2 If MBDCND less than 0 or MBDCND greater than 4
= 3 If NBDCND less than 0 or NBDCND greater

than 4
= 4 If attempt to find a solution fails. (the linear system generated is not diagonally dominant.)
= 5 If IDMN is too small (see discussion of IDMN)
= 6 If M is too small or too large (see discussion of M)
= 7 If N is too small (see discussion of N)
= 8 If IORDER is not 2 or 4
= 9 If INTL is not 0 or 1
= 10 If AFUN*DFUN less than or equal to 0 for some interior mesh point (XI,YJ)
= 11 If the work space length input in W(1) is less than the exact minimal work space length required output in W(1).

NOTE (concerning IERROR=4): for the coefficients input through COFX, COFY, the discretization may lead to a block tridiagonal linear system which is not diagonally dominant (for example, this happens if CFUN=0 and BFUN/(2.*DLX) greater than AFUN/DLX**2). In this case solution may fail. This cannot happen in the limit as DLX, DLY approach zero. Hence, the condition may be remedied by taking larger values for M or N.

Entry Points: SEPELI, SPELIP, CHKPRM, CHKSNG, ORTHOG, MINSOL, TRISP, DEFER, DX, DY, BLKTRI, BLKTR1, INDXB, INDXA, INDXC, PROD, PRODP, CPROD, CPRODP, PPADD, PSGF, BSRH, PPSGF, PPSPF, COMPB, TRUN1, STOR1, TQLRAT

Special Conditions: NONE

Common Blocks: SPLP, CBLKT

I/O: NONE

Precision: Single

Specialist: John C. Adams, NCAR, Boulder, Colorado 80307

Language: FORTRAN

History: Developed at NCAR during 1975-76.

Algorithm: SEPELI automatically discretizes the separable elliptic equation which is then solved by a generalized cyclic reduction algorithm in the subroutine, BLKTRI. The fourth-order solution is obtained using 'Deferred Corrections' which is described and referenced in sections, references and method.

Space Required: 14654 (octal) = 6572 (decimal)

Accuracy and Timing: The following computational results were

obtained by solving the sample problem at the end of this write-up on the Control Data 7600. The op count is proportional to M*N*log2(N). In contrast to the other routines in this chapter, accuracy is tested by computing and tabulating second- and fourth-order discretization errors. Below is a table containing computational results. The times given do not include initialization (i.e., times are for INTL=1). Note that the fourth-order accuracy is not realized until the mesh is sufficiently refined.

M	N	Second-order Execution Time (M SEC)	Fourth-order Execution Time (M SEC)	Second-order Error	Fourth-order Error
6	6	6	14	6.8E-1	1.2E0
14	14	23	58	1.4E-1	1.8E-1
30	30	100	247	3.2E-2	9.7E-3
62	62	445	1,091	7.5E-3	3.0E-4
126	126	2,002	4,772	1.8E-3	3.5E-6

Portability — There are no machine-dependent constants.

Required Resident Routines — SQRT, ABS, LOG

References — Keller, H.B., 'Numerical Methods for Two-point Boundary-value Problems', Blaisdel (1968), Waltham, Mass.

Swarztrauber, P., and R. Sweet (1975): 'Efficient FORTRAN Subprograms for The Solution of Elliptic Partial Differential Equations'. NCAR Technical Note NCAR-TN/IA-109, pp. 135-137.

```
***REFERENCES  H. B. Keller, Numerical Methods for Two-point
                 Boundary-value Problems, Blaisdel, Waltham, Mass.,
                 1968.
               P. N. Swarztrauber and R. Sweet, Efficient Fortran
                 subprograms for the solution of elliptic equations,
                 NCAR TN/IA-109, July 1975, 138 pp.
***ROUTINES CALLED  CHKPRM, SPELIP
***REVISION HISTORY  (YYMMDD)
   801001  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890531  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SEPX4

```
      SUBROUTINE SEPX4(IORDER,A,B,M,MBDCND,BDA,ALPHA,BDB,
     + BETA,C,D,N,NBDCND,BDC,BDD,COFX,
     + GRHS,USOL,IDMN,W,PERTRB,IERROR)
***BEGIN PROLOGUE  SEPX4
***PURPOSE  Solve for either the second or fourth order finite
            difference approximation to the solution of a separable
            elliptic partial differential equation on a rectangle.
            Any combination of periodic or mixed boundary conditions is
            allowed.
***LIBRARY   SLATEC (FISHPACK)
***CATEGORY  I2B1A2
***TYPE      SINGLE PRECISION (SEPX4-S)
***KEYWORDS  ELLIPTIC, FISHPACK, HELMHOLTZ, PDE, SEPARABLE
***AUTHOR  Adams, J., (NCAR)
           Swarztrauber, P. N., (NCAR)
           Sweet, R., (NCAR)
***DESCRIPTION

 Purpose                 SEPX4 solves for either the second-order
                         finite difference approximation or a
                         fourth-order approximation  to the
                         solution of a separable elliptic equation
                              AF(X)*UXX+BF(X)*UX+CF(X)*U+UYY = G(X,Y)

                         on a rectangle (X greater than or equal to A
                         and less than or equal to B; Y greater than
                         or equal to C and less than or equal to D).
                         Any combination of periodic or mixed boundary
                         conditions is allowed.
                         If boundary conditions in the X direction
                         are periodic (see MBDCND=0 below) then the
                         coefficients must satisfy
                         AF(X)=C1,BF(X)=0,CF(X)=C2 for all X.
                         Here C1,C2 are constants, C1.GT.0.

                         The possible boundary conditions are
                         in the X-direction:
                          (0) Periodic, U(X+B-A,Y)=U(X,Y) for all Y,X
                          (1) U(A,Y), U(B,Y) are specified for all Y
                          (2) U(A,Y), dU(B,Y)/dX+BETA*U(B,Y) are
                              specified for all Y
                          (3) dU(A,Y)/dX+ALPHA*U(A,Y),dU(B,Y)/dX+
                              BETA*U(B,Y) are specified for all Y
                          (4) dU(A,Y)/dX+ALPHA*U(A,Y),U(B,Y) are
                              specified for all Y

                         In the Y-direction:
                          (0) Periodic, U(X,Y+D-C)=U(X,Y) for all X,Y
                          (1) U(X,C),U(X,D) are specified for all X
                          (2) U(X,C),dU(X,D)/dY are specified for all X
                          (3) dU(X,C)/DY,dU(X,D)/dY are specified for
                             all X
                         (4) dU(X,C)/DY,U(X,D) are specified for all X

 Usage                   Call SEPX4(IORDER,A,B,M,MBDCND,BDA,ALPHA,BDB,
                                    BETA,C,D,N,NBDCND,BDC,BDD,COFX,
```

```
                    GRHS,USOL,IDMN,W,PERTRB,IERROR)
```

Arguments

```
IORDER
  = 2 If a second-order approximation is sought
  = 4 If a fourth-order approximation is sought

A,B
  The range of the X-independent variable;
  i.e., X is greater than or equal to A and
  less than or equal to B.  A must be less than
  B.

M
  The number of panels into which the interval
  [A,B] is subdivided.  Hence, there will be
  M+1 grid points in the X-direction given by
  XI=A+(I-1)*DLX for I=1,2,...,M+1 where
  DLX=(B-A)/M is the panel width.  M must be
  less than IDMN and greater than 5.

MBDCND
  Indicates the type of boundary condition at
  X=A and X=B
  = 0 If the solution is periodic in X; i.e.,
      U(X+B-A,Y)=U(X,Y) for all Y,X
  = 1 If the solution is specified at X=A and
      X=B; i.e., U(A,Y) and U(B,Y) are
      specified for all Y
  = 2 If the solution is specified at X=A and
      the boundary condition is mixed at X=B;
      i.e., U(A,Y) and dU(B,Y)/dX+BETA*U(B,Y)
      are specified for all Y
  = 3 If the boundary conditions at X=A and X=B
      are mixed; i.e., dU(A,Y)/dX+ALPHA*U(A,Y)
      and dU(B,Y)/dX+BETA*U(B,Y) are specified
      for all Y
  = 4 If the boundary condition at X=A is mixed
      and the solution is specified at X=B;
      i.e., dU(A,Y)/dX+ALPHA*U(A,Y) and U(B,Y)
      are specified for all Y

BDA
  A one-dimensional array of length N+1 that
  specifies the values of dU(A,Y)/dX+
  ALPHA*U(A,Y) at X=A, when MBDCND=3 or 4.
       BDA(J) = dU(A,YJ)/dX+ALPHA*U(A,YJ);
       J=1,2,...,N+1
  When MBDCND has any other value, BDA is a
  dummy parameter.
```

On Input

```
ALPHA
  The scalar multiplying the solution in case
  of a mixed boundary condition AT X=A (see
  argument BDA).  If MBDCND = 3,4 then ALPHA is
  a dummy parameter.

BDB
  A one-dimensional array of length N+1 that
```

```
  specifies the values of dU(B,Y)/dX+
  BETA*U(B,Y) at X=B.  when MBDCND=2 or 3
       BDB(J) = dU(B,YJ)/dX+BETA*U(B,YJ);
       J=1,2,...,N+1
  When MBDCND has any other value, BDB is a
  dummy parameter.

BETA
  The scalar multiplying the solution in case
  of a mixed boundary condition at X=B (see
  argument BDB).  If MBDCND=2,3 then BETA is a
  dummy parameter.

C,D
  The range of the Y-independent variable;
  i.e., Y is greater than or equal to C and
  less than or equal to D.  C must be less than
  D.

N
  The number of panels into which the interval
  [C,D] is subdivided.  Hence, there will be
  N+1 grid points in the Y-direction given by
  YJ=C+(J-1)*DLY for J=1,2,...,N+1 where
  DLY=(D-C)/N is the panel width.  In addition,
  N must be greater than 4.

NBDCND
  Indicates the types of boundary conditions at
  Y=C and Y=D
  = 0 If the solution is periodic in Y; i.e.,
      U(X,Y+D-C)=U(X,Y) for all X,Y
  = 1 If the solution is specified at Y=C and
      Y = D, i.e., U(X,C) and U(X,D) are
      specified for all X
  = 2 If the solution is specified at Y=C and
      the boundary condition is mixed at Y=D;
      i.e., dU(X,C)/dY and U(X,D)
      are specified for all X
  = 3 If the boundary conditions are mixed at
      Y= C and Y=D i.e., dU(X,D)/DY
      and dU(X,D)/dY are specified
      for all X
  = 4 If the boundary condition is mixed at Y=C
      and the solution is specified at Y=D;
      i.e. dU(X,C)/dY+GAMA*U(X,C) and U(X,D)
      are specified for all X

BDC
  A one-dimensional array of length M+1 that
  specifies the value dU(X,C)/DY
  at Y=C.  When NBDCND=3 or 4
    BDC(I) = dU(XI,C)/DY
     I=1,2,...,M+1.
  When NBDCND has any other value, BDC is a
  dummy parameter.


BDD
  A one-dimensional array of length M+1 that
```

specifies the value of dU(X,D)/DY
at Y=D. When NBDCND=2 or 3
BDD(I)=dU(XI,D)/DY
I=1,2,...,M+1.
When NBDCND has any other value, BDD is a
dummy parameter.

COFX
A user-supplied subprogram with
parameters X, AFUN, BFUN, CFUN which
returns the values of the X-dependent
coefficients AF(X), BF(X), CF(X) in
the elliptic equation at X.
If boundary conditions in the X direction
are periodic then the coefficients
must satisfy AF(X)=C1,BF(X)=0,CF(X)=C2 for
all X. Here C1.GT.0 and C2 are constants.

Note that COFX must be declared external
in the calling routine.

GRHS
A two-dimensional array that specifies the
values of the right-hand side of the elliptic
equation; i.e., GRHS(I,J)=G(XI,YI), for
I=2,...,M; J=2,...,N. At the boundaries,
GRHS is defined by

```
MBDCND   GRHS(1,J)   GRHS(M+1,J)
------   ---------   -----------
  0      G(A,YJ)     G(B,YJ)
  1         *           *
  2         *        G(B,YJ)  J=1,2,...,N+1
  3      G(A,YJ)     G(B,YJ)
  4      G(A,YJ)        *

NBDCND   GRHS(I,1)   GRHS(I,N+1)
------   ---------   -----------
  0      G(XI,C)     G(XI,D)
  1         *           *
  2         *        G(XI,D)  I=1,2,...,M+1
  3      G(XI,C)     G(XI,D)
  4      G(XI,C)        *
```

where * means these quantities are not used.
GRHS should be dimensioned IDMN by at least
N+1 in the calling routine.

USOL
A two-dimensional array that specifies the
values of the solution along the boundaries.
At the boundaries, USOL is defined by

```
MBDCND   USOL(1,J)   USOL(M+1,J)
------   ---------   -----------
  0         *           *
  1      U(A,YJ)     U(B,YJ)
  2      U(A,YJ)        *     J=1,2,...,N+1
  3         *           *
```

```
  4         *        U(B,YJ)

NBDCND   USOL(I,1)   USOL(I,N+1)
------   ---------   -----------
  0         *           *
  1      U(XI,C)     U(XI,D)
  2      U(XI,C)        *     I=1,2,...,M+1
  3         *           *
  4         *        U(XI,D)
```

where * means the quantities are not used in the solution.

If IORDER=2, the user may equivalence GRHS and USOL to save space. Note that in this case the tables specifying the boundaries of the GRHS and USOL arrays determine the boundaries uniquely except at the corners. If the tables call for both G(X,Y) and U(X,Y) at a corner then the solution must be chosen. For example, if MBDCND=2 and NBDCND=4, then U(A,C), U(A,D), U(B,D) must be chosen at the corners in addition to G(B,C).

If IORDER=4, then the two arrays, USOL and GRHS, must be distinct.

USOL should be dimensioned IDMN by at least N+1 in the calling routine.

IDMN
The row (or first) dimension of the arrays GRHS and USOL as it appears in the program calling SEPX4. This parameter is used to specify the variable dimension of GRHS and USOL. IDMN must be at least 7 and greater than or equal to M+1.

W
A one-dimensional array that must be provided by the user for work space. 10*N+(16+INT(log2(N)))*(M+1)+23 will suffice as a length for W. The actual length of W in the calling routine must be set in W(1) (see IERROR=11).

On Output

USOL
Contains the approximate solution to the elliptic equation. USOL(I,J) is the approximation to U(XI,YJ) for I=1,2...,M+1 and J=1,2,...,N+1. The approximation has error O(DLX**2+DLY**2) if called with IORDER=2 and O(DLX**4+DLY**4) if called with IORDER=4.

W
W(1) contains the exact minimal length (in floating point) required for the work space (see IERROR=11).

PERTRB
If a combination of periodic or derivative boundary conditions (i.e., ALPHA=BETA=0 if MBDCND=3) is specified and if CF(X)=0 for all X, then a solution to the discretized matrix equation may not exist (reflecting the non-uniqueness of solutions to the PDE). PERTRB is a constant calculated and subtracted from the right hand side of the matrix equation insuring the existence of a solution. SEPX4 computes this solution which is a weighted minimal least squares solution to the original problem. If singularity is not detected PERTRB=0.0 is returned by SEPX4.

IERROR
An error flag that indicates invalid input parameters or failure to find a solution
= 0 No error
= 1 If A greater than B or C greater than D
= 2 If MBDCND less than 0 or MBDCND greater than 4
= 3 If NBDCND less than 0 or NBDCND greater than 4
= 4 If attempt to find a solution fails. (the linear system generated is not diagonally dominant.)
= 5 If IDMN is too small (see discussion of IDMN)
= 6 If M is too small or too large (see discussion of M)
= 7 If N is too small (see discussion of N)
= 8 If IORDER is not 2 or 4
= 10 If AFUN is less than or equal to zero for some interior mesh point XI
= 11 If the work space length input in W(1) is less than the exact minimal work space length required output in W(1).
= 12 If MBDCND=0 and AF(X)=CF(X)=constant or BF(X)=0 for all X is not true.

*Long Description:

Dimension of Arguments	BDA(N+1), BDB(N+1), BDC(M+1), BDD(M+1), USOL(IDMN,N+1), GRHS(IDMN,N+1), W (see argument list)
Latest Revision	October 1980
Special Conditions	NONE
Common Blocks	SPL4
I/O	NONE
Precision	Single
Required Library Files	NONE

Specialist	John C. Adams, NCAR, Boulder, Colorado 80307
Language	FORTRAN
Entry Points	SEPX4,SPELI4,CHKPR4,CHKSN4,ORTHO4,MINSO4,TRIS4, DEFE4,DX4,DY4
History	SEPX4 was developed by modifying the ULIB routine SEPELI during October 1978. It should be used instead of SEPELI whenever possible. The increase in speed is at least a factor of three.
Algorithm	SEPX4 automatically discretizes the separable elliptic equation which is then solved by a generalized cyclic reduction algorithm in the subroutine POIS. The fourth order solution is obtained using the technique of deferred corrections referenced below.
References	Keller, H.B., 'Numerical Methods for Two-point Boundary-value Problems', Blaisdel (1968), Waltham, Mass. Swarztrauber, P., and R. Sweet (1975): 'Efficient FORTRAN Subprograms For The Solution of Elliptic Partial Differential Equations'. NCAR Technical Note NCAR-TN/IA-109, pp. 135-137.

```
***REFERENCES  H. B. Keller, Numerical Methods for Two-point
                 Boundary-value Problems, Blaisdel, Waltham, Mass.,
                 1968.
               P. N. Swarztrauber and R. Sweet, Efficient Fortran
                 subprograms for the solution of elliptic equations,
                 NCAR TN/IA-109, July 1975, 138 pp.
***ROUTINES CALLED  CHKPR4, SPELI4
***REVISION HISTORY  (YYMMDD)
   801001  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890531  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920122  Minor corrections and modifications to prologue.  (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SGBCO

```
      SUBROUTINE SGBCO (ABD, LDA, N, ML, MU, IPVT, RCOND, Z)
***BEGIN PROLOGUE  SGBCO
***PURPOSE  Factor a band matrix by Gaussian elimination and
            estimate the condition number of the matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A2
***TYPE      SINGLE PRECISION (SGBCO-S, DGBCO-D, CGBCO-C)
***KEYWORDS  BANDED, CONDITION NUMBER, LINEAR ALGEBRA, LINPACK,
             MATRIX FACTORIZATION
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SBGCO factors a real band matrix by Gaussian
      elimination and estimates the condition of the matrix.

      If  RCOND  is not needed, SGBFA is slightly faster.
      To solve  A*X = B , follow SBGCO by SGBSL.
      To compute  INVERSE(A)*C , follow SBGCO by SGBSL.
      To compute  DETERMINANT(A) , follow SBGCO by SGBDI.

      On Entry

         ABD     REAL(LDA, N)
                 contains the matrix in band storage.  The columns
                 of the matrix are stored in the columns of  ABD  and
                 the diagonals of the matrix are stored in rows
                 ML+1 through 2*ML+MU+1 of  ABD .
                 See the comments below for details.

         LDA     INTEGER
                 the leading dimension of the array  ABD .
                 LDA must be .GE. 2*ML + MU + 1 .

         N       INTEGER
                 the order of the original matrix.

         ML      INTEGER
                 number of diagonals below the main diagonal.
                 0 .LE. ML .LT. N .

         MU      INTEGER
                 number of diagonals above the main diagonal.
                 0 .LE. MU .LT. N .
                 More efficient if  ML .LE. MU .

      On Return

         ABD     an upper triangular matrix in band storage and
                 the multipliers which were used to obtain it.
                 The factorization can be written  A = L*U  where
                 L  is a product of permutation and unit lower
                 triangular matrices and  U  is upper triangular.

         IPVT    INTEGER(N)
                 an integer vector of pivot indices.
```

```
RCOND   REAL
        an estimate of the reciprocal condition of  A .
        For the system  A*X = B , relative perturbations
        in  A  and  B  of size  EPSILON  may cause
        relative perturbations in  X  of size  EPSILON/RCOND .
        If  RCOND  is so small that the logical expression
                  1.0 + RCOND .EQ. 1.0
        is true, then  A  may be singular to working
        precision.  In particular,  RCOND  is zero  if
        exact singularity is detected or the estimate
        underflows.

Z       REAL(N)
        a work vector whose contents are usually unimportant.
        If  A  is close to a singular matrix, then  Z  is
        an approximate null vector in the sense that
        NORM(A*Z) = RCOND*NORM(A)*NORM(Z) .
```

Band Storage

If A is a band matrix, the following program segment will set up the input.

```
      ML = (band width below the diagonal)
      MU = (band width above the diagonal)
      M = ML + MU + 1
      DO 20 J = 1, N
         I1 = MAX(1, J-MU)
         I2 = MIN(N, J+ML)
         DO 10 I = I1, I2
            K = I - J + M
            ABD(K,J) = A(I,J)
10       CONTINUE
20    CONTINUE
```

This uses rows ML+1 through 2*ML+MU+1 of ABD . In addition, the first ML rows in ABD are used for elements generated during the triangularization. The total number of rows needed in ABD is 2*ML+MU+1 . The ML+MU by ML+MU upper left triangle and the ML by ML lower right triangle are not referenced.

Example: If the original matrix is

```
11 12 13  0  0  0
21 22 23 24  0  0
 0 32 33 34 35  0
 0  0 43 44 45 46
 0  0  0 54 55 56
 0  0  0  0 65 66
```

then N = 6, ML = 1, MU = 2, LDA .GE. 5 and ABD should contain

```
 *  *  *  +  +  +  , * = not used
 *  * 13 24 35 46  , + = used for pivoting
 * 12 23 34 45 56
11 22 33 44 55 66
21 32 43 54 65  *
```

***REFERENCES J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.

```
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SASUM, SAXPY, SDOT, SGBFA, SSCAL
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SGBDI

```
      SUBROUTINE SGBDI (ABD, LDA, N, ML, MU, IPVT, DET)
***BEGIN PROLOGUE  SGBDI
***PURPOSE  Compute the determinant of a band matrix using the factors
            computed by SGBCO or SGBFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D3A2
***TYPE      SINGLE PRECISION (SGBDI-S, DGBDI-D, CGBDI-C)
***KEYWORDS  BANDED, DETERMINANT, INVERSE, LINEAR ALGEBRA, LINPACK,
             MATRIX
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SGBDI computes the determinant of a band matrix
      using the factors computed by SBGCO or SGBFA.
      If the inverse is needed, use SGBSL  N  times.

      On Entry

         ABD     REAL(LDA, N)
                 the output from SBGCO or SGBFA.

         LDA     INTEGER
                 the leading dimension of the array  ABD .

         N       INTEGER
                 the order of the original matrix.

         ML      INTEGER
                 number of diagonals below the main diagonal.

         MU      INTEGER
                 number of diagonals above the main diagonal.

         IPVT    INTEGER(N)
                 the pivot vector from SBGCO or SGBFA.

      On Return

         DET     REAL(2)
                 determinant of original matrix.
                 Determinant = DET(1) * 10.0**DET(2)
                 with  1.0 .LE. ABS(DET(1)) .LT. 10.0
                 or  DET(1) = 0.0 .

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SGBFA

```
      SUBROUTINE SGBFA (ABD, LDA, N, ML, MU, IPVT, INFO)
***BEGIN PROLOGUE  SGBFA
***PURPOSE  Factor a band matrix using Gaussian elimination.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A2
***TYPE      SINGLE PRECISION (SGBFA-S, DGBFA-D, CGBFA-C)
***KEYWORDS  BANDED, LINEAR ALGEBRA, LINPACK, MATRIX FACTORIZATION
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SGBFA factors a real band matrix by elimination.

      SGBFA is usually called by SBGCO, but it can be called
      directly with a saving in time if  RCOND  is not needed.

      On Entry

         ABD     REAL(LDA, N)
                 contains the matrix in band storage.  The columns
                 of the matrix are stored in the columns of  ABD  and
                 the diagonals of the matrix are stored in rows
                 ML+1 through 2*ML+MU+1 of  ABD .
                 See the comments below for details.

         LDA     INTEGER
                 the leading dimension of the array  ABD .
                 LDA must be .GE. 2*ML + MU + 1 .

         N       INTEGER
                 the order of the original matrix.

         ML      INTEGER
                 number of diagonals below the main diagonal.
                 0 .LE. ML .LT. N .

         MU      INTEGER
                 number of diagonals above the main diagonal.
                 0 .LE. MU .LT. N .
                 More efficient if  ML .LE. MU .
      On Return

         ABD     an upper triangular matrix in band storage and
                 the multipliers which were used to obtain it.
                 The factorization can be written  A = L*U , where
                 L  is a product of permutation and unit lower
                 triangular matrices and  U  is upper triangular.

         IPVT    INTEGER(N)
                 an integer vector of pivot indices.

         INFO    INTEGER
                 = 0  normal value.
                 = K  if  U(K,K) .EQ. 0.0 .  This is not an error
                      condition for this subroutine, but it does
                      indicate that SGBSL will divide by zero if
                      called.  Use  RCOND  in SBGCO for a reliable
```

indication of singularity.

Band Storage

If A is a band matrix, the following program segment will set up the input.

```
      ML = (band width below the diagonal)
      MU = (band width above the diagonal)
      M = ML + MU + 1
      DO 20 J = 1, N
         I1 = MAX(1, J-MU)
         I2 = MIN(N, J+ML)
         DO 10 I = I1, I2
            K = I - J + M
            ABD(K,J) = A(I,J)
   10    CONTINUE
   20 CONTINUE
```

This uses rows ML+1 through 2*ML+MU+1 of ABD . In addition, the first ML rows in ABD are used for elements generated during the triangularization. The total number of rows needed in ABD is 2*ML+MU+1 . The ML+MU by ML+MU upper left triangle and the ML by ML lower right triangle are not referenced.

***REFERENCES J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED ISAMAX, SAXPY, SSCAL
***REVISION HISTORY (YYMMDD)
780814 DATE WRITTEN
890531 Changed all specific intrinsics to generic. (WRB)
890831 Modified array declarations. (WRB)
890831 REVISION DATE from Version 3.2
891214 Prologue converted to Version 4.0 format. (BAB)
900326 Removed duplicate information from DESCRIPTION section. (WRB)
920501 Reformatted the REFERENCES section. (WRB)
END PROLOGUE

SGBMV

```
      SUBROUTINE SGBMV (TRANS, M, N, KL, KU, ALPHA, A, LDA,
     +     X, INCX, BETA, Y, INCY)
***BEGIN PROLOGUE  SGBMV
***PURPOSE  Multiply a real vector by a real general band matrix.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (SGBMV-S, DGBMV-D, CGBMV-C)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  SGBMV  performs one of the matrix-vector operations

     y := alpha*A*x + beta*y,   or   y := alpha*A'*x + beta*y,

  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 performed 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  - REAL            .
         On entry, ALPHA specifies the scalar alpha.
         Unchanged on exit.

A      - REAL             array of DIMENSION ( LDA, n ).
         Before entry, the leading ( kl + ku + 1 ) by n part of the
         array A must contain the matrix of coefficients, 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 position 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      - REAL             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 elements of
         X. INCX must not be zero.
         Unchanged on exit.

BETA   - REAL            .
         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      - REAL             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 elements of
         Y. INCY must not be zero.
```

```
          Unchanged on exit.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

SGBSL

```
      SUBROUTINE SGBSL (ABD, LDA, N, ML, MU, IPVT, B, JOB)
***BEGIN PROLOGUE  SGBSL
***PURPOSE  Solve the real band system A*X=B or TRANS(A)*X=B using
            the factors computed by SGBCO or SGBFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A2
***TYPE      SINGLE PRECISION (SGBSL-S, DGBSL-D, CGBSL-C)
***KEYWORDS  BANDED, LINEAR ALGEBRA, LINPACK, MATRIX, SOLVE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SGBSL solves the real band system
      A * X = B  or  TRANS(A) * X = B
      using the factors computed by SBGCO or SGBFA.

      On Entry

         ABD     REAL(LDA, N)
                 the output from SBGCO or SGBFA.

         LDA     INTEGER
                 the leading dimension of the array  ABD .

         N       INTEGER
                 the order of the original matrix.

         ML      INTEGER
                 number of diagonals below the main diagonal.

         MU      INTEGER
                 number of diagonals above the main diagonal.

         IPVT    INTEGER(N)
                 the pivot vector from SBGCO or SGBFA.

         B       REAL(N)
                 the right hand side vector.

         JOB     INTEGER
                 = 0         to solve  A*X = B ,
                 = nonzero   to solve  TRANS(A)*X = B , where
                             TRANS(A)  is the transpose.

      On Return

         B       the solution vector  X .

      Error Condition

         A division by zero will occur if the input factor contains a
         zero on the diagonal.  Technically, this indicates singularity,
         but it is often caused by improper arguments or improper
         setting of LDA .  It will not occur if the subroutines are
         called correctly and if SBGCO has set RCOND .GT. 0.0
         or SGBFA has set INFO .EQ. 0 .
```

```
     To compute  INVERSE(A) * C  where  C  is a matrix
     with  P  columns
           CALL SBGCO(ABD,LDA,N,ML,MU,IPVT,RCOND,Z)
           If (RCOND is too small) GO TO ...
           DO 10 J = 1, P
              CALL SGBSL(ABD,LDA,N,ML,MU,IPVT,C(1,J),0)
        10 CONTINUE

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SDOT
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SGECO

```
      SUBROUTINE SGECO (A, LDA, N, IPVT, RCOND, Z)
***BEGIN PROLOGUE  SGECO
***PURPOSE  Factor a matrix using Gaussian elimination and estimate
            the condition number of the matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A1
***TYPE      SINGLE PRECISION (SGECO-S, DGECO-D, CGECO-C)
***KEYWORDS  CONDITION NUMBER, GENERAL MATRIX, LINEAR ALGEBRA, LINPACK,
             MATRIX FACTORIZATION
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SGECO factors a real matrix by Gaussian elimination
      and estimates the condition of the matrix.

      If  RCOND  is not needed, SGEFA is slightly faster.
      To solve  A*X = B , follow SGECO by SGESL.
      To compute  INVERSE(A)*C , follow SGECO by SGESL.
      To compute  DETERMINANT(A) , follow SGECO by SGEDI.
      To compute  INVERSE(A) , follow SGECO by SGEDI.

      On Entry

         A       REAL(LDA, N)
                 the matrix to be factored.

         LDA     INTEGER
                 the leading dimension of the array  A .

         N       INTEGER
                 the order of the matrix  A .

      On Return

         A       an upper triangular matrix and the multipliers
                 which were used to obtain it.
                 The factorization can be written  A = L*U , where
                 L  is a product of permutation and unit lower
                 triangular matrices and  U  is upper triangular.

         IPVT    INTEGER(N)
                 an integer vector of pivot indices.

         RCOND   REAL
                 an estimate of the reciprocal condition of  A .
                 For the system  A*X = B , relative perturbations
                 in  A  and  B  of size  EPSILON  may cause
                 relative perturbations in  X  of size  EPSILON/RCOND .
                 If  RCOND  is so small that the logical expression
                           1.0 + RCOND .EQ. 1.0
                 is true, then  A  may be singular to working
                 precision.  In particular,  RCOND  is zero  if
                 exact singularity is detected or the estimate
                 underflows.

         Z       REAL(N)
```

```
                a work vector whose contents are usually unimportant.
                If  A  is close to a singular matrix, then  Z  is
                an approximate null vector in the sense that
                NORM(A*Z) = RCOND*NORM(A)*NORM(Z) .

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SASUM, SAXPY, SDOT, SGEFA, SSCAL
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SGEDI

```
      SUBROUTINE SGEDI (A, LDA, N, IPVT, DET, WORK, JOB)
***BEGIN PROLOGUE  SGEDI
***PURPOSE  Compute the determinant and inverse of a matrix using the
            factors computed by SGECO or SGEFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A1, D3A1
***TYPE      SINGLE PRECISION (SGEDI-S, DGEDI-D, CGEDI-C)
***KEYWORDS  DETERMINANT, INVERSE, LINEAR ALGEBRA, LINPACK, MATRIX
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SGEDI computes the determinant and inverse of a matrix
      using the factors computed by SGECO or SGEFA.

      On Entry

         A       REAL(LDA, N)
                 the output from SGECO or SGEFA.

         LDA     INTEGER
                 the leading dimension of the array  A .

         N       INTEGER
                 the order of the matrix  A .

         IPVT    INTEGER(N)
                 the pivot vector from SGECO or SGEFA.

         WORK    REAL(N)
                 work vector.  Contents destroyed.

         JOB     INTEGER
                 = 11   both determinant and inverse.
                 = 01   inverse only.
                 = 10   determinant only.

      On Return

         A       inverse of original matrix if requested.
                 Otherwise unchanged.

         DET     REAL(2)
                 determinant of original matrix if requested.
                 Otherwise not referenced.
                 Determinant = DET(1) * 10.0**DET(2)
                 with  1.0 .LE. ABS(DET(1)) .LT. 10.0
                 or  DET(1) .EQ. 0.0 .

      Error Condition

         A division by zero will occur if the input factor contains
         a zero on the diagonal and the inverse is requested.
         It will not occur if the subroutines are called correctly
         and if SGECO has set RCOND .GT. 0.0 or SGEFA has set
         INFO .EQ. 0 .
```

```
***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SSCAL, SSWAP
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SGEEV

```
      SUBROUTINE SGEEV (A, LDA, N, E, V, LDV, WORK, JOB, INFO)
***BEGIN PROLOGUE  SGEEV
***PURPOSE  Compute the eigenvalues and, optionally, the eigenvectors
            of a real general matrix.
***LIBRARY   SLATEC
***CATEGORY  D4A2
***TYPE      SINGLE PRECISION (SGEEV-S, CGEEV-C)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, GENERAL MATRIX
***AUTHOR  Kahaner, D. K., (NBS)
           Moler, C. B., (U. of New Mexico)
           Stewart, G. W., (U. of Maryland)
***DESCRIPTION

      Abstract
       SGEEV computes the eigenvalues and, optionally,
       the eigenvectors of a general real matrix.

      Call Sequence Parameters-
        (The values of parameters marked with * (star) will be changed
          by SGEEV.)

         A*      REAL(LDA,N)
                 real nonsymmetric input matrix.

         LDA     INTEGER
                 set by the user to
                 the leading dimension of the real array A.

         N       INTEGER
                 set by the user to
                 the order of the matrices A and V, and
                 the number of elements in E.

         E*      COMPLEX(N)
                 on return from SGEEV, E contains the eigenvalues of A.
                 See also INFO below.

         V*      COMPLEX(LDV,N)
                 on return from SGEEV, if the user has set JOB
                 = 0        V is not referenced.
                 = nonzero  the N eigenvectors of A are stored in the
                 first N columns of V.  See also INFO below.
                 (Note that if the input matrix A is nearly degenerate,
                  V may be badly conditioned, i.e., may have nearly
                  dependent columns.)

         LDV     INTEGER
                 set by the user to
                 the leading dimension of the array V if JOB is also
                 set nonzero.  In that case, N must be .LE. LDV.
                 If JOB is set to zero, LDV is not referenced.

         WORK*   REAL(2N)
                 temporary storage vector.  Contents changed by SGEEV.

         JOB     INTEGER
```

```
                set by the user to
                = 0        eigenvalues only to be calculated by SGEEV.
                           Neither V nor LDV is referenced.
                = nonzero  eigenvalues and vectors to be calculated.
                           In this case, A & V must be distinct arrays.
                           Also, if LDA .GT. LDV, SGEEV changes all the
                           elements of A thru column N.  If LDA < LDV,
                           SGEEV changes all the elements of V through
                           column N. If LDA = LDV, only A(I,J) and V(I,
                           J) for I,J = 1,...,N are changed by SGEEV.

        INFO*   INTEGER
                on return from SGEEV the value of INFO is
                = 0  normal return, calculation successful.
                = K  if the eigenvalue iteration fails to converge,
                     eigenvalues K+1 through N are correct, but
                     no eigenvectors were computed even if they were
                     requested (JOB nonzero).

      Error Messages
           No. 1  recoverable  N is greater than LDA
           No. 2  recoverable  N is less than one.
           No. 3  recoverable  JOB is nonzero and N is greater than LDV
           No. 4  warning      LDA > LDV, elements of A other than the
                               N by N input elements have been changed.
           No. 5  warning      LDA < LDV, elements of V other than the
                               N x N output elements have been changed.

***REFERENCES  (NONE)
***ROUTINES CALLED  BALANC, BALBAK, HQR, HQR2, ORTHES, ORTRAN, SCOPY,
                    SCOPYM, XERMSG
***REVISION HISTORY  (YYMMDD)
   800808  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890531  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   END PROLOGUE
```

SGEFA

```
      SUBROUTINE SGEFA (A, LDA, N, IPVT, INFO)
***BEGIN PROLOGUE  SGEFA
***PURPOSE  Factor a matrix using Gaussian elimination.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A1
***TYPE      SINGLE PRECISION (SGEFA-S, DGEFA-D, CGEFA-C)
***KEYWORDS  GENERAL MATRIX, LINEAR ALGEBRA, LINPACK,
             MATRIX FACTORIZATION
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SGEFA factors a real matrix by Gaussian elimination.

      SGEFA is usually called by SGECO, but it can be called
      directly with a saving in time if  RCOND  is not needed.
      (Time for SGECO) = (1 + 9/N)*(Time for SGEFA) .

      On Entry

         A       REAL(LDA, N)
                 the matrix to be factored.

         LDA     INTEGER
                 the leading dimension of the array  A .

         N       INTEGER
                 the order of the matrix  A .

      On Return

         A       an upper triangular matrix and the multipliers
                 which were used to obtain it.
                 The factorization can be written  A = L*U , where
                 L  is a product of permutation and unit lower
                 triangular matrices and  U  is upper triangular.

         IPVT    INTEGER(N)
                 an integer vector of pivot indices.

         INFO    INTEGER
                 = 0  normal value.
                 = K  if  U(K,K) .EQ. 0.0 .  This is not an error
                      condition for this subroutine, but it does
                      indicate that SGESL or SGEDI will divide by zero
                      if called.  Use  RCOND  in SGECO for a reliable
                      indication of singularity.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  ISAMAX, SAXPY, SSCAL
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
```

```
        (WRB)
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

SGEFS

```
      SUBROUTINE SGEFS (A, LDA, N, V, ITASK, IND, WORK, IWORK)
***BEGIN PROLOGUE  SGEFS
***PURPOSE  Solve a general system of linear equations.
***LIBRARY   SLATEC
***CATEGORY  D2A1
***TYPE      SINGLE PRECISION (SGEFS-S, DGEFS-D, CGEFS-C)
***KEYWORDS  COMPLEX LINEAR EQUATIONS, GENERAL MATRIX,
             GENERAL SYSTEM OF LINEAR EQUATIONS
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

    Subroutine SGEFS solves a general NxN system of single
    precision linear equations using LINPACK subroutines SGECO
    and SGESL.  That is, if A is an NxN real matrix and if X
    and B are real N-vectors, then SGEFS solves the equation

                           A*X=B.

    The matrix A is first factored into upper and lower tri-
    angular matrices U and L using partial pivoting.  These
    factors and the pivoting information are used to find the
    solution vector X.  An approximate condition number is
    calculated to provide a rough estimate of the number of
    digits of accuracy in the computed solution.

    If the equation A*X=B is to be solved for more than one vector
    B, the factoring of A does not need to be performed again and
    the option to only solve (ITASK .GT. 1) will be faster for
    the succeeding solutions.  In this case, the contents of A,
    LDA, N and IWORK must not have been altered by the user follow-
    ing factorization (ITASK=1).  IND will not be changed by SGEFS
    in this case.

  Argument Description ***

    A      REAL(LDA,N)
             on entry, the doubly subscripted array with dimension
               (LDA,N) which contains the coefficient matrix.
             on return, an upper triangular matrix U and the
               multipliers necessary to construct a matrix L
               so that A=L*U.
    LDA    INTEGER
             the leading dimension of the array A.  LDA must be great-
             er than or equal to N.  (terminal error message IND=-1)
    N      INTEGER
             the order of the matrix A.  The first N elements of
             the array A are the elements of the first column of
             the  matrix A.  N must be greater than or equal to 1.
             (terminal error message IND=-2)
    V      REAL(N)
             on entry, the singly subscripted array(vector) of di-
               mension N which contains the right hand side B of a
               system of simultaneous linear equations A*X=B.
             on return, V contains the solution vector, X .
    ITASK  INTEGER
             If ITASK=1, the matrix A is factored and then the
```

```
                linear equation is solved.
              If ITASK .GT. 1, the equation is solved using the existing
                factored matrix A and IWORK.
              If ITASK .LT. 1, then terminal error message IND=-3 is
                printed.
    IND    INTEGER
              GT. 0  IND is a rough estimate of the number of digits
                      of accuracy in the solution, X.
              LT. 0  see error message corresponding to IND below.
    WORK   REAL(N)
              a singly subscripted array of dimension at least N.
    IWORK  INTEGER(N)
              a singly subscripted array of dimension at least N.

  Error Messages Printed ***

    IND=-1  terminal   N is greater than LDA.
    IND=-2  terminal   N is less than 1.
    IND=-3  terminal   ITASK is less than 1.
    IND=-4  terminal   The matrix A is computationally singular.
                         A solution has not been computed.
    IND=-10 warning    The solution has no apparent significance.
                         The solution may be inaccurate or the matrix
                         A may be poorly scaled.

                Note-  The above terminal(*fatal*) error messages are
                       designed to be handled by XERMSG in which
                       LEVEL=1 (recoverable) and IFLAG=2 .  LEVEL=0
                       for warning error messages from XERMSG.  Unless
                       the user provides otherwise, an error message
                       will be printed followed by an abort.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  R1MACH, SGECO, SGESL, XERMSG
***REVISION HISTORY  (YYMMDD)
   800317  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900510  Convert XERRWV calls to XERMSG calls.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SGEIR

```
     SUBROUTINE SGEIR (A, LDA, N, V, ITASK, IND, WORK, IWORK)
***BEGIN PROLOGUE  SGEIR
***PURPOSE  Solve a general system of linear equations.  Iterative
            refinement is used to obtain an error estimate.
***LIBRARY   SLATEC
***CATEGORY  D2A1
***TYPE      SINGLE PRECISION (SGEIR-S, CGEIR-C)
***KEYWORDS  COMPLEX LINEAR EQUATIONS, GENERAL MATRIX,
             GENERAL SYSTEM OF LINEAR EQUATIONS
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

    Subroutine SGEIR solves a general NxN system of single
    precision linear equations using LINPACK subroutines SGEFA and
    SGESL.  One pass of iterative refinement is used only to obtain
    an estimate of the accuracy.  That is, if A is an NxN real
    matrix and if X and B are real N-vectors, then SGEIR solves
    the equation

                           A*X=B.

    The matrix A is first factored into upper and lower tri-
    angular matrices U and L using partial pivoting.  These
    factors and the pivoting information are used to calculate
    the solution, X.  Then the residual vector is found and
    used to calculate an estimate of the relative error, IND.
    IND estimates the accuracy of the solution only when the
    input matrix and the right hand side are represented
    exactly in the computer and does not take into account
    any errors in the input data.

    If the equation A*X=B is to be solved for more than one vector
    B, the factoring of A does not need to be performed again and
    the option to solve only (ITASK .GT. 1) will be faster for
    the succeeding solutions.  In this case, the contents of A,
    LDA, N, WORK, and IWORK must not have been altered by the
    user following factorization (ITASK=1).  IND will not be
    changed by SGEIR in this case.

  Argument Description ***

    A      REAL(LDA,N)
             the doubly subscripted array with dimension (LDA,N)
             which contains the coefficient matrix.  A is not
             altered by the routine.
    LDA    INTEGER
             the leading dimension of the array A.  LDA must be great-
             er than or equal to N.  (terminal error message IND=-1)
    N      INTEGER
             the order of the matrix A.  The first N elements of
             the array A are the elements of the first column of
             matrix A.  N must be greater than or equal to 1.
             (terminal error message IND=-2)
    V      REAL(N)
             on entry, the singly subscripted array(vector) of di-
               mension N which contains the right hand side B of a
```

```
              system of simultaneous linear equations A*X=B.
            on return, V contains the solution vector, X .
   ITASK  INTEGER
            If ITASK=1, the matrix A is factored and then the
              linear equation is solved.
            If ITASK .GT. 1, the equation is solved using the existing
              factored matrix A (stored in WORK).
            If ITASK .LT. 1, then terminal error message IND=-3 is
              printed.
   IND    INTEGER
            GT. 0  IND is a rough estimate of the number of digits
                    of accuracy in the solution, X.  IND=75 means
                    that the solution vector X is zero.
            LT. 0  see error message corresponding to IND below.
   WORK   REAL(N*(N+1))
            a singly subscripted array of dimension at least N*(N+1).
   IWORK  INTEGER(N)
            a singly subscripted array of dimension at least N.

  Error Messages Printed ***

   IND=-1  terminal   N is greater than LDA.
   IND=-2  terminal   N is less than one.
   IND=-3  terminal   ITASK is less than one.
   IND=-4  terminal   The matrix A is computationally singular.
                        A solution has not been computed.
   IND=-10 warning    The solution has no apparent significance.
                        The solution may be inaccurate or the matrix
                        A may be poorly scaled.

              Note-  The above terminal(*fatal*) error messages are
                     designed to be handled by XERMSG in which
                     LEVEL=1 (recoverable) and IFLAG=2 .  LEVEL=0
                     for warning error messages from XERMSG.  Unless
                     the user provides otherwise, an error message
                     will be printed followed by an abort.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  R1MACH, SASUM, SCOPY, SDSDOT, SGEFA, SGESL, XERMSG
***REVISION HISTORY  (YYMMDD)
   800430  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900510  Convert XERRWV calls to XERMSG calls.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SGEMM

```
      SUBROUTINE SGEMM (TRANSA, TRANSB, M, N, ALPHA, A, LDA,
     +     B, LDB, BETA, C, LDC)
***BEGIN PROLOGUE  SGEMM
***PURPOSE  Multiply a real general matrix by a real general matrix.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B6
***TYPE      SINGLE PRECISION (SGEMM-S, DGEMM-D, CGEMM-C)
***KEYWORDS  LEVEL 3 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J., (ANL)
           Duff, I., (AERE)
           Du Croz, J., (NAG)
           Hammarling, S. (NAG)
***DESCRIPTION

  SGEMM  performs one of the matrix-matrix operations

     C := alpha*op( A )*op( B ) + beta*C,

  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 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.

  TRANSB - CHARACTER*1.
           On entry, TRANSB specifies the form of op( B ) to be used in
           the matrix multiplication as follows:

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

A       - REAL             array of DIMENSION ( LDA, ka ), where ka is
          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.

B       - REAL             array of DIMENSION ( LDB, kb ), where kb is
          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    - REAL            .
          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       - REAL             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.

***REFERENCES  Dongarra, J., Du Croz, J., Duff, I., and Hammarling, S.
                 A set of level 3 basic linear algebra subprograms.
                 ACM TOMS, Vol. 16, No. 1, pp. 1-17, March 1990.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   890208  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

SGEMV

```
      SUBROUTINE SGEMV (TRANS, M, N, ALPHA, A, LDA, X, INCX,
     +    BETA, Y, INCY)
***BEGIN PROLOGUE  SGEMV
***PURPOSE  Multiply a real vector by a real general matrix.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (SGEMV-S, DGEMV-D, CGEMV-C)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  SGEMV  performs one of the matrix-vector operations

     y := alpha*A*x + beta*y,   or   y := alpha*A'*x + beta*y,

  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 performed 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  - REAL            .
           On entry, ALPHA specifies the scalar alpha.
           Unchanged on exit.

  A      - REAL             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       - REAL             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 elements of
            X. INCX must not be zero.
            Unchanged on exit.

  BETA    - REAL            .
            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       - REAL             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 elements of
            Y. INCY must not be zero.
            Unchanged on exit.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

SGER

```
      SUBROUTINE SGER (M, N, ALPHA, X, INCX, Y, INCY, A, LDA)
***BEGIN PROLOGUE  SGER
***PURPOSE  Perform rank 1 update of a real general matrix.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (SGER-S)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  SGER   performs the rank 1 operation

     A := alpha*x*y' + A,

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

  X      - REAL             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 elements of
           X. INCX must not be zero.
           Unchanged on exit.

  Y      - REAL             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      - REAL             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.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

SGESL

```
      SUBROUTINE SGESL (A, LDA, N, IPVT, B, JOB)
***BEGIN PROLOGUE  SGESL
***PURPOSE  Solve the real system A*X=B or TRANS(A)*X=B using the
            factors of SGECO or SGEFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A1
***TYPE      SINGLE PRECISION (SGESL-S, DGESL-D, CGESL-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX, SOLVE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SGESL solves the real system
      A * X = B  or  TRANS(A) * X = B
      using the factors computed by SGECO or SGEFA.

      On Entry

         A       REAL(LDA, N)
                 the output from SGECO or SGEFA.

         LDA     INTEGER
                 the leading dimension of the array  A .

         N       INTEGER
                 the order of the matrix  A .

         IPVT    INTEGER(N)
                 the pivot vector from SGECO or SGEFA.

         B       REAL(N)
                 the right hand side vector.

         JOB     INTEGER
                 = 0         to solve  A*X = B ,
                 = nonzero   to solve  TRANS(A)*X = B  where
                             TRANS(A)  is the transpose.

      On Return

         B       the solution vector  X .

      Error Condition

         A division by zero will occur if the input factor contains a
         zero on the diagonal.  Technically, this indicates singularity,
         but it is often caused by improper arguments or improper
         setting of LDA .  It will not occur if the subroutines are
         called correctly and if SGECO has set RCOND .GT. 0.0
         or SGEFA has set INFO .EQ. 0 .

      To compute  INVERSE(A) * C  where  C  is a matrix
      with  P  columns
            CALL SGECO(A,LDA,N,IPVT,RCOND,Z)
            IF (RCOND is too small) GO TO ...
            DO 10 J = 1, P
               CALL SGESL(A,LDA,N,IPVT,C(1,J),0)
```

```
        10 CONTINUE

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SDOT
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SGLSS

```
      SUBROUTINE SGLSS (A, MDA, M, N, B, MDB, NB, RNORM, WORK, LW,
     +   IWORK, LIW, INFO)
***BEGIN PROLOGUE  SGLSS
***PURPOSE  Solve a linear least squares problems by performing a QR
            factorization of the matrix using Householder
            transformations.  Emphasis is put on detecting possible
            rank deficiency.
***LIBRARY   SLATEC
***CATEGORY  D9, D5
***TYPE      SINGLE PRECISION (SGLSS-S, DGLSS-D)
***KEYWORDS  LINEAR LEAST SQUARES, LQ FACTORIZATION, QR FACTORIZATION,
             UNDERDETERMINED LINEAR SYSTEMS
***AUTHOR  Manteuffel, T. A., (LANL)
***DESCRIPTION

      SGLSS solves both underdetermined and overdetermined
      LINEAR systems AX = B, where A is an M by N matrix
      and B is an M by NB matrix of right hand sides. If
      M.GE.N, the least squares solution is computed by
      decomposing the matrix A into the product of an
      orthogonal matrix Q and an upper triangular matrix
      R (QR factorization). If M.LT.N, the minimal
      length solution is computed by factoring the
      matrix A into the product of a lower triangular
      matrix L and an orthogonal matrix Q (LQ factor-
      ization). If the matrix A is determined to be rank
      deficient, that is the rank of A is less than
      MIN(M,N), then the minimal length least squares
      solution is computed.

      SGLSS assumes full machine precision in the data.
      If more control over the uncertainty in the data
      is desired, the codes LLSIA and ULSIA are
      recommended.

      SGLSS requires MDA*N + (MDB + 1)*NB + 5*MIN(M,N) dimensioned
      real space and M+N dimensioned integer space.


    ******************************************************************
    *                                                                *
    *         WARNING - All input arrays are changed on exit.        *
    *                                                                *
    ******************************************************************
      SUBROUTINE SGLSS(A,MDA,M,N,B,MDB,NB,RNORM,WORK,LW,IWORK,LIW,INFO)

      Input..

      A(,)          Linear coefficient matrix of AX=B, with MDA the
       MDA,M,N      actual first dimension of A in the calling program.
                    M is the row dimension (no. of EQUATIONS of the
                    problem) and N the col dimension (no. of UNKNOWNS).

      B(,)          Right hand side(s), with MDB the actual first
       MDB,NB       dimension of B in the calling program. NB is the
                    number of M by 1 right hand sides. Must have
```

```
                MDB.GE.MAX(M,N). If NB = 0, B is never accessed.

     RNORM()    Vector of length at least NB.  On input the contents
                of RNORM are unused.

     WORK()     A real work array dimensioned 5*MIN(M,N).

     LW         Actual dimension of WORK.

     IWORK()    Integer work array dimensioned at least N+M.

     LIW        Actual dimension of IWORK.

     INFO       A flag which provides for the efficient
                solution of subsequent problems involving the
                same A but different B.
                If INFO = 0 original call
                   INFO = 1 subsequent calls
                On subsequent calls, the user must supply A, INFO,
                LW, IWORK, LIW, and the first 2*MIN(M,N) locations
                of WORK as output by the original call to SGLSS.

     Output..

     A(,)       Contains the triangular part of the reduced matrix
                and the transformation information. It together with
                the first 2*MIN(M,N) elements of WORK (see below)
                completely specify the factorization of A.

     B(,)       Contains the N by NB solution matrix X.

     RNORM()    Contains the Euclidean length of the NB residual
                vectors  B(I)-AX(I), I=1,NB.

     WORK()     The first 2*MIN(M,N) locations of WORK contain value
                necessary to reproduce the factorization of A.

     IWORK()    The first M+N locations contain the order in
                which the rows and columns of A were used.
                If M.GE.N columns then rows. If M.LT.N rows
                then columns.

     INFO       Flag to indicate status of computation on completion
               -1   Parameter error(s)
                0 - Full rank
                N.GT.0 - Reduced rank  rank=MIN(M,N)-INFO

***REFERENCES  T. Manteuffel, An interval analysis approach to rank
                 determination in linear least squares problems,
                 Report SAND80-0655, Sandia Laboratories, June 1980.
***ROUTINES CALLED  LLSIA, ULSIA
***REVISION HISTORY  (YYMMDD)
   810801  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
```

```
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

SGMRES

```
      SUBROUTINE SGMRES(N, B, X, NELT, IA, JA, A, ISYM, MATVEC, MSOLVE,
     $     ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT, SB, SX,
     $     RGWK, LRGW, IGWK, LIGW, RWORK, IWORK)
***BEGIN PROLOGUE  SGMRES
***PURPOSE  Preconditioned GMRES Iterative Sparse Ax=b Solver.
            This routine uses the generalized minimum residual
            (GMRES) method with preconditioning to solve
            non-symmetric linear systems of the form: Ax = b.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SGMRES-S, DGMRES-D)
***KEYWORDS  GENERALIZED MINIMUM RESIDUAL, ITERATIVE PRECONDITION,
             NON-SYMMETRIC LINEAR SYSTEM, SLAP, SPARSE
***AUTHOR  Brown, Peter, (LLNL), pnbrown@llnl.gov
           Hindmarsh, Alan, (LLNL), alanh@llnl.gov
           Seager, Mark K., (LLNL), seager@llnl.gov
             Lawrence Livermore National Laboratory
             PO Box 808, L-60
             Livermore, CA 94550 (510) 423-3141
***DESCRIPTION

 *Usage:
      INTEGER   N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
      INTEGER   ITER, IERR, IUNIT, LRGW, IGWK(LIGW), LIGW
      INTEGER   IWORK(USER DEFINED)
      REAL      B(N), X(N), A(NELT), TOL, ERR, SB(N), SX(N)
      REAL      RGWK(LRGW), RWORK(USER DEFINED)
      EXTERNAL  MATVEC, MSOLVE

      CALL SGMRES(N, B, X, NELT, IA, JA, A, ISYM, MATVEC, MSOLVE,
     $     ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT, SB, SX,
     $     RGWK, LRGW, IGWK, LIGW, RWORK, IWORK)

 *Arguments:
 N      :IN       Integer.
         Order of the Matrix.
 B      :IN       Real B(N).
         Right-hand side vector.
 X      :INOUT    Real X(N).
         On input X is your initial guess for the solution vector.
         On output X is the final approximate solution.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :IN       Integer IA(NELT).
 JA     :IN       Integer JA(NELT).
 A      :IN       Real A(NELT).
         These arrays contain the matrix data structure for A.
         It could take any form.  See "Description", below,
         for more details.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 MATVEC :EXT      External.
         Name of a routine which performs the matrix vector multiply
```

Y = A*X given A and X. The name of the MATVEC routine must be declared external in the calling program. The calling sequence to MATVEC is:

```
    CALL MATVEC(N, X, Y, NELT, IA, JA, A, ISYM)
```

where N is the number of unknowns, Y is the product A*X upon return, X is an input vector, and NELT is the number of non-zeros in the SLAP IA, JA, A storage for the matrix A. ISYM is a flag which, if non-zero, denotes that A is symmetric and only the lower or upper triangle is stored.

MSOLVE :EXT External.

Name of the routine which solves a linear system Mz = r for z given r with the preconditioning matrix M (M is supplied via RWORK and IWORK arrays. The name of the MSOLVE routine must be declared external in the calling program. The calling sequence to MSOLVE is:

```
    CALL MSOLVE(N, R, Z, NELT, IA, JA, A, ISYM, RWORK, IWORK)
```

Where N is the number of unknowns, R is the right-hand side vector and Z is the solution upon return. NELT, IA, JA, A and ISYM are defined as above. RWORK is a real array that can be used to pass necessary preconditioning information and/or workspace to MSOLVE. IWORK is an integer work array for the same purpose as RWORK.

ITOL :IN Integer.

Flag to indicate the type of convergence criterion used.

ITOL=0 Means the iteration stops when the test described below on the residual RL is satisfied. This is the "Natural Stopping Criteria" for this routine. Other values of ITOL cause extra, otherwise unnecessary, computation per iteration and are therefore much less efficient. See ISSGMR (the stop test routine) for more information.

ITOL=1 Means the iteration stops when the first test described below on the residual RL is satisfied, and there is either right or no preconditioning being used.

ITOL=2 Implies that the user is using left preconditioning, and the second stopping criterion below is used.

ITOL=3 Means the iteration stops when the third test described below on Minv*Residual is satisfied, and there is either left or no preconditioning being used.

ITOL=11 is often useful for checking and comparing different routines. For this case, the user must supply the "exact" solution or a very accurate approximation (one with an error much less than TOL) through a common block,

```
    COMMON /SSLBLK/ SOLN( )
```

If ITOL=11, iteration stops when the 2-norm of the difference between the iterative approximation and the user-supplied solution divided by the 2-norm of the user-supplied solution is less than TOL. Note that this requires the user to set up the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine. The routine with this declaration should be loaded before the stop test so that the correct length is used by the loader. This procedure is not standard Fortran and may not work correctly on your system (although it has worked on every system the authors have tried). If ITOL is not 11

```
                then this common block is indeed standard Fortran.
TOL    :INOUT    Real.
        Convergence criterion, as described below.  If TOL is set
        to zero on input, then a default value of 500*(the smallest
        positive magnitude, machine epsilon) is used.
ITMAX  :DUMMY    Integer.
        Maximum number of iterations in most SLAP routines.  In
        this routine this does not make sense.  The maximum number
        of iterations here is given by ITMAX = MAXL*(NRMAX+1).
        See IGWK for definitions of MAXL and NRMAX.
ITER   :OUT      Integer.
        Number of iterations required to reach convergence, or
        ITMAX if convergence criterion could not be achieved in
        ITMAX iterations.
ERR    :OUT      Real.
        Error estimate of error in final approximate solution, as
        defined by ITOL.  Letting norm() denote the Euclidean
        norm, ERR is defined as follows..

        If ITOL=0, then ERR = norm(SB*(B-A*X(L)))/norm(SB*B),
                                for right or no preconditioning, and
                        ERR = norm(SB*(M-inverse)*(B-A*X(L)))/
                               norm(SB*(M-inverse)*B),
                                for left preconditioning.
        If ITOL=1, then ERR = norm(SB*(B-A*X(L)))/norm(SB*B),
                                since right or no preconditioning
                                being used.
        If ITOL=2, then ERR = norm(SB*(M-inverse)*(B-A*X(L)))/
                               norm(SB*(M-inverse)*B),
                                since left preconditioning is being
                                used.
        If ITOL=3, then ERR =  Max  |(Minv*(B-A*X(L)))(i)/x(i)|
                                i=1,n
        If ITOL=11, then ERR = norm(SB*(X(L)-SOLN))/norm(SB*SOLN).
IERR   :OUT      Integer.
        Return error flag.
              IERR = 0 => All went well.
              IERR = 1 => Insufficient storage allocated for
                          RGWK or IGWK.
              IERR = 2 => Routine SGMRES failed to reduce the norm
                          of the current residual on its last call,
                          and so the iteration has stalled.  In
                          this case, X equals the last computed
                          approximation.  The user must either
                          increase MAXL, or choose a different
                          initial guess.
              IERR =-1 => Insufficient length for RGWK array.
                          IGWK(6) contains the required minimum
                          length of the RGWK array.
              IERR =-2 => Illegal value of ITOL, or ITOL and JPRE
                          values are inconsistent.
        For IERR <= 2, RGWK(1) = RHOL, which is the norm on the
        left-hand-side of the relevant stopping test defined
        below associated with the residual for the current
        approximation X(L).
IUNIT  :IN       Integer.
        Unit number on which to write the error at each iteration,
        if this is desired for monitoring convergence.  If unit
        number is 0, no writing will occur.
SB     :IN       Real SB(N).
```

```
        Array of length N containing scale factors for the right
        hand side vector B.  If JSCAL.eq.0 (see below), SB need
        not be supplied.
SX     :IN       Real SX(N).
        Array of length N containing scale factors for the solution
        vector X.  If JSCAL.eq.0 (see below), SX need not be
        supplied.  SB and SX can be the same array in the calling
        program if desired.
RGWK   :INOUT    Real RGWK(LRGW).
        Real array used for workspace by SGMRES.
        On return, RGWK(1) = RHOL.  See IERR for definition of RHOL.
LRGW   :IN       Integer.
        Length of the real workspace, RGWK.
        LRGW >= 1 + N*(MAXL+6) + MAXL*(MAXL+3).
        See below for definition of MAXL.
        For the default values, RGWK has size at least 131 + 16*N.
IGWK   :INOUT    Integer IGWK(LIGW).
        The following IGWK parameters should be set by the user
        before calling this routine.
        IGWK(1) = MAXL.  Maximum dimension of Krylov subspace in
           which X - X0 is to be found (where, X0 is the initial
           guess).  The default value of MAXL is 10.
        IGWK(2) = KMP.  Maximum number of previous Krylov basis
           vectors to which each new basis vector is made orthogonal.
           The default value of KMP is MAXL.
        IGWK(3) = JSCAL.  Flag indicating whether the scaling
           arrays SB and SX are to be used.
           JSCAL = 0 => SB and SX are not used and the algorithm
              will perform as if all SB(I) = 1 and SX(I) = 1.
           JSCAL = 1 =>  Only SX is used, and the algorithm
              performs as if all SB(I) = 1.
           JSCAL = 2 =>  Only SB is used, and the algorithm
              performs as if all SX(I) = 1.
           JSCAL = 3 =>  Both SB and SX are used.
        IGWK(4) = JPRE.  Flag indicating whether preconditioning
           is being used.
           JPRE = 0  =>  There is no preconditioning.
           JPRE > 0  =>  There is preconditioning on the right
              only, and the solver will call routine MSOLVE.
           JPRE < 0  =>  There is preconditioning on the left
              only, and the solver will call routine MSOLVE.
        IGWK(5) = NRMAX.  Maximum number of restarts of the
           Krylov iteration.  The default value of NRMAX = 10.
           if IWORK(5) = -1,  then no restarts are performed (in
           this case, NRMAX is set to zero internally).
        The following IWORK parameters are diagnostic information
        made available to the user after this routine completes.
        IGWK(6) = MLWK.  Required minimum length of RGWK array.
        IGWK(7) = NMS.  The total number of calls to MSOLVE.
LIGW   :IN       Integer.
        Length of the integer workspace, IGWK.  LIGW >= 20.
RWORK  :WORK     Real RWORK(USER DEFINED).
        Real array that can be used for workspace in MSOLVE.
IWORK  :WORK     Integer IWORK(USER DEFINED).
        Integer array that can be used for workspace in MSOLVE.

*Description:
      SGMRES solves a linear system A*X = B rewritten in the form:

       (SB*A*(M-inverse)*(SX-inverse))*(SX*M*X) = SB*B,
```

with right preconditioning, or

```
(SB*(M-inverse)*A*(SX-inverse))*(SX*X) = SB*(M-inverse)*B,
```

with left preconditioning, where A is an N-by-N real matrix, X and B are N-vectors, SB and SX are diagonal scaling matrices, and M is a preconditioning matrix. It uses preconditioned Krylov subpace methods based on the generalized minimum residual method (GMRES). This routine optionally performs either the full orthogonalization version of the GMRES algorithm or an incomplete variant of it. Both versions use restarting of the linear iteration by default, although the user can disable this feature.

The GMRES algorithm generates a sequence of approximations X(L) to the true solution of the above linear system. The convergence criteria for stopping the iteration is based on the size of the scaled norm of the residual R(L) = B - A*X(L). The actual stopping test is either:

```
norm(SB*(B-A*X(L))) .le. TOL*norm(SB*B),
```

for right preconditioning, or

```
norm(SB*(M-inverse)*(B-A*X(L))) .le.
        TOL*norm(SB*(M-inverse)*B),
```

for left preconditioning, where norm() denotes the Euclidean norm, and TOL is a positive scalar less than one input by the user. If TOL equals zero when SGMRES is called, then a default value of 500*(the smallest positive magnitude, machine epsilon) is used. If the scaling arrays SB and SX are used, then ideally they should be chosen so that the vectors SX*X(or SX*M*X) and SB*B have all their components approximately equal to one in magnitude. If one wants to use the same scaling in X and B, then SB and SX can be the same array in the calling program.

The following is a list of the other routines and their functions used by SGMRES:

SPIGMR	Contains the main iteration loop for GMRES.
SORTH	Orthogonalizes a new vector against older basis vectors.
SHEQR	Computes a QR decomposition of a Hessenberg matrix.
SHELS	Solves a Hessenberg least-squares system, using QR factors.
SRLCAL	Computes the scaled residual RL.
SXLCAL	Computes the solution XL.
ISSGMR	User-replaceable stopping routine.

This routine does not care what matrix data structure is used for A and M. It simply calls the MATVEC and MSOLVE routines, with the arguments as described above. The user could write any type of structure and the appropriate MATVEC and MSOLVE routines. It is assumed that A is stored in the IA, JA, A arrays in some fashion and that M (or INV(M)) is stored in IWORK and RWORK in some fashion. The SLAP routines SSDCG and SSICCG are examples of this procedure.

Two examples of matrix data structures are the: 1) SLAP

Triad format and 2) SLAP Column format.

```
=================== S L A P Triad format ===================
```

This routine requires that the matrix A be stored in the SLAP Triad format. In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

```
=================== S L A P Column format ==================
```

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Cautions:

```
     This routine will attempt to write to the Fortran logical output
     unit IUNIT, if IUNIT .ne. 0.  Thus, the user must make sure that
     this logical unit is attached to a file or terminal before calling
     this routine with a non-zero value for IUNIT.  This routine does
     not check for the validity of a non-zero IUNIT unit number.

***REFERENCES  1. Peter N. Brown and A. C. Hindmarsh, Reduced Storage
                  Matrix Methods in Stiff ODE Systems, Lawrence Liver-
                  more National Laboratory Report UCRL-95088, Rev. 1,
                  Livermore, California, June 1987.
               2. Mark K. Seager, A SLAP for the Masses, in
                  G. F. Carey, Ed., Parallel Supercomputing: Methods,
                  Algorithms and Applications, Wiley, 1989, pp.135-155.
***ROUTINES CALLED  R1MACH, SCOPY, SNRM2, SPIGMR
***REVISION HISTORY  (YYMMDD)
   871001  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   891004  Added new reference.
   910411  Prologue converted to Version 4.0 format.  (BAB)
   910506  Corrected errors in C***ROUTINES CALLED list.  (FNF)
   920407  COMMON BLOCK renamed SSLBLK.  (WRB)
   920511  Added complete declaration section.  (WRB)
   920929  Corrected format of references.  (FNF)
   921019  Changed 500.0 to 500 to reduce SP/DP differences.  (FNF)
   921026  Added check for valid value of ITOL.  (FNF)
   END PROLOGUE
```

SGTSL

```
      SUBROUTINE SGTSL (N, C, D, E, B, INFO)
***BEGIN PROLOGUE  SGTSL
***PURPOSE  Solve a tridiagonal linear system.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A2A
***TYPE      SINGLE PRECISION (SGTSL-S, DGTSL-D, CGTSL-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX, SOLVE, TRIDIAGONAL
***AUTHOR  Dongarra, J., (ANL)
***DESCRIPTION

      SGTSL given a general tridiagonal matrix and a right hand
      side will find the solution.

      On Entry

         N       INTEGER
                 is the order of the tridiagonal matrix.

         C       REAL(N)
                 is the subdiagonal of the tridiagonal matrix.
                 C(2) through C(N) should contain the subdiagonal.
                 On output, C is destroyed.

         D       REAL(N)
                 is the diagonal of the tridiagonal matrix.
                 On output, D is destroyed.

         E       REAL(N)
                 is the superdiagonal of the tridiagonal matrix.
                 E(1) through E(N-1) should contain the superdiagonal.
                 On output, E is destroyed.

         B       REAL(N)
                 is the right hand side vector.

      On Return

         B       is the solution vector.

         INFO    INTEGER
                 = 0 normal value.
                 = K if the K-th element of the diagonal becomes
                     exactly zero.  The subroutine returns when
                     this is detected.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
```

```
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

SINDG

```
      FUNCTION SINDG (X)
***BEGIN PROLOGUE  SINDG
***PURPOSE  Compute the sine of an argument in degrees.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C4A
***TYPE      SINGLE PRECISION (SINDG-S, DSINDG-D)
***KEYWORDS  DEGREES, ELEMENTARY FUNCTIONS, FNLIB, SINE, TRIGONOMETRIC
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 SINDG(X) evaluates the single precision sine of X where
 X is in degrees.

***REFERENCES  (NONE)
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   770601  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890531  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   END PROLOGUE
```

SINQB

```
      SUBROUTINE SINQB (N, X, WSAVE)
***BEGIN PROLOGUE  SINQB
***PURPOSE  Compute the unnormalized inverse of SINQF.
***LIBRARY   SLATEC (FFTPACK)
***CATEGORY  J1A3
***TYPE      SINGLE PRECISION (SINQB-S)
***KEYWORDS  FFTPACK, FOURIER TRANSFORM
***AUTHOR  Swarztrauber, P. N., (NCAR)
***DESCRIPTION

  Subroutine SINQB computes the fast Fourier transform of quarter
  wave data.  That is, SINQB computes a sequence from its
  representation in terms of a sine series with odd wave numbers.
  the transform is defined below at output parameter X.

  SINQF is the unnormalized inverse of SINQB since a call of SINQB
  followed by a call of SINQF will multiply the input sequence X
  by 4*N.

  The array WSAVE which is used by subroutine SINQB must be
  initialized by calling subroutine SINQI(N,WSAVE).

  Input Parameters

  N       the length of the array X to be transformed.  The method
          is most efficient when N is a product of small primes.

  X       an array which contains the sequence to be transformed

  WSAVE   a work array which must be dimensioned at least 3*N+15
          in the program that calls SINQB.  The WSAVE array must be
          initialized by calling subroutine SINQI(N,WSAVE), and a
          different WSAVE array must be used for each different
          value of N.  This initialization does not have to be
          repeated so long as N remains unchanged.  Thus subsequent
          transforms can be obtained faster than the first.

  Output Parameters

  X       For I=1,...,N

               X(I)= the sum from K=1 to K=N of

                 4*X(K)*SIN((2*K-1)*I*PI/(2*N))

               a call of SINQB followed by a call of
               SINQF will multiply the sequence X by 4*N.
               Therefore SINQF is the unnormalized inverse
               of SINQB.

  WSAVE   contains initialization calculations which must not
          be destroyed between calls of SINQB or SINQF.

***REFERENCES  P. N. Swarztrauber, Vectorizing the FFTs, in Parallel
                 Computations (G. Rodrigue, ed.), Academic Press,
                 1982, pp. 51-83.
```

```
***ROUTINES CALLED  COSQB
***REVISION HISTORY  (YYMMDD)
   790601  DATE WRITTEN
   830401  Modified to use SLATEC library source file format.
   860115  Modified by Ron Boisvert to adhere to Fortran 77 by
           changing dummy array size declarations (1) to (*).
   861211  REVISION DATE from Version 3.2
   881128  Modified by Dick Valent to meet prologue standards.
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SINQF

```
      SUBROUTINE SINQF (N, X, WSAVE)
***BEGIN PROLOGUE  SINQF
***PURPOSE  Compute the forward sine transform with odd wave numbers.
***LIBRARY   SLATEC (FFTPACK)
***CATEGORY  J1A3
***TYPE      SINGLE PRECISION (SINQF-S)
***KEYWORDS  FFTPACK, FOURIER TRANSFORM
***AUTHOR  Swarztrauber, P. N., (NCAR)
***DESCRIPTION

  Subroutine SINQF computes the fast Fourier transform of quarter
  wave data.  That is, SINQF computes the coefficients in a sine
  series representation with only odd wave numbers.  The transform
  is defined below at output parameter X.

  SINQB is the unnormalized inverse of SINQF since a call of SINQF
  followed by a call of SINQB will multiply the input sequence X
  by 4*N.

  The array WSAVE which is used by subroutine SINQF must be
  initialized by calling subroutine SINQI(N,WSAVE).

  Input Parameters

  N       the length of the array X to be transformed.  The method
          is most efficient when N is a product of small primes.

  X       an array which contains the sequence to be transformed

  WSAVE   a work array which must be dimensioned at least 3*N+15
          in the program that calls SINQF.  The WSAVE array must be
          initialized by calling subroutine SINQI(N,WSAVE), and a
          different WSAVE array must be used for each different
          value of N.  This initialization does not have to be
          repeated so long as N remains unchanged.  Thus subsequent
          transforms can be obtained faster than the first.

  Output Parameters

  X       For I=1,...,N

               X(I) = (-1)**(I-1)*X(N)

                  + the sum from K=1 to K=N-1 of

                  2*X(K)*SIN((2*I-1)*K*PI/(2*N))

               A call of SINQF followed by a call of
               SINQB will multiply the sequence X by 4*N.
               Therefore SINQB is the unnormalized inverse
               of SINQF.

  WSAVE   contains initialization calculations which must not
          be destroyed between calls of SINQF or SINQB.

***REFERENCES  P. N. Swarztrauber, Vectorizing the FFTs, in Parallel
```

```
                 Computations (G. Rodrigue, ed.), Academic Press,
                 1982, pp. 51-83.
***ROUTINES CALLED  COSQF
***REVISION HISTORY  (YYMMDD)
   790601  DATE WRITTEN
   830401  Modified to use SLATEC library source file format.
   860115  Modified by Ron Boisvert to adhere to Fortran 77 by
           changing dummy array size declarations (1) to (*)
   861211  REVISION DATE from Version 3.2
   881128  Modified by Dick Valent to meet prologue standards.
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SINQI

```
      SUBROUTINE SINQI (N, WSAVE)
***BEGIN PROLOGUE  SINQI
***PURPOSE  Initialize a work array for SINQF and SINQB.
***LIBRARY   SLATEC (FFTPACK)
***CATEGORY  J1A3
***TYPE      SINGLE PRECISION (SINQI-S)
***KEYWORDS  FFTPACK, FOURIER TRANSFORM
***AUTHOR  Swarztrauber, P. N., (NCAR)
***DESCRIPTION

  Subroutine SINQI initializes the array WSAVE which is used in
  both SINQF and SINQB.  The prime factorization of N together with
  a tabulation of the trigonometric functions are computed and
  stored in WSAVE.

  Input Parameter

  N       the length of the sequence to be transformed.  The method
          is most efficient when N is a product of small primes.

  Output Parameter

  WSAVE   a work array which must be dimensioned at least 3*N+15.
          The same work array can be used for both SINQF and SINQB
          as long as N remains unchanged.  Different WSAVE arrays
          are required for different values of N.  The contents of
          WSAVE must not be changed between calls of SINQF or SINQB.

***REFERENCES  P. N. Swarztrauber, Vectorizing the FFTs, in Parallel
                Computations (G. Rodrigue, ed.), Academic Press,
                1982, pp. 51-83.
***ROUTINES CALLED  COSQI
***REVISION HISTORY  (YYMMDD)
   790601  DATE WRITTEN
   830401  Modified to use SLATEC library source file format.
   860115  Modified by Ron Boisvert to adhere to Fortran 77 by
           changing dummy array size declarations (1) to (*)
   861211  REVISION DATE from Version 3.2
   881128  Modified by Dick Valent to meet prologue standards.
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SINT

```
      SUBROUTINE SINT (N, X, WSAVE)
***BEGIN PROLOGUE  SINT
***PURPOSE  Compute the sine transform of a real, odd sequence.
***LIBRARY   SLATEC (FFTPACK)
***CATEGORY  J1A3
***TYPE      SINGLE PRECISION (SINT-S)
***KEYWORDS  FFTPACK, FOURIER TRANSFORM
***AUTHOR  Swarztrauber, P. N., (NCAR)
***DESCRIPTION

  Subroutine SINT computes the discrete Fourier sine transform
  of an odd sequence X(I).  The transform is defined below at
  output parameter X.

  SINT is the unnormalized inverse of itself since a call of SINT
  followed by another call of SINT will multiply the input sequence
  X by 2*(N+1).

  The array WSAVE which is used by subroutine SINT must be
  initialized by calling subroutine SINTI(N,WSAVE).

  Input Parameters

  N       the length of the sequence to be transformed.  The method
          is most efficient when N+1 is the product of small primes.

  X       an array which contains the sequence to be transformed


  WSAVE   a work array with dimension at least INT(3.5*N+16)
          in the program that calls SINT.  The WSAVE array must be
          initialized by calling subroutine SINTI(N,WSAVE), and a
          different WSAVE array must be used for each different
          value of N.  This initialization does not have to be
          repeated so long as N remains unchanged.  Thus subsequent
          transforms can be obtained faster than the first.

  Output Parameters

  X       For I=1,...,N

               X(I)= the sum from K=1 to K=N

                    2*X(K)*SIN(K*I*PI/(N+1))

               A call of SINT followed by another call of
               SINT will multiply the sequence X by 2*(N+1).
               Hence SINT is the unnormalized inverse
               of itself.

  WSAVE   contains initialization calculations which must not be
          destroyed between calls of SINT.

***REFERENCES  P. N. Swarztrauber, Vectorizing the FFTs, in Parallel
                 Computations (G. Rodrigue, ed.), Academic Press,
                 1982, pp. 51-83.
```

```
***ROUTINES CALLED  RFFTF
***REVISION HISTORY  (YYMMDD)
   790601  DATE WRITTEN
   830401  Modified to use SLATEC library source file format.
   860115  Modified by Ron Boisvert to adhere to Fortran 77 by
           (a) changing dummy array size declarations (1) to (*),
           (b) changing definition of variable SQRT3 by using
               FORTRAN intrinsic function SQRT instead of a DATA
               statement.
   881128  Modified by Dick Valent to meet prologue standards.
   891009  Removed unreferenced statement label.  (WRB)
   891009  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SINTI

```
      SUBROUTINE SINTI (N, WSAVE)
***BEGIN PROLOGUE  SINTI
***PURPOSE  Initialize a work array for SINT.
***LIBRARY   SLATEC (FFTPACK)
***CATEGORY  J1A3
***TYPE      SINGLE PRECISION (SINTI-S)
***KEYWORDS  FFTPACK, FOURIER TRANSFORM
***AUTHOR  Swarztrauber, P. N., (NCAR)
***DESCRIPTION

  Subroutine SINTI initializes the array WSAVE which is used in
  subroutine SINT.  The prime factorization of N together with
  a tabulation of the trigonometric functions are computed and
  stored in WSAVE.

  Input Parameter

  N       the length of the sequence to be transformed.  The method
          is most efficient when N+1 is a product of small primes.

  Output Parameter

  WSAVE   a work array with at least INT(3.5*N+16) locations.
          Different WSAVE arrays are required for different values
          of N.  The contents of WSAVE must not be changed between
          calls of SINT.

***REFERENCES  P. N. Swarztrauber, Vectorizing the FFTs, in Parallel
                 Computations (G. Rodrigue, ed.), Academic Press,
                 1982, pp. 51-83.
***ROUTINES CALLED  RFFTI
***REVISION HISTORY  (YYMMDD)
   790601  DATE WRITTEN
   830401  Modified to use SLATEC library source file format.
   860115  Modified by Ron Boisvert to adhere to Fortran 77 by
           (a) changing dummy array size declarations (1) to (*),
           (b) changing references to intrinsic function FLOAT
               to REAL, and
           (c) changing definition of variable PI by using
               FORTRAN intrinsic function ATAN instead of a DATA
               statement.
   881128  Modified by Dick Valent to meet prologue standards.
   890531  Changed all specific intrinsics to generic.  (WRB)
   890531  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SINTRP

```
      SUBROUTINE SINTRP (X, Y, XOUT, YOUT, YPOUT, NEQN, KOLD, PHI, IVC,
     +   IV, KGI, GI, ALPHA, OG, OW, OX, OY)
***BEGIN PROLOGUE  SINTRP
***PURPOSE  Approximate the solution at XOUT by evaluating the
            polynomial computed in STEPS at XOUT.  Must be used in
            conjunction with STEPS.
***LIBRARY   SLATEC (DEPAC)
***CATEGORY  I1A1B
***TYPE      SINGLE PRECISION (SINTRP-S, DINTP-D)
***KEYWORDS  ADAMS METHOD, DEPAC, INITIAL VALUE PROBLEMS, ODE,
             ORDINARY DIFFERENTIAL EQUATIONS, PREDICTOR-CORRECTOR,
             SMOOTH INTERPOLANT
***AUTHOR  Watts, H. A., (SNLA)
***DESCRIPTION

   The methods in subroutine  STEPS  approximate the solution near  X
   by a polynomial.  Subroutine  SINTRP  approximates the solution at
   XOUT  by evaluating the polynomial there.  Information defining this
   polynomial is passed from  STEPS  so  SINTRP  cannot be used alone.

   Subroutine STEPS is completely explained and documented in the text,
   "Computer Solution of Ordinary Differential Equations, the Initial
   Value Problem"  by L. F. Shampine and M. K. Gordon.

   Input to SINTRP --

   The user provides storage in the calling program for the arrays in
   the call list
      DIMENSION Y(NEQN),YOUT(NEQN),YPOUT(NEQN),PHI(NEQN,16),OY(NEQN)
                AND ALPHA(12),OG(13),OW(12),GI(11),IV(10)
   and defines
      XOUT -- point at which solution is desired.
   The remaining parameters are defined in  STEPS  and passed to
   SINTRP  from that subroutine

   Output from  SINTRP --

      YOUT(*) -- solution at  XOUT
      YPOUT(*) -- derivative of solution at  XOUT
   The remaining parameters are returned unaltered from their input
   values.  Integration with  STEPS  may be continued.

***REFERENCES  H. A. Watts, A smoother interpolant for DE/STEP, INTRP
                II, Report SAND84-0293, Sandia Laboratories, 1984.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   840201  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SIR

```
      SUBROUTINE SIR(N, B, X, NELT, IA, JA, A, ISYM, MATVEC, MSOLVE, ITOL,
     $     TOL, ITMAX, ITER, ERR, IERR, IUNIT, R, Z, DZ, RWORK, IWORK)
***BEGIN PROLOGUE  SIR
***PURPOSE  Preconditioned Iterative Refinement Sparse Ax = b Solver.
            Routine to solve a general linear system  Ax = b  using
            iterative refinement with a matrix splitting.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SIR-S, DIR-D)
***KEYWORDS  ITERATIVE PRECONDITION, LINEAR SYSTEM, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
      INTEGER  N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
      INTEGER  ITER, IERR, IUNIT, IWORK(USER DEFINED)
      REAL     B(N), X(N), A(NELT), TOL, ERR, R(N), Z(N), DZ(N),
      REAL     RWORK(USER DEFINED)
      EXTERNAL MATVEC, MSOLVE

      CALL SIR(N, B, X, NELT, IA, JA, A, ISYM, MATVEC, MSOLVE, ITOL,
     $     TOL, ITMAX, ITER, ERR, IERR, IUNIT, R, Z, DZ, RWORK, IWORK)

 *Arguments:
 N      :IN       Integer.
         Order of the Matrix.
 B      :IN       Real B(N).
         Right-hand side vector.
 X      :INOUT    Real X(N).
         On input X is your initial guess for solution vector.
         On output X is the final approximate solution.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :IN       Integer IA(NELT).
 JA     :IN       Integer JA(NELT).
 A      :IN       Real A(NELT).
         These arrays contain the matrix data structure for A.
         It could take any form.  See "Description", below,
         for more details.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 MATVEC :EXT      External.
         Name of a routine which performs the matrix vector multiply
         Y = A*X given A and X.  The name of the MATVEC routine must
         be declared external in the calling program.  The calling
         sequence to MATVEC is:
             CALL MATVEC( N, X, Y, NELT, IA, JA, A, ISYM )
         Where N is the number of unknowns, Y is the product A*X
```

```
        upon return, X is an input vector, NELT is the number of
        non-zeros in the SLAP IA, JA, A storage for the matrix A.
        ISYM is a flag which, if non-zero, denotes that A is
        symmetric and only the lower or upper triangle is stored.
MSOLVE :EXT      External.
        Name of a routine which solves a linear system MZ = R for
        Z given R with the preconditioning matrix M (M is supplied via
        RWORK and IWORK arrays).  The name of the MSOLVE routine must
        be declared external in the calling program.  The calling
        sequence to MSOLVE is:
            CALL MSOLVE(N, R, Z, NELT, IA, JA, A, ISYM, RWORK, IWORK)
        Where N is the number of unknowns, R is the right-hand side
        vector and Z is the solution upon return.  NELT, IA, JA, A and
        ISYM are defined as above.  RWORK is a real array that can
        be used to pass necessary preconditioning information and/or
        workspace to MSOLVE.  IWORK is an integer work array for
        the same purpose as RWORK.
ITOL   :IN       Integer.
        Flag to indicate type of convergence criterion.
        If ITOL=1, iteration stops when the 2-norm of the residual
        divided by the 2-norm of the right-hand side is less than TOL.
        If ITOL=2, iteration stops when the 2-norm of M-inv times the
        residual divided by the 2-norm of M-inv times the right hand
        side is less than TOL, where M-inv is the inverse of the
        diagonal of A.
        ITOL=11 is often useful for checking and comparing different
        routines.  For this case, the user must supply the "exact"
        solution or a very accurate approximation (one with an error
        much less than TOL) through a common block,
            COMMON /SSLBLK/ SOLN( )
        If ITOL=11, iteration stops when the 2-norm of the difference
        between the iterative approximation and the user-supplied
        solution divided by the 2-norm of the user-supplied solution
        is less than TOL.  Note that this requires the user to set up
        the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine.
        The routine with this declaration should be loaded before the
        stop test so that the correct length is used by the loader.
        This procedure is not standard Fortran and may not work
        correctly on your system (although it has worked on every
        system the authors have tried).  If ITOL is not 11 then this
        common block is indeed standard Fortran.
TOL    :INOUT    Real.
        Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX  :IN       Integer.
        Maximum number of iterations.
ITER   :OUT      Integer.
        Number of iterations required to reach convergence, or
        ITMAX+1 if convergence criterion could not be achieved in
        ITMAX iterations.
ERR    :OUT      Real.
        Error estimate of error in final approximate solution, as
        defined by ITOL.
IERR   :OUT      Integer.
        Return error flag.
          IERR = 0 => All went well.
          IERR = 1 => Insufficient space allocated for WORK or IWORK.
          IERR = 2 => Method failed to converge in ITMAX steps.
          IERR = 3 => Error in user input.
                      Check input values of N, ITOL.
          IERR = 4 => User error tolerance set too tight.
```

```
                    Reset to 500*R1MACH(3).  Iteration proceeded.
          IERR = 5 => Preconditioning matrix, M, is not positive
                      definite.  (r,z) < 0.
          IERR = 6 => Matrix A is not positive definite.  (p,Ap) < 0.
IUNIT  :IN       Integer.
        Unit number on which to write the error at each iteration,
        if this is desired for monitoring convergence.  If unit
        number is 0, no writing will occur.
R      :WORK     Real R(N).
Z      :WORK     Real Z(N).
DZ     :WORK     Real DZ(N).
        Real arrays used for workspace.
RWORK  :WORK     Real RWORK(USER DEFINED).
        Real array that can be used by  MSOLVE.
IWORK  :WORK     Integer IWORK(USER DEFINED).
        Integer array that can be used by  MSOLVE.
```

*Description:

The basic algorithm for iterative refinement (also known as iterative improvement) is:

$$X^{n+1} = X^{n} + M^{-1}(B - AX^{n}).$$

If $M = A^{-1}$ then this is the standard iterative refinement algorithm and the "subtraction" in the residual calculation should be done in double precision (which it is not in this routine).
If M = DIAG(A), the diagonal of A, then iterative refinement is known as Jacobi's method. The SLAP routine SSJAC implements this iterative strategy.
If M = L, the lower triangle of A, then iterative refinement is known as Gauss-Seidel. The SLAP routine SSGS implements this iterative strategy.

This routine does not care what matrix data structure is used for A and M. It simply calls the MATVEC and MSOLVE routines, with the arguments as described above. The user could write any type of structure and the appropriate MATVEC and MSOLVE routines. It is assumed that A is stored in the IA, JA, A arrays in some fashion and that M (or INV(M)) is stored in IWORK and RWORK) in some fashion. The SLAP routines SSJAC and SSGS are examples of this procedure.

Two examples of matrix data structures are the: 1) SLAP Triad format and 2) SLAP Column format.

========== S L A P Triad format ==========

In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear

systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Examples:
See the SLAP routines SSJAC, SSGS

*Cautions:
This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO SSJAC, SSGS

***REFERENCES 1. Gene Golub and Charles Van Loan, Matrix Computations, Johns Hopkins University Press, Baltimore, Maryland, 1983.
2. Mark K. Seager, A SLAP for the Masses, in

```
                  G. F. Carey, Ed., Parallel Supercomputing: Methods,
                  Algorithms and Applications, Wiley, 1989, pp.135-155.
***ROUTINES CALLED  ISSIR, R1MACH
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890921  Removed TeX from comments.  (FNF)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   891004  Added new reference.
   910411  Prologue converted to Version 4.0 format.  (BAB)
   910502  Removed MATVEC and MSOLVE from ROUTINES CALLED list.  (FNF)
   920407  COMMON BLOCK renamed SSLBLK.  (WRB)
   920511  Added complete declaration section.  (WRB)
   920929  Corrected format of references.  (FNF)
   921019  Changed 500.0 to 500 to reduce SP/DP differences.  (FNF)
   END PROLOGUE
```

SLLTI2

```
      SUBROUTINE SLLTI2 (N, B, X, NEL, IEL, JEL, EL, DINV)
***BEGIN PROLOGUE  SLLTI2
***PURPOSE  SLAP Backsolve routine for LDL' Factorization.
            Routine to solve a system of the form  L*D*L' X = B,
            where L is a unit lower triangular matrix and D is a
            diagonal matrix and ' means transpose.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2E
***TYPE      SINGLE PRECISION (SLLTI2-S, DLLTI2-D)
***KEYWORDS  INCOMPLETE FACTORIZATION, ITERATIVE PRECONDITION, SLAP,
             SPARSE, SYMMETRIC LINEAR SYSTEM SOLVE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NEL, IEL(NEL), JEL(NEL)
     REAL    B(N), X(N), EL(NEL), DINV(N)

     CALL SLLTI2( N, B, X, NEL, IEL, JEL, EL, DINV )

 *Arguments:
 N      :IN       Integer
         Order of the Matrix.
 B      :IN       Real B(N).
         Right hand side vector.
 X      :OUT      Real X(N).
         Solution to L*D*L' x = b.
 NEL    :IN       Integer.
         Number of non-zeros in the EL array.
 IEL    :IN       Integer IEL(NEL).
 JEL    :IN       Integer JEL(NEL).
 EL     :IN       Real     EL(NEL).
         IEL, JEL, EL contain the unit lower triangular factor   of
         the incomplete decomposition   of the A  matrix  stored in
         SLAP Row format.   The diagonal of ones *IS* stored.  This
         structure can be set  up  by  the SS2LT routine.  See  the
         "Description", below for more details about the  SLAP  Row
         format.
 DINV   :IN       Real DINV(N).
         Inverse of the diagonal matrix D.

 *Description:
       This routine is supplied with  the SLAP package as a routine
       to perform the MSOLVE operation in the SCG iteration routine
       for  the driver  routine SSICCG.   It must be called via the
       SLAP  MSOLVE calling sequence  convention  interface routine
       SSLLI.
         **** THIS ROUTINE ITSELF DOES NOT CONFORM TO THE ****
               **** SLAP MSOLVE CALLING CONVENTION ****

       IEL, JEL, EL should contain the unit lower triangular factor
```

of the incomplete decomposition of the A matrix stored in SLAP Row format. This IC factorization can be computed by the SSICS routine. The diagonal (which is all one's) is stored.

```
==================== S L A P Row format ====================
```

This routine requires that the matrix A be stored in the SLAP Row format. In this format the non-zeros are stored counting across rows (except for the diagonal entry, which must appear first in each "row") and are stored in the real array A. In other words, for each row in the matrix put the diagonal entry in A. Then put in the other non-zero elements going across the row (except the diagonal) in order. The JA array holds the column index for each non-zero. The IA array holds the offsets into the JA, A arrays for the beginning of each row. That is, JA(IA(IROW)), A(IA(IROW)) points to the beginning of the IROW-th row in JA and A. JA(IA(IROW+1)-1), A(IA(IROW+1)-1) points to the end of the IROW-th row. Note that we always have IA(N+1) = NELT+1, where N is the number of rows in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Row storage format for a 5x5 Matrix (in the A and JA arrays '|' denotes the end of a row):

```
    5x5 Matrix         SLAP Row format for 5x5 matrix on left.
                        1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 12 15 | 22 21 | 33 35 | 44 | 55 51 53
|21 22  0  0  0|  JA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  IA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

With the SLAP Row format the "inner loop" of this routine should vectorize on machines with hardware support for vector gather/scatter operations. Your compiler may require a compiler directive to convince it that there are no implicit vector dependencies. Compiler directives for the Alliant FX/Fortran and CRI CFT/CFT77 compilers are supplied with the standard SLAP distribution.

```
***SEE ALSO  SSICCG, SSICS
***REFERENCES  (NONE)
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920511  Added complete declaration section.  (WRB)
   921113  Corrected C***CATEGORY line.  (FNF)
   930701  Updated CATEGORY section.  (FNF, WRB)
   END PROLOGUE
```

SLPDOC

```
      SUBROUTINE SLPDOC
***BEGIN PROLOGUE  SLPDOC
***PURPOSE  Sparse Linear Algebra Package Version 2.0.2 Documentation.
            Routines to solve large sparse symmetric and nonsymmetric
            positive definite linear systems, Ax = b, using precondi-
            tioned iterative methods.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4, Z
***TYPE      SINGLE PRECISION (SLPDOC-S, DLPDOC-D)
***KEYWORDS  BICONJUGATE GRADIENT SQUARED, DOCUMENTATION,
             GENERALIZED MINIMUM RESIDUAL, ITERATIVE IMPROVEMENT,
             NORMAL EQUATIONS, ORTHOMIN,
             PRECONDITIONED CONJUGATE GRADIENT, SLAP,
             SPARSE ITERATIVE METHODS
***AUTHOR  Seager, Mark. K., (LLNL)
             User Systems Division
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550
             (FTS) 543-3141, (510) 423-3141
             seager@llnl.gov
***DESCRIPTION
                                 The
                   Sparse Linear Algebra Package

               @@@@  @            @@    @@@@
              @       @ @           @   @   @       @
              @         @         @     @  @       @
               @@@@  @         @       @ @@@@
                       @ @         @@@@@ @
              @        @ @         @       @ @
               @@@@  @@@@@ @       @ @

      @       @                            @@@@        @@@
      @       @                           @       @      @    @
      @       @  @@@@  @ @                    @     @    @  @
      @       @ @       @ @  @              @@@      @   @   @
       @     @  @@@@@ @                @           @  @    @
        @   @   @         @              @           @@  @    @
          @@     @@@@  @                @@@@@ @@   @@@
```

```
==================================================================
========================== Introduction ==========================
==================================================================
```

This package was originally derived from a set of iterative routines written by Anne Greenbaum, as announced in "Routines for Solving Large Sparse Linear Systems", Tentacle, Lawrence Livermore National Laboratory, Livermore Computing Center (January 1986), pp 15-21.

This document contains the specifications for the SLAP Version 2.0 package, a Fortran 77 package for the solution of large sparse linear systems, Ax = b, via preconditioned iterative methods. Included in this package are "core" routines to do Iterative Refinement (Jacobi's method), Conjugate Gradient,

Conjugate Gradient on the normal equations, AA'y = b, (where x = A'y and A' denotes the transpose of A), BiConjugate Gradient, BiConjugate Gradient Squared, Orthomin and Generalized Minimum Residual Iteration. These "core" routines do not require a "fixed" data structure for storing the matrix A and the preconditioning matrix M. The user is free to choose any structure that facilitates efficient solution of the problem at hand. The drawback to this approach is that the user must also supply at least two routines (MATVEC and MSOLVE, say). MATVEC must calculate, y = Ax, given x and the user's data structure for A. MSOLVE must solve, r = Mz, for z (*NOT* r) given r and the user's data structure for M (or its inverse). The user should choose M so that inv(M)*A is approximately the identity and the solution step r = Mz is "easy" to solve. For some of the "core" routines (Orthomin, BiConjugate Gradient and Conjugate Gradient on the normal equations) the user must also supply a matrix transpose times vector routine (MTTVEC, say) and (possibly, depending on the "core" method) a routine that solves the transpose of the preconditioning step (MTSOLV, say). Specifically, MTTVEC is a routine which calculates y = A'x, given x and the user's data structure for A (A' is the transpose of A). MTSOLV is a routine which solves the system r = M'z for z given r and the user's data structure for M.

This process of writing the matrix vector operations can be time consuming and error prone. To alleviate these problems we have written drivers for the "core" methods that assume the user supplies one of two specific data structures (SLAP Triad and SLAP Column format), see below. Utilizing these data structures we have augmented each "core" method with two preconditioners: Diagonal Scaling and Incomplete Factorization. Diagonal scaling is easy to implement, vectorizes very well and for problems that are not too ill-conditioned reduces the number of iterations enough to warrant its use. On the other hand, an Incomplete factorization (Incomplete Cholesky for symmetric systems and Incomplete LU for nonsymmetric systems) may take much longer to calculate, but it reduces the iteration count (for most problems) significantly. Our implementations of IC and ILU vectorize for machines with hardware gather scatter, but the vector lengths can be quite short if the number of non-zeros in a column is not large.

==========
========== Supplied Data Structures ==========
==========
The following describes the data structures supplied with the package: SLAP Triad and Column formats.

========== S L A P Triad format ==========

In the SLAP Triad format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). If the matrix is symmetric then one need only store the lower triangle (including the diagonal) and NELT would be the corresponding number of non-zeros stored. For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to

generate. On the other hand, it is not very efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a nonsymmetric 5x5 Matrix. NELT=11. Recall that the entries may appear in any order.

```
 5x5 Matrix       SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

In the SLAP Column format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix first put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) are the first elements of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) are the last elements of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix. If the matrix is symmetric one need only store the lower triangle (including the diagonal) and NELT would be the corresponding number of non-zeros stored.

Here is an example of the SLAP Column storage format for a nonsymmetric 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
   5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

==========
========== Which Method To Use ==========
==========

BACKGROUND

In solving a large sparse linear system Ax = b using an iterative method, it is not necessary to actually store the matrix A. Rather, what is needed is a procedure for multiplying the matrix A times a given vector y to obtain the matrix-vector product, Ay. SLAP has been written to take advantage of this fact. The higher level routines in the package require storage only of the non-zero elements of A (and their positions), and even this can be

avoided, if the user writes his own subroutine for multiplying the matrix times a vector and calls the lower-level iterative routines in the package.

If the matrix A is ill-conditioned, then most iterative methods will be slow to converge (if they converge at all!). To improve the convergence rate, one may use a "matrix splitting," or, "preconditioning matrix," say, M. It is then necessary to solve, at each iteration, a linear system with coefficient matrix M. A good preconditioner M should have two properties: (1) M should "approximate" A, in the sense that the matrix inv(M)*A (or some variant thereof) is better conditioned than the original matrix A; and (2) linear systems with coefficient matrix M should be much easier to solve than the original system with coefficient matrix A. Preconditioning routines in the SLAP package are separate from the iterative routines, so that any of the preconditioners provided in the package, or one that the user codes himself, can be used with any of the iterative routines.

CHOICE OF PRECONDITIONER

If you willing to live with either the SLAP Triad or Column matrix data structure you can then choose one of two types of preconditioners to use: diagonal scaling or incomplete factorization. To choose between these two methods requires knowing something about the computer you're going to run these codes on and how well incomplete factorization approximates the inverse of your matrix.

Let us suppose you have a scalar machine. Then, unless the incomplete factorization is very, very poor this is *GENERALLY* the method to choose. It will reduce the number of iterations significantly and is not all that expensive to compute. So if you have just one linear system to solve and "just want to get the job done" then try incomplete factorization first. If you are thinking of integrating some SLAP iterative method into your favorite "production code" then try incomplete factorization first, but also check to see that diagonal scaling is indeed slower for a large sample of test problems.

Let us now suppose you have a vector computer with hardware gather/scatter support (Cray X-MP, Y-MP, SCS-40 or Cyber 205, ETA 10, ETA Piper, Convex C-1, etc.). Then it is much harder to choose between the two methods. The versions of incomplete factorization in SLAP do in fact vectorize, but have short vector lengths and the factorization step is relatively more expensive. Hence, for most problems (i.e., unless your problem is ill conditioned, sic!) diagonal scaling is faster, with its very fast set up time and vectorized (with long vectors) preconditioning step (even though it may take more iterations). If you have several systems (or right hand sides) to solve that can utilize the same preconditioner then the cost of the incomplete factorization can be amortized over these several solutions. This situation gives more advantage to the incomplete factorization methods. If you have a vector machine without hardware gather/scatter (Cray 1, Cray 2 & Cray 3) then the advantages for incomplete factorization are even less.

If you're trying to shoehorn SLAP into your favorite "production code" and can not easily generate either the SLAP Triad or Column format then you are left to your own devices in terms of

preconditioning. Also, you may find that the preconditioners supplied with SLAP are not sufficient for your problem. In this situation we would recommend that you talk with a numerical analyst versed in iterative methods about writing other preconditioning subroutines (e.g., polynomial preconditioning, shifted incomplete factorization, SOR or SSOR iteration). You can always "roll your own" by using the "core" iterative methods and supplying your own MSOLVE and MATVEC (and possibly MTSOLV and MTTVEC) routines.

SYMMETRIC SYSTEMS

If your matrix is symmetric then you would want to use one of the symmetric system solvers. If your system is also positive definite, (Ax,x) (Ax dot product with x) is positive for all non-zero vectors x, then use Conjugate Gradient (SCG, SSDCG, SSICSG). If you're not sure it's SPD (symmetric and Positive Definite) then try SCG anyway and if it works, fine. If you're sure your matrix is not positive definite then you may want to try the iterative refinement methods (SIR) or the GMRES code (SGMRES) if SIR converges too slowly.

NONSYMMETRIC SYSTEMS

This is currently an area of active research in numerical analysis and there are new strategies being developed. Consequently take the following advice with a grain of salt. If you matrix is positive definite, (Ax,x) (Ax dot product with x is positive for all non-zero vectors x), then you can use any of the methods for nonsymmetric systems (Orthomin, GMRES, BiConjugate Gradient, BiConjugate Gradient Squared and Conjugate Gradient applied to the normal equations). If your system is not too ill conditioned then try BiConjugate Gradient Squared (BCGS) or GMRES (SGMRES). Both of these methods converge very quickly and do not require A' or M' (' denotes transpose) information. SGMRES does require some additional storage, though. If the system is very ill conditioned or nearly positive indefinite ((Ax,x) is positive, but may be very small), then GMRES should be the first choice, but try the other methods if you have to fine tune the solution process for a "production code". If you have a great preconditioner for the normal equations (i.e., M is an approximation to the inverse of AA' rather than just A) then this is not a bad route to travel. Old wisdom would say that the normal equations are a disaster (since it squares the condition number of the system and SCG convergence is linked to this number of infamy), but some preconditioners (like incomplete factorization) can reduce the condition number back below that of the original system.

==========
========== Naming Conventions ==========
==========

SLAP iterative methods, matrix vector and preconditioner calculation routines follow a naming convention which, when understood, allows one to determine the iterative method and data structure(s) used. The subroutine naming convention takes the following form:

P[S][M]DESC

where

- P stands for the precision (or data type) of the routine and is required in all names,
- S denotes whether or not the routine requires the SLAP Triad

```
       or Column format (it does if the second letter of the name
       is S and does not otherwise),
    M  stands for the type of preconditioner used (only appears
       in drivers for "core" routines), and
 DESC  is some number of letters describing the method or purpose
       of the routine.  The following is a list of the "DESC"
       fields for iterative methods and their meaning:
         BCG,BC:       BiConjugate Gradient
         CG:           Conjugate Gradient
         CGN,CN:       Conjugate Gradient on the Normal equations
         CGS,CS:       biConjugate Gradient Squared
         GMRES,GMR,GM: Generalized Minimum RESidual
         IR,R:         Iterative Refinement
         JAC:          JACobi's method
         GS:           Gauss-Seidel
         OMN,OM:       OrthoMiN

In the single precision version of SLAP, all routine names start
with an S. The brackets around the S and M designate that these
fields are optional.

Here are some examples of the routines:
1) SBCG: Single precision BiConjugate Gradient "core" routine.
   One can deduce that this is a "core" routine, because the S and
   M fields are missing and BiConjugate Gradient is an iterative
   method.
2) SSDBCG: Single precision, SLAP data structure BCG with Diagonal
   scaling.
3) SSLUBC: Single precision, SLAP data structure BCG with incom-
   plete LU factorization as the preconditioning.
4) SCG: Single precision Conjugate Gradient "core" routine.
5) SSDCG: Single precision, SLAP data structure Conjugate Gradient
   with Diagonal scaling.
6) SSICCG: Single precision, SLAP data structure Conjugate Gra-
   dient with Incomplete Cholesky factorization preconditioning.


====================================================================
===================== USER CALLABLE ROUTINES =====================
====================================================================
The following is a list of  the "user callable" SLAP routines and
their one line descriptions.  The headers denote  the  file names
where the routines can be found, as distributed for UNIX systems.

Note:  Each core routine, SXXX, has a corresponding stop routine,
     ISSXXX.  If the stop routine does not have the specific stop
     test the user requires (e.g., weighted infinity norm),  then
     the user should modify the source for ISSXXX accordingly.

============================== sir.f ==============================
SIR: Preconditioned Iterative Refinement Sparse Ax = b Solver.
SSJAC: Jacobi's Method Iterative Sparse Ax = b Solver.
SSGS: Gauss-Seidel Method Iterative Sparse Ax = b Solver.
SSILUR: Incomplete LU Iterative Refinement Sparse Ax = b Solver.

============================== scg.f ==============================
SCG: Preconditioned Conjugate Gradient Sparse Ax=b Solver.
SSDCG: Diagonally Scaled Conjugate Gradient Sparse Ax=b Solver.
SSICCG: Incomplete Cholesky Conjugate Gradient Sparse Ax=b Solver.
```

```
============================= scgn.f ============================
SCGN: Preconditioned CG Sparse Ax=b Solver for Normal Equations.
SSDCGN: Diagonally Scaled CG Sparse Ax=b Solver for Normal Eqn's.
SSLUCN: Incomplete LU CG Sparse Ax=b Solver for Normal Equations.

============================= sbcg.f ============================
SBCG: Preconditioned BiConjugate Gradient Sparse Ax = b Solver.
SSDBCG: Diagonally Scaled BiConjugate Gradient Sparse Ax=b Solver.
SSLUBC: Incomplete LU BiConjugate Gradient Sparse Ax=b Solver.

============================= scgs.f ============================
SCGS: Preconditioned BiConjugate Gradient Squared Ax=b Solver.
SSDCGS: Diagonally Scaled CGS Sparse Ax=b Solver.
SSLUCS: Incomplete LU BiConjugate Gradient Squared Ax=b Solver.

============================= somn.f ============================
SOMN: Preconditioned Orthomin Sparse Iterative Ax=b Solver.
SSDOMN: Diagonally Scaled Orthomin Sparse Iterative Ax=b Solver.
SSLUOM: Incomplete LU Orthomin Sparse Iterative Ax=b Solver.

============================ sgmres.f ===========================
SGMRES: Preconditioned GMRES Iterative Sparse Ax=b Solver.
SSDGMR: Diagonally Scaled GMRES Iterative Sparse Ax=b Solver.
SSLUGM: Incomplete LU GMRES Iterative Sparse Ax=b Solver.

============================ smset.f ============================
   The following routines are used to set up preconditioners.

SSDS: Diagonal Scaling Preconditioner SLAP Set Up.
SSDSCL: Diagonally Scales/Unscales a SLAP Column Matrix.
SSD2S: Diagonal Scaling Preconditioner SLAP Normal Eqns Set Up.
SS2LT: Lower Triangle Preconditioner SLAP Set Up.
SSICS: Incomplete Cholesky Decomp. Preconditioner SLAP Set Up.
SSILUS: Incomplete LU Decomposition Preconditioner SLAP Set Up.

============================ smvops.f ===========================
   Most of the incomplete  factorization  (LL' and LDU) solvers
   in this  file require an  intermediate routine  to translate
   from the SLAP MSOLVE(N, R, Z, NELT, IA,  JA, A, ISYM, RWORK,
   IWORK) calling  convention to the calling  sequence required
   by  the solve routine.   This generally  is  accomplished by
   fishing out pointers to the preconditioner (stored in RWORK)
   from the  IWORK  array and then making a call to the routine
   that actually does the backsolve.

SSMV: SLAP Column Format Sparse Matrix Vector Product.
SSMTV: SLAP Column Format Sparse Matrix (transpose) Vector Prod.
SSDI: Diagonal Matrix Vector Multiply.
SSLI: SLAP MSOLVE for Lower Triangle Matrix (set up for SSLI2).
SSLI2: Lower Triangle Matrix Backsolve.
SSLLTI: SLAP MSOLVE for LDL' (IC) Fact. (set up for SLLTI2).
SLLTI2: Backsolve routine for LDL' Factorization.
SSLUI: SLAP MSOLVE for LDU Factorization (set up for SSLUI2).
SSLUI2: SLAP Backsolve for LDU Factorization.
SSLUTI: SLAP MTSOLV for LDU Factorization (set up for SSLUI4).
SSLUI4: SLAP Backsolve for LDU Factorization.
SSMMTI: SLAP MSOLVE for LDU Fact of Normal Eq (set up for SSMMI2).
SSMMI2: SLAP Backsolve for LDU Factorization of Normal Equations.

========================== slaputil.f ==========================
```

```
      The following utility routines are useful additions to SLAP.

    SBHIN: Read Sparse Linear System in the Boeing/Harwell Format.
    SCHKW: SLAP WORK/IWORK Array Bounds Checker.
    SCPPLT: Printer Plot of SLAP Column Format Matrix.
    SS2Y: SLAP Triad to SLAP Column Format Converter.
    QS2I1R: Quick Sort Integer array, moving integer and real arrays.
            (Used by SS2Y.)
    STIN: Read in SLAP Triad Format Linear System.
    STOUT: Write out SLAP Triad Format Linear System.


***REFERENCES  1. Mark K. Seager, A SLAP for the Masses, in
                  G. F. Carey, Ed., Parallel Supercomputing: Methods,
                  Algorithms and Applications, Wiley, 1989, pp.135-155.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   880715  DATE WRITTEN
   890404  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890921  Removed TeX from comments.  (FNF)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
           -----( This produced Version 2.0.1. )-----
   891003  Rearranged list of user callable routines to agree with
           order in source deck.  (FNF)
   891004  Updated reference.
   910411  Prologue converted to Version 4.0 format.  (BAB)
           -----( This produced Version 2.0.2. )-----
   910506  Minor improvements to prologue.  (FNF)
   920511  Added complete declaration section.  (WRB)
   920929  Corrected format of reference.  (FNF)
   921019  Improved one-line descriptions, reordering some.  (FNF)
   END PROLOGUE
```

SNBCO

```
      SUBROUTINE SNBCO (ABE, LDA, N, ML, MU, IPVT, RCOND, Z)
***BEGIN PROLOGUE  SNBCO
***PURPOSE  Factor a band matrix using Gaussian elimination and
            estimate the condition number.
***LIBRARY   SLATEC
***CATEGORY  D2A2
***TYPE      SINGLE PRECISION (SNBCO-S, DNBCO-D, CNBCO-C)
***KEYWORDS  BANDED, LINEAR EQUATIONS, MATRIX FACTORIZATION,
             NONSYMMETRIC
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

      SNBCO factors a real band matrix by Gaussian
      elimination and estimates the condition of the matrix.

      If RCOND is not needed, SNBFA is slightly faster.
      To solve  A*X = B , follow SNBCO by SNBSL.
      To compute  INVERSE(A)*C , follow SNBCO by SNBSL.
      To compute  DETERMINANT(A) , follow SNBCO by SNBDI.

      On Entry

         ABE     REAL(LDA, NC)
                 contains the matrix in band storage.  The rows
                 of the original matrix are stored in the rows
                 of ABE and the diagonals of the original matrix
                 are stored in columns 1 through ML+MU+1 of ABE.
                 NC must be .GE. 2*ML+MU+1 .
                 See the comments below for details.

         LDA     INTEGER
                 the leading dimension of the array ABE.
                 LDA must be .GE. N .

         N       INTEGER
                 the order of the original matrix.

         ML      INTEGER
                 number of diagonals below the main diagonal.
                 0 .LE. ML .LT. N .

         MU      INTEGER
                 number of diagonals above the main diagonal.
                 0 .LE. MU .LT. N .
                 More efficient if ML .LE. MU .

      On Return

         ABE     an upper triangular matrix in band storage
                 and the multipliers which were used to obtain it.
                 The factorization can be written  A = L*U , where
                 L is a product of permutation and unit lower
                 triangular matrices and  U  is upper triangular.

         IPVT    INTEGER(N)
                 an integer vector of pivot indices.
```

```
   RCOND   REAL
           an estimate of the reciprocal condition of  A .
           For the system  A*X = B , relative perturbations
           in  A  and  B  of size  EPSILON  may cause
           relative perturbations in  X  of size  EPSILON/RCOND .
           If  RCOND  is so small that the logical expression
                    1.0 + RCOND .EQ. 1.0
           is true, then  A  may be singular to working
           precision.  In particular,  RCOND  is zero  if
           exact singularity is detected or the estimate
           underflows.

   Z       REAL(N)
           a work vector whose contents are usually unimportant.
           If  A  is close to a singular matrix, then  Z  is
           an approximate null vector in the sense that
           NORM(A*Z) = RCOND*NORM(A)*NORM(Z) .
```

Band Storage

If A is a band matrix, the following program segment will set up the input.

```
      ML = (band width below the diagonal)
      MU = (band width above the diagonal)
      DO 20 I = 1, N
         J1 = MAX(1, I-ML)
         J2 = MIN(N, I+MU)
         DO 10 J = J1, J2
            K = J - I + ML + 1
            ABE(I,K) = A(I,J)
   10    CONTINUE
   20 CONTINUE
```

This uses columns 1 through ML+MU+1 of ABE . Furthermore, ML additional columns are needed in ABE starting with column ML+MU+2 for elements generated during the triangularization. The total number of columns needed in ABE is 2*ML+MU+1 .

Example: If the original matrix is

```
11 12 13  0  0  0
21 22 23 24  0  0
 0 32 33 34 35  0
 0  0 43 44 45 46
 0  0  0 54 55 56
 0  0  0  0 65 66
```

then N = 6, ML = 1, MU = 2, LDA .GE. 5 and ABE should contain

```
 * 11 12 13  +     , * = not used
21 22 23 24  +     , + = used for pivoting
32 33 34 35  +
43 44 45 46  +
54 55 56  *  +
65 66  *  *  +
```

***REFERENCES J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.

```
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SASUM, SAXPY, SDOT, SNBFA, SSCAL
***REVISION HISTORY  (YYMMDD)
   800723  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SNBDI

```
      SUBROUTINE SNBDI (ABE, LDA, N, ML, MU, IPVT, DET)
***BEGIN PROLOGUE  SNBDI
***PURPOSE  Compute the determinant of a band matrix using the factors
            computed by SNBCO or SNBFA.
***LIBRARY   SLATEC
***CATEGORY  D3A2
***TYPE      SINGLE PRECISION (SNBDI-S, DNBDI-D, CNBDI-C)
***KEYWORDS  BANDED, DETERMINANT, LINEAR EQUATIONS, NONSYMMETRIC
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

      SNBDI computes the determinant of a band matrix
      using the factors computed by SNBCO or SNBFA.
      If the inverse is needed, use SNBSL  N  times.

      On Entry

         ABE     REAL(LDA, NC)
                 the output from SNBCO or SNBFA.
                 NC must be .GE. 2*ML+MU+1 .

         LDA     INTEGER
                 the leading dimension of the array  ABE .

         N       INTEGER
                 the order of the original matrix.

         ML      INTEGER
                 number of diagonals below the main diagonal.

         MU      INTEGER
                 number of diagonals above the main diagonal.

         IPVT    INTEGER(N)
                 the pivot vector from SNBCO or SNBFA.

      On Return

         DET     REAL(2)
                 determinant of original matrix.
                 Determinant = DET(1) * 10.0**DET(2)
                 with  1.0 .LE. ABS(DET(1)) .LT. 10.0
                 or  DET(1) = 0.0 .

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   800725  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SNBFA

```
      SUBROUTINE SNBFA (ABE, LDA, N, ML, MU, IPVT, INFO)
***BEGIN PROLOGUE  SNBFA
***PURPOSE  Factor a real band matrix by elimination.
***LIBRARY   SLATEC
***CATEGORY  D2A2
***TYPE      SINGLE PRECISION (SNBFA-S, DNBFA-D, CNBFA-C)
***KEYWORDS  BANDED, LINEAR EQUATIONS, MATRIX FACTORIZATION,
             NONSYMMETRIC
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

      SNBFA factors a real band matrix by elimination.

      SNBFA is usually called by SNBCO, but it can be called
      directly with a saving in time if RCOND is not needed.

      On Entry

         ABE     REAL(LDA, NC)
                 contains the matrix in band storage.  The rows
                 of the original matrix are stored in the rows
                 of ABE and the diagonals of the original matrix
                 are stored in columns 1 through ML+MU+1 of ABE.
                 NC must be .GE. 2*ML+MU+1 .
                 See the comments below for details.

         LDA     INTEGER
                 the leading dimension of the array ABE.
                 LDA must be .GE. N .

         N       INTEGER
                 the order of the original matrix.

         ML      INTEGER
                 number of diagonals below the main diagonal.
                 0 .LE. ML .LT. N .

         MU      INTEGER
                 number of diagonals above the main diagonal.
                 0 .LE. MU .LT. N .
                 More efficient if ML .LE. MU .

      On Return

         ABE     an upper triangular matrix in band storage
                 and the multipliers which were used to obtain it.
                 The factorization can be written  A = L*U , where
                 L is a product of permutation and unit lower
                 triangular matrices and  U  is upper triangular.

         IPVT    INTEGER(N)
                 an integer vector of pivot indices.

         INFO    INTEGER
                 =0  normal value
                 =K  if  U(K,K) .EQ. 0.0 .  This is not an error
```

condition for this subroutine, but it does indicate that SNBSL will divide by zero if called. Use RCOND in SNBCO for a reliable indication of singularity.

Band Storage

If A is a band matrix, the following program segment will set up the input.

```
      ML = (band width below the diagonal)
      MU = (band width above the diagonal)
      DO 20 I = 1, N
         J1 = MAX(1, I-ML)
         J2 = MIN(N, I+MU)
         DO 10 J = J1, J2
            K = J - I + ML + 1
            ABE(I,K) = A(I,J)
   10    CONTINUE
   20 CONTINUE
```

This uses columns 1 through ML+MU+1 of ABE . Furthermore, ML additional columns are needed in ABE starting with column ML+MU+2 for elements generated during the triangularization. The total number of columns needed in ABE is 2*ML+MU+1 .

Example: If the original matrix is

```
11 12 13  0  0  0
21 22 23 24  0  0
 0 32 33 34 35  0
 0  0 43 44 45 46
 0  0  0 54 55 56
 0  0  0  0 65 66
```

then N = 6, ML = 1, MU = 2, LDA .GE. 5 and ABE should contain

```
 * 11 12 13  +     , * = not used
21 22 23 24  +     , + = used for pivoting
32 33 34 35  +
43 44 45 46  +
54 55 56  *  +
65 66  *  *  +
```

***REFERENCES J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED ISAMAX, SAXPY, SSCAL, SSWAP
***REVISION HISTORY (YYMMDD)
800606 DATE WRITTEN
890531 Changed all specific intrinsics to generic. (WRB)
890831 Modified array declarations. (WRB)
890831 REVISION DATE from Version 3.2
891214 Prologue converted to Version 4.0 format. (BAB)
920501 Reformatted the REFERENCES section. (WRB)
END PROLOGUE

SNBFS

```
      SUBROUTINE SNBFS (ABE, LDA, N, ML, MU, V, ITASK, IND, WORK, IWORK)
***BEGIN PROLOGUE  SNBFS
***PURPOSE  Solve a general nonsymmetric banded system of linear
            equations.
***LIBRARY   SLATEC
***CATEGORY  D2A2
***TYPE      SINGLE PRECISION (SNBFS-S, DNBFS-D, CNBFS-C)
***KEYWORDS  BANDED, LINEAR EQUATIONS, NONSYMMETRIC
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

    Subroutine SNBFS solves a general nonsymmetric banded NxN
    system of single precision real linear equations using
    SLATEC subroutines SNBCO and SNBSL.  These are adaptations
    of the LINPACK subroutines SBGCO and SGBSL, which require
    a different format for storing the matrix elements.  If
    A  is an NxN real matrix and if  X  and  B  are real
    N-vectors, then SNBFS solves the equation

                          A*X=B.

    A band matrix is a matrix whose nonzero elements are all
    fairly near the main diagonal, specifically  A(I,J) = 0
    if  I-J is greater than  ML  or  J-I  is greater than
    MU .  The integers ML and MU are called the lower and upper
    band widths and  M = ML+MU+1  is the total band width.
    SNBFS uses less time and storage than the corresponding
    program for general matrices (SGEFS) if 2*ML+MU .LT. N  .

    The matrix A is first factored into upper and lower tri-
    angular matrices U and L using partial pivoting.  These
    factors and the pivoting information are used to find the
    solution vector X.  An approximate condition number is
    calculated to provide a rough estimate of the number of
    digits of accuracy in the computed solution.

    If the equation A*X=B is to be solved for more than one vector
    B, the factoring of A does not need to be performed again and
    the option to only solve (ITASK .GT. 1) will be faster for
    the succeeding solutions.  In this case, the contents of A,
    LDA, N and IWORK must not have been altered by the user follow-
    ing factorization (ITASK=1).  IND will not be changed by SNBFS
    in this case.


    Band Storage

          If  A  is a band matrix, the following program segment
          will set up the input.

                  ML = (band width below the diagonal)
                  MU = (band width above the diagonal)
                  DO 20 I = 1, N
                     J1 = MAX(1, I-ML)
                     J2 = MIN(N, I+MU)
                     DO 10 J = J1, J2
```

```
                K = J - I + ML + 1
                ABE(I,K) = A(I,J)
   10    CONTINUE
   20 CONTINUE
```

This uses columns 1 through ML+MU+1 of ABE . Furthermore, ML additional columns are needed in ABE starting with column ML+MU+2 for elements generated during the triangularization. The total number of columns needed in ABE is 2*ML+MU+1 .

Example: If the original matrix is

```
11 12 13  0  0  0
21 22 23 24  0  0
 0 32 33 34 35  0
 0  0 43 44 45 46
 0  0  0 54 55 56
 0  0  0  0 65 66
```

then N = 6, ML = 1, MU = 2, LDA .GE. 5 and ABE should contain

```
 * 11 12 13  +     , * = not used
21 22 23 24  +     , + = used for pivoting
32 33 34 35  +
43 44 45 46  +
54 55 56  *  +
65 66  *  *  +
```

Argument Description ***

ABE REAL(LDA,NC)
on entry, contains the matrix in band storage as described above. NC must not be less than 2*ML+MU+1 . The user is cautioned to specify NC with care since it is not an argument and cannot be checked by SNBFS. The rows of the original matrix are stored in the rows of ABE and the diagonals of the original matrix are stored in columns 1 through ML+MU+1 of ABE .
on return, contains an upper triangular matrix U and the multipliers necessary to construct a matrix L so that A=L*U.

LDA INTEGER
the leading dimension of array ABE. LDA must be greater than or equal to N. (terminal error message IND=-1)

N INTEGER
the order of the matrix A. N must be greater than or equal to 1 . (terminal error message IND=-2)

ML INTEGER
the number of diagonals below the main diagonal. ML must not be less than zero nor greater than or equal to N . (terminal error message IND=-5)

MU INTEGER
the number of diagonals above the main diagonal. MU must not be less than zero nor greater than or equal to N . (terminal error message IND=-6)

V REAL(N)
on entry, the singly subscripted array(vector) of di-

```
              mension N which contains the right hand side B of a
              system of simultaneous linear equations A*X=B.
            on return, V contains the solution vector, X .
    ITASK  INTEGER
            If ITASK=1, the matrix A is factored and then the
              linear equation is solved.
            If ITASK .GT. 1, the equation is solved using the existing
              factored matrix A and IWORK.
            If ITASK .LT. 1, then terminal error message IND=-3 is
              printed.
    IND    INTEGER
            GT. 0  IND is a rough estimate of the number of digits
                    of accuracy in the solution, X.
            LT. 0  See error message corresponding to IND below.
    WORK   REAL(N)
            a singly subscripted array of dimension at least N.
    IWORK  INTEGER(N)
            a singly subscripted array of dimension at least N.

  Error Messages Printed ***

    IND=-1  terminal   N is greater than LDA.
    IND=-2  terminal   N is less than 1.
    IND=-3  terminal   ITASK is less than 1.
    IND=-4  terminal   the matrix A is computationally singular.
                         A solution has not been computed.
    IND=-5  terminal   ML is less than zero or is greater than
                         or equal to N .
    IND=-6  terminal   MU is less than zero or is greater than
                         or equal to N .
    IND=-10 warning    the solution has no apparent significance.
                         The solution may be inaccurate or the matrix
                         A may be poorly scaled.

              Note-  The above terminal(*fatal*) error messages are
                     designed to be handled by XERMSG in which
                     LEVEL=1 (recoverable) and IFLAG=2 .  LEVEL=0
                     for warning error messages from XERMSG.  Unless
                     the user provides otherwise, an error message
                     will be printed followed by an abort.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  R1MACH, SNBCO, SNBSL, XERMSG
***REVISION HISTORY  (YYMMDD)
   800808  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900510  Convert XERRWV calls to XERMSG calls.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SNBIR

```
      SUBROUTINE SNBIR (ABE, LDA, N, ML, MU, V, ITASK, IND, WORK, IWORK)
***BEGIN PROLOGUE  SNBIR
***PURPOSE  Solve a general nonsymmetric banded system of linear
            equations.  Iterative refinement is used to obtain an error
            estimate.
***LIBRARY   SLATEC
***CATEGORY  D2A2
***TYPE      SINGLE PRECISION (SNBIR-S, CNBIR-C)
***KEYWORDS  BANDED, LINEAR EQUATIONS, NONSYMMETRIC
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

    Subroutine SNBIR solves a general nonsymmetric banded NxN
    system of single precision real linear equations using
    SLATEC subroutines SNBFA and SNBSL.  These are adaptations
    of the LINPACK subroutines SGBFA and SGBSL, which require
    a different format for storing the matrix elements.
    One pass of iterative refinement is used only to obtain an
    estimate of the accuracy.  If  A  is an NxN real banded
    matrix and if  X  and  B  are real N-vectors, then SNBIR
    solves the equation

                          A*X=B.

    A band matrix is a matrix whose nonzero elements are all
    fairly near the main diagonal, specifically  A(I,J) = 0
    if  I-J is greater than  ML  or  J-I  is greater than
    MU .  The integers ML and MU are called the lower and upper
    band widths and  M = ML+MU+1  is the total band width.
    SNBIR uses less time and storage than the corresponding
    program for general matrices (SGEIR) if 2*ML+MU .LT. N .

    The matrix A is first factored into upper and lower tri-
    angular matrices U and L using partial pivoting.  These
    factors and the pivoting information are used to find the
    solution vector X .  Then the residual vector is found and used
    to calculate an estimate of the relative error, IND .  IND esti-
    mates the accuracy of the solution only when the input matrix
    and the right hand side are represented exactly in the computer
    and does not take into account any errors in the input data.

    If the equation A*X=B is to be solved for more than one vector
    B, the factoring of A does not need to be performed again and
    the option to only solve (ITASK .GT. 1) will be faster for
    the succeeding solutions.  In this case, the contents of A, LDA,
    N, work and IWORK must not have been altered by the user follow-
    ing factorization (ITASK=1).  IND will not be changed by SNBIR
    in this case.


    Band Storage

          If  A  is a band matrix, the following program segment
          will set up the input.

                  ML = (band width below the diagonal)
```

```
      MU = (band width above the diagonal)
      DO 20 I = 1, N
         J1 = MAX(1, I-ML)
         J2 = MIN(N, I+MU)
         DO 10 J = J1, J2
            K = J - I + ML + 1
            ABE(I,K) = A(I,J)
   10    CONTINUE
   20 CONTINUE
```

This uses columns 1 Through ML+MU+1 of ABE .

Example: If the original matrix is

```
11 12 13  0  0  0
21 22 23 24  0  0
 0 32 33 34 35  0
 0  0 43 44 45 46
 0  0  0 54 55 56
 0  0  0  0 65 66
```

then N = 6, ML = 1, MU = 2, LDA .GE. 5 and ABE should contain

```
 * 11 12 13        , * = not used
21 22 23 24
32 33 34 35
43 44 45 46
54 55 56  *
65 66  *  *
```

Argument Description ***

ABE REAL(LDA,MM)
on entry, contains the matrix in band storage as described above. MM must not be less than M = ML+MU+1 . The user is cautioned to dimension ABE with care since MM is not an argument and cannot be checked by SNBIR. The rows of the original matrix are stored in the rows of ABE and the diagonals of the original matrix are stored in columns 1 through ML+MU+1 of ABE . ABE is not altered by the program.

LDA INTEGER
the leading dimension of array ABE. LDA must be greater than or equal to N. (terminal error message IND=-1)

N INTEGER
the order of the matrix A. N must be greater than or equal to 1 . (terminal error message IND=-2)

ML INTEGER
the number of diagonals below the main diagonal. ML must not be less than zero nor greater than or equal to N . (terminal error message IND=-5)

MU INTEGER
the number of diagonals above the main diagonal. MU must not be less than zero nor greater than or equal to N . (terminal error message IND=-6)

V REAL(N)
on entry, the singly subscripted array(vector) of dimension N which contains the right hand side B of a

```
              system of simultaneous linear equations A*X=B.
            on return, V contains the solution vector, X .
   ITASK  INTEGER
            If ITASK=1, the matrix A is factored and then the
              linear equation is solved.
            If ITASK .GT. 1, the equation is solved using the existing
              factored matrix A and IWORK.
            If ITASK .LT. 1, then terminal error message IND=-3 is
              printed.
   IND    INTEGER
            GT. 0  IND is a rough estimate of the number of digits
                    of accuracy in the solution, X .  IND=75 means
                    that the solution vector  X  is zero.
            LT. 0  See error message corresponding to IND below.
   WORK   REAL(N*(NC+1))
            a singly subscripted array of dimension at least
            N*(NC+1)  where  NC = 2*ML+MU+1 .
   IWORK  INTEGER(N)
            a singly subscripted array of dimension at least N.

  Error Messages Printed ***

   IND=-1  terminal   N is greater than LDA.
   IND=-2  terminal   N is less than 1.
   IND=-3  terminal   ITASK is less than 1.
   IND=-4  terminal   the matrix A is computationally singular.
                        A solution has not been computed.
   IND=-5  terminal   ML is less than zero or is greater than
                        or equal to N .
   IND=-6  terminal   MU is less than zero or is greater than
                        or equal to N .
   IND=-10 warning    the solution has no apparent significance.
                        The solution may be inaccurate or the matrix
                        A may be poorly scaled.

              Note-  The above terminal(*fatal*) error messages are
                     designed to be handled by XERMSG in which
                     LEVEL=1 (recoverable) and IFLAG=2 .  LEVEL=0
                     for warning error messages from XERMSG.  Unless
                     the user provides otherwise, an error message
                     will be printed followed by an abort.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  R1MACH, SASUM, SCOPY, SDSDOT, SNBFA, SNBSL, XERMSG
***REVISION HISTORY  (YYMMDD)
   800815  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900510  Convert XERRWV calls to XERMSG calls.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SNBSL

```
      SUBROUTINE SNBSL (ABE, LDA, N, ML, MU, IPVT, B, JOB)
***BEGIN PROLOGUE  SNBSL
***PURPOSE  Solve a real band system using the factors computed by
            SNBCO or SNBFA.
***LIBRARY   SLATEC
***CATEGORY  D2A2
***TYPE      SINGLE PRECISION (SNBSL-S, DNBSL-D, CNBSL-C)
***KEYWORDS  BANDED, LINEAR EQUATIONS, NONSYMMETRIC, SOLVE
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

      SNBSL solves the real band system
      A * X = B  or  TRANS(A) * X = B
      using the factors computed by SNBCO or SNBFA.

      On Entry

         ABE     REAL(LDA, NC)
                 the output from SNBCO or SNBFA.
                 NC must be .GE. 2*ML+MU+1 .

         LDA     INTEGER
                 the leading dimension of the array  ABE .

         N       INTEGER
                 the order of the original matrix.

         ML      INTEGER
                 number of diagonals below the main diagonal.

         MU      INTEGER
                 number of diagonals above the main diagonal.

         IPVT    INTEGER(N)
                 the pivot vector from SNBCO or SNBFA.

         B       REAL(N)
                 the right hand side vector.

         JOB     INTEGER
                 = 0         to solve  A*X = B .
                 = nonzero   to solve  TRANS(A)*X = B , where
                             TRANS(A)  is the transpose.

      On Return

         B       the solution vector  X .

      Error Condition

         A division by zero will occur if the input factor contains a
         zero on the diagonal.  Technically, this indicates singularity,
         but it is often caused by improper arguments or improper
         setting of LDA.  It will not occur if the subroutines are
         called correctly and if SNBCO has set RCOND .GT. 0.0
         or SNBFA has set INFO .EQ. 0 .
```

```
     To compute  INVERSE(A) * C  where  C  is a matrix
     with  P  columns
           CALL SNBCO(ABE,LDA,N,ML,MU,IPVT,RCOND,Z)
           IF (RCOND is too small) GO TO ...
           DO 10 J = 1, P
             CALL SNBSL(ABE,LDA,N,ML,MU,IPVT,C(1,J),0)
        10 CONTINUE

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SDOT
***REVISION HISTORY  (YYMMDD)
   800717  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SNLS1

```
      SUBROUTINE SNLS1 (FCN, IOPT, M, N, X, FVEC, FJAC, LDFJAC, FTOL,
         XTOL, GTOL, MAXFEV, EPSFCN, DIAG, MODE, FACTOR, NPRINT, INFO,
     +   NFEV, NJEV, IPVT, QTF, WA1, WA2, WA3, WA4)
***BEGIN PROLOGUE  SNLS1
***PURPOSE  Minimize the sum of the squares of M nonlinear functions
            in N variables by a modification of the Levenberg-Marquardt
            algorithm.
***LIBRARY   SLATEC
***CATEGORY  K1B1A1, K1B1A2
***TYPE      SINGLE PRECISION (SNLS1-S, DNLS1-D)
***KEYWORDS  LEVENBERG-MARQUARDT, NONLINEAR DATA FITTING,
             NONLINEAR LEAST SQUARES
***AUTHOR  Hiebert, K. L., (SNLA)
***DESCRIPTION
```

1. Purpose.

The purpose of SNLS1 is to minimize the sum of the squares of M nonlinear functions in N variables by a modification of the Levenberg-Marquardt algorithm. The user must provide a subroutine which calculates the functions. The user has the option of how the Jacobian will be supplied. The user can supply the full Jacobian, or the rows of the Jacobian (to avoid storing the full Jacobian), or let the code approximate the Jacobian by forward-differencing. This code is the combination of the MINPACK codes (Argonne) LMDER, LMDIF, and LMSTR.

2. Subroutine and Type Statements.

```
      SUBROUTINE SNLS1(FCN,IOPT,M,N,X,FVEC,FJAC,LDFJAC,FTOL,XTOL,
     *                 GTOL,MAXFEV,EPSFCN,DIAG,MODE,FACTOR,NPRINT,INFO
     *                 ,NFEV,NJEV,IPVT,QTF,WA1,WA2,WA3,WA4)
      INTEGER IOPT,M,N,LDFJAC,MAXFEV,MODE,NPRINT,INFO,NFEV,NJEV
      INTEGER IPVT(N)
      REAL FTOL,XTOL,GTOL,EPSFCN,FACTOR
      REAL X(N),FVEC(M),FJAC(LDFJAC,N),DIAG(N),QTF(N),
     *     WA1(N),WA2(N),WA3(N),WA4(M)
```

3. Parameters.

Parameters designated as input parameters must be specified on entry to SNLS1 and are not changed on exit, while parameters designated as output parameters need not be specified on entry and are set to appropriate values on exit from SNLS1.

FCN is the name of the user-supplied subroutine which calculates the functions. If the user wants to supply the Jacobian (IOPT=2 or 3), then FCN must be written to calculate the Jacobian, as well as the functions. See the explanation of the IOPT argument below.
If the user wants the iterates printed (NPRINT positive), then FCN must do the printing. See the explanation of NPRINT below. FCN must be declared in an EXTERNAL statement in the calling program and should be written as follows.

```
SUBROUTINE FCN(IFLAG,M,N,X,FVEC,FJAC,LDFJAC)
INTEGER IFLAG,LDFJAC,M,N
REAL X(N),FVEC(M)
----------
FJAC and LDFJAC may be ignored     , if IOPT=1.
REAL FJAC(LDFJAC,N)                , if IOPT=2.
REAL FJAC(N)                       , if IOPT=3.
----------
  If IFLAG=0, the values in X and FVEC are available
  for printing.  See the explanation of NPRINT below.
  IFLAG will never be zero unless NPRINT is positive.
  The values of X and FVEC must not be changed.
RETURN
----------
  If IFLAG=1, calculate the functions at X and return
  this vector in FVEC.
RETURN
----------
  If IFLAG=2, calculate the full Jacobian at X and return
  this matrix in FJAC.  Note that IFLAG will never be 2 unless
  IOPT=2.  FVEC contains the function values at X and must
  not be altered.  FJAC(I,J) must be set to the derivative
  of FVEC(I) with respect to X(J).
RETURN
----------
  If IFLAG=3, calculate the LDFJAC-th row of the Jacobian
  and return this vector in FJAC.  Note that IFLAG will
  never be 3 unless IOPT=3.  FVEC contains the function
  values at X and must not be altered.  FJAC(J) must be
  set to the derivative of FVEC(LDFJAC) with respect to X(J).
RETURN
----------
END
```

The value of IFLAG should not be changed by FCN unless the user wants to terminate execution of SNLS1. In this case, set IFLAG to a negative integer.

IOPT is an input variable which specifies how the Jacobian will be calculated. If IOPT=2 or 3, then the user must supply the Jacobian, as well as the function values, through the subroutine FCN. If IOPT=2, the user supplies the full Jacobian with one call to FCN. If IOPT=3, the user supplies one row of the Jacobian with each call. (In this manner, storage can be saved because the full Jacobian is not stored.) If IOPT=1, the code will approximate the Jacobian by forward differencing.

M is a positive integer input variable set to the number of functions.

N is a positive integer input variable set to the number of variables. N must not exceed M.

X is an array of length N. On input, X must contain an initial estimate of the solution vector. On output, X contains the

final estimate of the solution vector.

FVEC is an output array of length M which contains the functions evaluated at the output X.

FJAC is an output array. For IOPT=1 and 2, FJAC is an M by N array. For IOPT=3, FJAC is an N by N array. The upper N by N submatrix of FJAC contains an upper triangular matrix R with diagonal elements of nonincreasing magnitude such that

$$P^T*(JAC^T*JAC)*P = R^T*R,$$

where P is a permutation matrix and JAC is the final calculated Jacobian. Column J of P is column IPVT(J) (see below) of the identity matrix. The lower part of FJAC contains information generated during the computation of R.

LDFJAC is a positive integer input variable which specifies the leading dimension of the array FJAC. For IOPT=1 and 2, LDFJAC must not be less than M. For IOPT=3, LDFJAC must not be less than N.

FTOL is a non-negative input variable. Termination occurs when both the actual and predicted relative reductions in the sum of squares are at most FTOL. Therefore, FTOL measures the relative error desired in the sum of squares. Section 4 contains more details about FTOL.

XTOL is a non-negative input variable. Termination occurs when the relative error between two consecutive iterates is at most XTOL. Therefore, XTOL measures the relative error desired in the approximate solution. Section 4 contains more details about XTOL.

GTOL is a non-negative input variable. Termination occurs when the cosine of the angle between FVEC and any column of the Jacobian is at most GTOL in absolute value. Therefore, GTOL measures the orthogonality desired between the function vector and the columns of the Jacobian. Section 4 contains more details about GTOL.

MAXFEV is a positive integer input variable. Termination occurs when the number of calls to FCN to evaluate the functions has reached MAXFEV.

EPSFCN is an input variable used in determining a suitable step for the forward-difference approximation. This approximation assumes that the relative errors in the functions are of the order of EPSFCN. If EPSFCN is less than the machine precision, it is assumed that the relative errors in the functions are of the order of the machine precision. If IOPT=2 or 3, then EPSFCN can be ignored (treat it as a dummy argument).

DIAG is an array of length N. If MODE = 1 (see below), DIAG is internally set. If MODE = 2, DIAG must contain positive entries that serve as implicit (multiplicative) scale factors for the variables.

MODE is an integer input variable. If MODE = 1, the variables

will be scaled internally. If MODE = 2, the scaling is specified by the input DIAG. Other values of MODE are equivalent to MODE = 1.

FACTOR is a positive input variable used in determining the initial step bound. This bound is set to the product of FACTOR and the Euclidean norm of DIAG*X if nonzero, or else to FACTOR itself. In most cases FACTOR should lie in the interval (.1,100.). 100. is a generally recommended value.

NPRINT is an integer input variable that enables controlled printing of iterates if it is positive. In this case, FCN is called with IFLAG = 0 at the beginning of the first iteration and every NPRINT iterations thereafter and immediately prior to return, with X and FVEC available for printing. Appropriate print statements must be added to FCN (see example) and FVEC should not be altered. If NPRINT is not positive, no special calls to FCN with IFLAG = 0 are made.

INFO is an integer output variable. If the user has terminated execution, INFO is set to the (negative) value of IFLAG. See description of FCN and JAC. Otherwise, INFO is set as follows.

INFO = 0 improper input parameters.

INFO = 1 both actual and predicted relative reductions in the sum of squares are at most FTOL.

INFO = 2 relative error between two consecutive iterates is at most XTOL.

INFO = 3 conditions for INFO = 1 and INFO = 2 both hold.

INFO = 4 the cosine of the angle between FVEC and any column of the Jacobian is at most GTOL in absolute value.

INFO = 5 number of calls to FCN for function evaluation has reached MAXFEV.

INFO = 6 FTOL is too small. No further reduction in the sum of squares is possible.

INFO = 7 XTOL is too small. No further improvement in the approximate solution X is possible.

INFO = 8 GTOL is too small. FVEC is orthogonal to the columns of the Jacobian to machine precision.

Sections 4 and 5 contain more details about INFO.

NFEV is an integer output variable set to the number of calls to FCN for function evaluation.

NJEV is an integer output variable set to the number of evaluations of the full Jacobian. If IOPT=2, only one call to FCN is required for each evaluation of the full Jacobian.
If IOPT=3, the M calls to FCN are required.
If IOPT=1, then NJEV is set to zero.

IPVT is an integer output array of length N. IPVT defines a

permutation matrix P such that JAC*P = Q*R, where JAC is the final calculated Jacobian, Q is orthogonal (not stored), and R is upper triangular with diagonal elements of nonincreasing magnitude. Column J of P is column IPVT(J) of the identity matrix.

QTF is an output array of length N which contains the first N elements of the vector (Q transpose)*FVEC.

WA1, WA2, and WA3 are work arrays of length N.

WA4 is a work array of length M.

4. Successful Completion.

The accuracy of SNLS1 is controlled by the convergence parameters FTOL, XTOL, and GTOL. These parameters are used in tests which make three types of comparisons between the approximation X and a solution XSOL. SNLS1 terminates when any of the tests is satisfied. If any of the convergence parameters is less than the machine precision (as defined by the function R1MACH(4)), then SNLS1 only attempts to satisfy the test defined by the machine precision. Further progress is not usually possible.

The tests assume that the functions are reasonably well behaved, and, if the Jacobian is supplied by the user, that the functions and the Jacobian are coded consistently. If these conditions are not satisfied, then SNLS1 may incorrectly indicate convergence. If the Jacobian is coded correctly or IOPT=1, then the validity of the answer can be checked, for example, by rerunning SNLS1 with tighter tolerances.

First Convergence Test. If ENORM(Z) denotes the Euclidean norm of a vector Z, then this test attempts to guarantee that

```
ENORM(FVEC) .LE. (1+FTOL)*ENORM(FVECS),
```

where FVECS denotes the functions evaluated at XSOL. If this condition is satisfied with FTOL = 10**(-K), then the final residual norm ENORM(FVEC) has K significant decimal digits and INFO is set to 1 (or to 3 if the second test is also satisfied). Unless high precision solutions are required, the recommended value for FTOL is the square root of the machine precision.

Second Convergence Test. If D is the diagonal matrix whose entries are defined by the array DIAG, then this test attempts to guarantee that

```
ENORM(D*(X-XSOL)) .LE. XTOL*ENORM(D*XSOL).
```

If this condition is satisfied with XTOL = 10**(-K), then the larger components of D*X have K significant decimal digits and INFO is set to 2 (or to 3 if the first test is also satisfied). There is a danger that the smaller components of D*X may have large relative errors, but if MODE = 1, then the accuracy of the components of X is usually related to their sensitivity. Unless high precision solutions are required, the recommended value for XTOL is the square root of the

machine precision.

Third Convergence Test. This test is satisfied when the cosine of the angle between FVEC and any column of the Jacobian at X is at most GTOL in absolute value. There is no clear relationship between this test and the accuracy of SNLS1, and furthermore, the test is equally well satisfied at other critical points, namely maximizers and saddle points. Therefore, termination caused by this test (INFO = 4) should be examined carefully. The recommended value for GTOL is zero.

5. Unsuccessful Completion.

Unsuccessful termination of SNLS1 can be due to improper input parameters, arithmetic interrupts, or an excessive number of function evaluations.

Improper Input Parameters. INFO is set to 0 if IOPT .LT. 1 or IOPT .GT. 3, or N .LE. 0, or M .LT. N, or for IOPT=1 or 2 LDFJAC .LT. M, or for IOPT=3 LDFJAC .LT. N, or FTOL .LT. 0.E0, or XTOL .LT. 0.E0, or GTOL .LT. 0.E0, or MAXFEV .LE. 0, or FACTOR .LE. 0.E0.

Arithmetic Interrupts. If these interrupts occur in the FCN subroutine during an early stage of the computation, they may be caused by an unacceptable choice of X by SNLS1. In this case, it may be possible to remedy the situation by rerunning SNLS1 with a smaller value of FACTOR.

Excessive Number of Function Evaluations. A reasonable value for MAXFEV is 100*(N+1) for IOPT=2 or 3 and 200*(N+1) for IOPT=1. If the number of calls to FCN reaches MAXFEV, then this indicates that the routine is converging very slowly as measured by the progress of FVEC, and INFO is set to 5. In this case, it may be helpful to restart SNLS1 with MODE set to 1.

6. Characteristics of the Algorithm.

SNLS1 is a modification of the Levenberg-Marquardt algorithm. Two of its main characteristics involve the proper use of implicitly scaled variables (if MODE = 1) and an optimal choice for the correction. The use of implicitly scaled variables achieves scale invariance of SNLS1 and limits the size of the correction in any direction where the functions are changing rapidly. The optimal choice of the correction guarantees (under reasonable conditions) global convergence from starting points far from the solution and a fast rate of convergence for problems with small residuals.

Timing. The time required by SNLS1 to solve a given problem depends on M and N, the behavior of the functions, the accuracy requested, and the starting point. The number of arithmetic operations needed by SNLS1 is about N**3 to process each evaluation of the functions (call to FCN) and to process each evaluation of the Jacobian it takes M*N**2 for IOPT=2 (one call to FCN), M*N**2 for IOPT=1 (N calls to FCN) and 1.5*M*N**2 for IOPT=3 (M calls to FCN). Unless FCN

can be evaluated quickly, the timing of SNLS1 will be strongly influenced by the time spent in FCN.

Storage. SNLS1 requires (M*N + 2*M + 6*N) for IOPT=1 or 2 and (N**2 + 2*M + 6*N) for IOPT=3 single precision storage locations and N integer storage locations, in addition to the storage required by the program. There are no internally declared storage arrays.

*Long Description:

7. Example.

The problem is to determine the values of X(1), X(2), and X(3) which provide the best fit (in the least squares sense) of

```
      X(1) + U(I)/(V(I)*X(2) + W(I)*X(3)),  I = 1, 15
```

to the data

```
      Y = (0.14,0.18,0.22,0.25,0.29,0.32,0.35,0.39,
           0.37,0.58,0.73,0.96,1.34,2.10,4.39),
```

where U(I) = I, V(I) = 16 - I, and W(I) = MIN(U(I),V(I)). The I-th component of FVEC is thus defined by

```
      Y(I) - (X(1) + U(I)/(V(I)*X(2) + W(I)*X(3))).
```

****

```
      PROGRAM TEST
C
C     Driver for SNLS1 example.
C
      INTEGER J,IOPT,M,N,LDFJAC,MAXFEV,MODE,NPRINT,INFO,NFEV,NJEV,
     *        NWRITE
      INTEGER IPVT(3)
      REAL FTOL,XTOL,GTOL,FACTOR,FNORM,EPSFCN
      REAL X(3),FVEC(15),FJAC(15,3),DIAG(3),QTF(3),
     *     WA1(3),WA2(3),WA3(3),WA4(15)
      REAL ENORM,R1MACH
      EXTERNAL FCN
      DATA NWRITE /6/
C
      IOPT = 1
      M = 15
      N = 3
C
C     The following starting values provide a rough fit.
C
      X(1) = 1.E0
      X(2) = 1.E0
      X(3) = 1.E0
C
      LDFJAC = 15
C
C     Set FTOL and XTOL to the square root of the machine precision
C     and GTOL to zero.  Unless high precision solutions are
C     required, these are the recommended settings.
C
```

```
      FTOL = SQRT(R1MACH(4))
      XTOL = SQRT(R1MACH(4))
      GTOL = 0.E0
C
      MAXFEV = 400
      EPSFCN = 0.0
      MODE = 1
      FACTOR = 1.E2
      NPRINT = 0
C
      CALL SNLS1(FCN,IOPT,M,N,X,FVEC,FJAC,LDFJAC,FTOL,XTOL,
     *           GTOL,MAXFEV,EPSFCN,DIAG,MODE,FACTOR,NPRINT,
     *           INFO,NFEV,NJEV,IPVT,QTF,WA1,WA2,WA3,WA4)
      FNORM = ENORM(M,FVEC)
      WRITE (NWRITE,1000) FNORM,NFEV,NJEV,INFO,(X(J),J=1,N)
      STOP
 1000 FORMAT (5X,' FINAL L2 NORM OF THE RESIDUALS',E15.7 //
     *        5X,' NUMBER OF FUNCTION EVALUATIONS',I10 //
     *        5X,' NUMBER OF JACOBIAN EVALUATIONS',I10 //
     *        5X,' EXIT PARAMETER',16X,I10 //
     *        5X,' FINAL APPROXIMATE SOLUTION' // 5X,3E15.7)
      END
      SUBROUTINE FCN(IFLAG,M,N,X,FVEC,DUM,IDUM)
C     This is the form of the FCN routine if IOPT=1,
C     that is, if the user does not calculate the Jacobian.
      INTEGER M,N,IFLAG
      REAL X(N),FVEC(M)
      INTEGER I
      REAL TMP1,TMP2,TMP3,TMP4
      REAL Y(15)
      DATA Y(1),Y(2),Y(3),Y(4),Y(5),Y(6),Y(7),Y(8),
     *     Y(9),Y(10),Y(11),Y(12),Y(13),Y(14),Y(15)
     *     /1.4E-1,1.8E-1,2.2E-1,2.5E-1,2.9E-1,3.2E-1,3.5E-1,3.9E-1,
     *      3.7E-1,5.8E-1,7.3E-1,9.6E-1,1.34E0,2.1E0,4.39E0/
C
      IF (IFLAG .NE. 0) GO TO 5
C
C     Insert print statements here when NPRINT is positive.
C
      RETURN
    5 CONTINUE
      DO 10 I = 1, M
         TMP1 = I
         TMP2 = 16 - I
         TMP3 = TMP1
         IF (I .GT. 8) TMP3 = TMP2
         FVEC(I) = Y(I) - (X(1) + TMP1/(X(2)*TMP2 + X(3)*TMP3))
   10    CONTINUE
      RETURN
      END
```

```
      Results obtained with different compilers or machines
      may be slightly different.

      FINAL L2 NORM OF THE RESIDUALS  0.9063596E-01

      NUMBER OF FUNCTION EVALUATIONS        25

      NUMBER OF JACOBIAN EVALUATIONS         0
```

```
      EXIT PARAMETER                        1

      FINAL APPROXIMATE SOLUTION

       0.8241058E-01  0.1133037E+01  0.2343695E+01


      For IOPT=2, FCN would be modified as follows to also
      calculate the full Jacobian when IFLAG=2.

      SUBROUTINE FCN(IFLAG,M,N,X,FVEC,FJAC,LDFJAC)
C
C     This is the form of the FCN routine if IOPT=2,
C     that is, if the user calculates the full Jacobian.
C
      INTEGER LDFJAC,M,N,IFLAG
      REAL X(N),FVEC(M)
      REAL FJAC(LDFJAC,N)
      INTEGER I
      REAL TMP1,TMP2,TMP3,TMP4
      REAL Y(15)
      DATA Y(1),Y(2),Y(3),Y(4),Y(5),Y(6),Y(7),Y(8),
     *     Y(9),Y(10),Y(11),Y(12),Y(13),Y(14),Y(15)
     *     /1.4E-1,1.8E-1,2.2E-1,2.5E-1,2.9E-1,3.2E-1,3.5E-1,3.9E-1,
     *      3.7E-1,5.8E-1,7.3E-1,9.6E-1,1.34E0,2.1E0,4.39E0/
C
      IF (IFLAG .NE. 0) GO TO 5
C
C     Insert print statements here when NPRINT is positive.
C
      RETURN
    5 CONTINUE
      IF(IFLAG.NE.1) GO TO 20
      DO 10 I = 1, M
         TMP1 = I
         TMP2 = 16 - I
         TMP3 = TMP1
         IF (I .GT. 8) TMP3 = TMP2
         FVEC(I) = Y(I) - (X(1) + TMP1/(X(2)*TMP2 + X(3)*TMP3))
   10    CONTINUE
      RETURN
C
C     Below, calculate the full Jacobian.
C
   20    CONTINUE
C
      DO 30 I = 1, M
         TMP1 = I
         TMP2 = 16 - I
         TMP3 = TMP1
         IF (I .GT. 8) TMP3 = TMP2
         TMP4 = (X(2)*TMP2 + X(3)*TMP3)**2
         FJAC(I,1) = -1.E0
         FJAC(I,2) = TMP1*TMP2/TMP4
         FJAC(I,3) = TMP1*TMP3/TMP4
   30    CONTINUE
      RETURN
      END
```

```
      For IOPT = 3, FJAC would be dimensioned as FJAC(3,3),
        LDFJAC would be set to 3, and FCN would be written as
        follows to calculate a row of the Jacobian when IFLAG=3.

      SUBROUTINE FCN(IFLAG,M,N,X,FVEC,FJAC,LDFJAC)
C     This is the form of the FCN routine if IOPT=3,
C     that is, if the user calculates the Jacobian row by row.
      INTEGER M,N,IFLAG
      REAL X(N),FVEC(M)
      REAL FJAC(N)
      INTEGER I
      REAL TMP1,TMP2,TMP3,TMP4
      REAL Y(15)
      DATA Y(1),Y(2),Y(3),Y(4),Y(5),Y(6),Y(7),Y(8),
     *     Y(9),Y(10),Y(11),Y(12),Y(13),Y(14),Y(15)
     *     /1.4E-1,1.8E-1,2.2E-1,2.5E-1,2.9E-1,3.2E-1,3.5E-1,3.9E-1,
     *      3.7E-1,5.8E-1,7.3E-1,9.6E-1,1.34E0,2.1E0,4.39E0/
C
      IF (IFLAG .NE. 0) GO TO 5
C
C     Insert print statements here when NPRINT is positive.
C
      RETURN
    5 CONTINUE
      IF( IFLAG.NE.1) GO TO 20
      DO 10 I = 1, M
         TMP1 = I
         TMP2 = 16 - I
         TMP3 = TMP1
         IF (I .GT. 8) TMP3 = TMP2
         FVEC(I) = Y(I) - (X(1) + TMP1/(X(2)*TMP2 + X(3)*TMP3))
   10    CONTINUE
      RETURN
C
C     Below, calculate the LDFJAC-th row of the Jacobian.
C
   20 CONTINUE

      I = LDFJAC
         TMP1 = I
         TMP2 = 16 - I
         TMP3 = TMP1
         IF (I .GT. 8) TMP3 = TMP2
         TMP4 = (X(2)*TMP2 + X(3)*TMP3)**2
         FJAC(1) = -1.E0
         FJAC(2) = TMP1*TMP2/TMP4
         FJAC(3) = TMP1*TMP3/TMP4
      RETURN
      END
```

***REFERENCES Jorge J. More, The Levenberg-Marquardt algorithm: implementation and theory. In Numerical Analysis Proceedings (Dundee, June 28 - July 1, 1977, G. A. Watson, Editor), Lecture Notes in Mathematics 630, Springer-Verlag, 1978.

***ROUTINES CALLED CHKDER, ENORM, FDJAC3, LMPAR, QRFAC, R1MACH, RWUPDT, XERMSG

***REVISION HISTORY (YYMMDD)

800301 DATE WRITTEN

```
890531  Changed all specific intrinsics to generic.  (WRB)
890531  REVISION DATE from Version 3.2
891214  Prologue converted to Version 4.0 format.  (BAB)
900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
900510  Convert XERRWV calls to XERMSG calls.  (RWC)
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

SNLS1E

```
      SUBROUTINE SNLS1E (FCN, IOPT, M, N, X, FVEC, TOL, NPRINT, INFO,
     +   IW, WA, LWA)
***BEGIN PROLOGUE  SNLS1E
***PURPOSE  An easy-to-use code which minimizes the sum of the squares
            of M nonlinear functions in N variables by a modification
            of the Levenberg-Marquardt algorithm.
***LIBRARY   SLATEC
***CATEGORY  K1B1A1, K1B1A2
***TYPE      SINGLE PRECISION (SNLS1E-S, DNLS1E-D)
***KEYWORDS  EASY-TO-USE, LEVENBERG-MARQUARDT, NONLINEAR DATA FITTING,
             NONLINEAR LEAST SQUARES
***AUTHOR  Hiebert, K. L., (SNLA)
***DESCRIPTION
```

1. Purpose.

The purpose of SNLS1E is to minimize the sum of the squares of M nonlinear functions in N variables by a modification of the Levenberg-Marquardt algorithm. This is done by using the more general least-squares solver SNLS1. The user must provide a subroutine which calculates the functions. The user has the option of how the Jacobian will be supplied. The user can supply the full Jacobian, or the rows of the Jacobian (to avoid storing the full Jacobian), or let the code approximate the Jacobian by forward-differencing. This code is the combination of the MINPACK codes (Argonne) LMDER1, LMDIF1, and LMSTR1.

2. Subroutine and Type Statements.

```
      SUBROUTINE SNLS1E(FCN,IOPT,M,N,X,FVEC,TOL,NPRINT,
     *                  INFO,IW,WA,LWA)
      INTEGER IOPT,M,N,NPRINT,INFO,LWA
      INTEGER IW(N)
      REAL TOL
      REAL X(N),FVEC(M),WA(LWA)
      EXTERNAL FCN
```

3. Parameters.

Parameters designated as input parameters must be specified on entry to SNLS1E and are not changed on exit, while parameters designated as output parameters need not be specified on entry and are set to appropriate values on exit from SNLS1E.

FCN is the name of the user-supplied subroutine which calculates the functions. If the user wants to supply the Jacobian (IOPT=2 or 3), then FCN must be written to calculate the Jacobian, as well as the functions. See the explanation of the IOPT argument below.
If the user wants the iterates printed (NPRINT positive), then FCN must do the printing. See the explanation of NPRINT below. FCN must be declared in an EXTERNAL statement in the calling program and should be written as follows.

```
SUBROUTINE FCN(IFLAG,M,N,X,FVEC,FJAC,LDFJAC)
INTEGER IFLAG,LDFJAC,M,N
REAL X(N),FVEC(M)
----------
FJAC and LDFJAC may be ignored     , if IOPT=1.
REAL FJAC(LDFJAC,N)                , if IOPT=2.
REAL FJAC(N)                       , if IOPT=3.
----------
  If IFLAG=0, the values in X and FVEC are available
  for printing.  See the explanation of NPRINT below.
  IFLAG will never be zero unless NPRINT is positive.
  The values of X and FVEC must not be changed.
RETURN
----------
  If IFLAG=1, calculate the functions at X and return
  this vector in FVEC.
RETURN
----------
  If IFLAG=2, calculate the full Jacobian at X and return
  this matrix in FJAC.  Note that IFLAG will never be 2 unless
  IOPT=2.  FVEC contains the function values at X and must
  not be altered.  FJAC(I,J) must be set to the derivative
  of FVEC(I) with respect to X(J).
RETURN
----------
  If IFLAG=3, calculate the LDFJAC-th row of the Jacobian
  and return this vector in FJAC.  Note that IFLAG will
  never be 3 unless IOPT=3.  FVEC contains the function
  values at X and must not be altered.  FJAC(J) must be
  set to the derivative of FVEC(LDFJAC) with respect to X(J).
RETURN
----------
END
```

The value of IFLAG should not be changed by FCN unless the user wants to terminate execution of SNLS1E. In this case, set IFLAG to a negative integer.

IOPT is an input variable which specifies how the Jacobian will be calculated. If IOPT=2 or 3, then the user must supply the Jacobian, as well as the function values, through the subroutine FCN. If IOPT=2, the user supplies the full Jacobian with one call to FCN. If IOPT=3, the user supplies one row of the Jacobian with each call. (In this manner, storage can be saved because the full Jacobian is not stored.) If IOPT=1, the code will approximate the Jacobian by forward differencing.

M is a positive integer input variable set to the number of functions.

N is a positive integer input variable set to the number of variables. N must not exceed M.

X is an array of length N. On input, X must contain an initial estimate of the solution vector. On output, X contains the final estimate of the solution vector.

FVEC is an output array of length M which contains the functions evaluated at the output X.

TOL is a non-negative input variable. Termination occurs when the algorithm estimates either that the relative error in the sum of squares is at most TOL or that the relative error between X and the solution is at most TOL. Section 4 contains more details about TOL.

NPRINT is an integer input variable that enables controlled printing of iterates if it is positive. In this case, FCN is called with IFLAG = 0 at the beginning of the first iteration and every NPRINT iterations thereafter and immediately prior to return, with X and FVEC available for printing. Appropriate print statements must be added to FCN (see example) and FVEC should not be altered. If NPRINT is not positive, no special calls of FCN with IFLAG = 0 are made.

INFO is an integer output variable. If the user has terminated execution, INFO is set to the (negative) value of IFLAG. See description of FCN and JAC. Otherwise, INFO is set as follows.

INFO = 0 improper input parameters.

INFO = 1 algorithm estimates that the relative error in the sum of squares is at most TOL.

INFO = 2 algorithm estimates that the relative error between X and the solution is at most TOL.

INFO = 3 conditions for INFO = 1 and INFO = 2 both hold.

INFO = 4 FVEC is orthogonal to the columns of the Jacobian to machine precision.

INFO = 5 number of calls to FCN has reached 100*(N+1) for IOPT=2 or 3 or 200*(N+1) for IOPT=1.

INFO = 6 TOL is too small. No further reduction in the sum of squares is possible.

INFO = 7 TOL is too small. No further improvement in the approximate solution X is possible.

Sections 4 and 5 contain more details about INFO.

IW is an INTEGER work array of length N.

WA is a work array of length LWA.

LWA is a positive integer input variable not less than N*(M+5)+M for IOPT=1 and 2 or N*(N+5)+M for IOPT=3.

4. Successful Completion.

The accuracy of SNLS1E is controlled by the convergence parameter TOL. This parameter is used in tests which make three types of comparisons between the approximation X and a solution XSOL.

SNLS1E terminates when any of the tests is satisfied. If TOL is less than the machine precision (as defined by the function R1MACH(4)), then SNLS1E only attempts to satisfy the test defined by the machine precision. Further progress is not usually possible. Unless high precision solutions are required, the recommended value for TOL is the square root of the machine precision.

The tests assume that the functions are reasonably well behaved, and, if the Jacobian is supplied by the user, that the functions and the Jacobian are coded consistently. If these conditions are not satisfied, then SNLS1E may incorrectly indicate convergence. If the Jacobian is coded correctly or IOPT=1, then the validity of the answer can be checked, for example, by rerunning SNLS1E with tighter tolerances.

First Convergence Test. If ENORM(Z) denotes the Euclidean norm of a vector Z, then this test attempts to guarantee that

```
ENORM(FVEC) .LE. (1+TOL)*ENORM(FVECS),
```

where FVECS denotes the functions evaluated at XSOL. If this condition is satisfied with TOL = 10**(-K), then the final residual norm ENORM(FVEC) has K significant decimal digits and INFO is set to 1 (or to 3 if the second test is also satisfied).

Second Convergence Test. If D is a diagonal matrix (implicitly generated by SNLS1E) whose entries contain scale factors for the variables, then this test attempts to guarantee that

```
ENORM(D*(X-XSOL)) .LE.  TOL*ENORM(D*XSOL).
```

If this condition is satisfied with TOL = 10**(-K), then the larger components of D*X have K significant decimal digits and INFO is set to 2 (or to 3 if the first test is also satisfied). There is a danger that the smaller components of D*X may have large relative errors, but the choice of D is such that the accuracy of the components of X is usually related to their sensitivity.

Third Convergence Test. This test is satisfied when FVEC is orthogonal to the columns of the Jacobian to machine precision. There is no clear relationship between this test and the accuracy of SNLS1E, and furthermore, the test is equally well satisfied at other critical points, namely maximizers and saddle points. Therefore, termination caused by this test (INFO = 4) should be examined carefully.

5. Unsuccessful Completion.

Unsuccessful termination of SNLS1E can be due to improper input parameters, arithmetic interrupts, or an excessive number of function evaluations.

Improper Input Parameters. INFO is set to 0 if IOPT .LT. 1 or IOPT .GT. 3, or N .LE. 0, or M .LT. N, or TOL .LT. 0.E0, or for IOPT=1 or 2 LWA .LT. N*(M+5)+M, or for IOPT=3 LWA .LT. N*(N+5)+M.

Arithmetic Interrupts. If these interrupts occur in the FCN subroutine during an early stage of the computation, they may be caused by an unacceptable choice of X by SNLS1E. In this case, it may be possible to remedy the situation by not evaluating the functions here, but instead setting the components of FVEC to numbers that exceed those in the initial FVEC.

Excessive Number of Function Evaluations. If the number of calls to FCN reaches 100*(N+1) for IOPT=2 or 3 or 200*(N+1) for IOPT=1, then this indicates that the routine is converging very slowly as measured by the progress of FVEC, and INFO is set to 5. In this case, it may be helpful to restart SNLS1E, thereby forcing it to disregard old (and possibly harmful) information.

6. Characteristics of the Algorithm.

SNLS1E is a modification of the Levenberg-Marquardt algorithm. Two of its main characteristics involve the proper use of implicitly scaled variables and an optimal choice for the correction. The use of implicitly scaled variables achieves scale invariance of SNLS1E and limits the size of the correction in any direction where the functions are changing rapidly. The optimal choice of the correction guarantees (under reasonable conditions) global convergence from starting points far from the solution and a fast rate of convergence for problems with small residuals.

Timing. The time required by SNLS1E to solve a given problem depends on M and N, the behavior of the functions, the accuracy requested, and the starting point. The number of arithmetic operations needed by SNLS1E is about N**3 to process each evaluation of the functions (call to FCN) and to process each evaluation of the Jacobian SNLS1E takes M*N**2 for IOPT=2 (one call to JAC), M*N**2 for IOPT=1 (N calls to FCN) and 1.5*M*N**2 for IOPT=3 (M calls to FCN). Unless FCN can be evaluated quickly, the timing of SNLS1E will be strongly influenced by the time spent in FCN.

Storage. SNLS1E requires (M*N + 2*M + 6*N) for IOPT=1 or 2 and (N**2 + 2*M + 6*N) for IOPT=3 single precision storage locations and N integer storage locations, in addition to the storage required by the program. There are no internally declared storage arrays.

*Long Description:

7. Example.

The problem is to determine the values of X(1), X(2), and X(3) which provide the best fit (in the least squares sense) of

```
X(1) + U(I)/(V(I)*X(2) + W(I)*X(3)),  I = 1, 15
```

to the data

```
Y = (0.14,0.18,0.22,0.25,0.29,0.32,0.35,0.39,
     0.37,0.58,0.73,0.96,1.34,2.10,4.39),
```

```
      where U(I) = I, V(I) = 16 - I, and W(I) = MIN(U(I),V(I)).  The
      I-th component of FVEC is thus defined by

            Y(I) - (X(1) + U(I)/(V(I)*X(2) + W(I)*X(3))).

      **********

      PROGRAM TEST
C
C     Driver for SNLS1E example.
C
      INTEGER I,IOPT,M,N,NPRINT,JNFO,LWA,NWRITE
      INTEGER IW(3)
      REAL TOL,FNORM
      REAL X(3),FVEC(15),WA(75)
      REAL ENORM,R1MACH
      EXTERNAL FCN
      DATA NWRITE /6/
C
      IOPT = 1
      M = 15
      N = 3
C
C     The following starting values provide a rough fit.
C
      X(1) = 1.E0
      X(2) = 1.E0
      X(3) = 1.E0
C
      LWA = 75
      NPRINT = 0
C
C     Set TOL to the square root of the machine precision.
C     Unless high precision solutions are required,
C     this is the recommended setting.
C
      TOL = SQRT(R1MACH(4))
C
      CALL SNLS1E(FCN,IOPT,M,N,X,FVEC,TOL,NPRINT,
     *            INFO,IW,WA,LWA)
      FNORM = ENORM(M,FVEC)
      WRITE (NWRITE,1000) FNORM,INFO,(X(J),J=1,N)
      STOP
 1000 FORMAT (5X,' FINAL L2 NORM OF THE RESIDUALS',E15.7 //
     *        5X,' EXIT PARAMETER',16X,I10 //
     *        5X,' FINAL APPROXIMATE SOLUTION' // 5X,3E15.7)
      END
      SUBROUTINE FCN(IFLAG,M,N,X,FVEC,DUM,IDUM)
C     This is the form of the FCN routine if IOPT=1,
C     that is, if the user does not calculate the Jacobian.
      INTEGER M,N,IFLAG
      REAL X(N),FVEC(M)
      INTEGER I
      REAL TMP1,TMP2,TMP3,TMP4
      REAL Y(15)
      DATA Y(1),Y(2),Y(3),Y(4),Y(5),Y(6),Y(7),Y(8),
     *     Y(9),Y(10),Y(11),Y(12),Y(13),Y(14),Y(15)
     *     /1.4E-1,1.8E-1,2.2E-1,2.5E-1,2.9E-1,3.2E-1,3.5E-1,3.9E-1,
     *      3.7E-1,5.8E-1,7.3E-1,9.6E-1,1.34E0,2.1E0,4.39E0/
```

```
C
      IF (IFLAG .NE. 0) GO TO 5
C
C     Insert print statements here when NPRINT is positive.
C
      RETURN
    5 CONTINUE
      DO 10 I = 1, M
         TMP1 = I
         TMP2 = 16 - I
         TMP3 = TMP1
         IF (I .GT. 8) TMP3 = TMP2
         FVEC(I) = Y(I) - (X(1) + TMP1/(X(2)*TMP2 + X(3)*TMP3))
   10    CONTINUE
      RETURN
      END
```

Results obtained with different compilers or machines may be slightly different.

```
FINAL L2 NORM OF THE RESIDUALS  0.9063596E-01

EXIT PARAMETER                          1

FINAL APPROXIMATE SOLUTION

 0.8241058E-01  0.1133037E+01  0.2343695E+01
```

For IOPT=2, FCN would be modified as follows to also calculate the full Jacobian when IFLAG=2.

```
      SUBROUTINE FCN(IFLAG,M,N,X,FVEC,FJAC,LDFJAC)
C
C     This is the form of the FCN routine if IOPT=2,
C     that is, if the user calculates the full Jacobian.
C
      INTEGER LDFJAC,M,N,IFLAG
      REAL X(N),FVEC(M)
      REAL FJAC(LDFJAC,N)
      INTEGER I
      REAL TMP1,TMP2,TMP3,TMP4
      REAL Y(15)
      DATA Y(1),Y(2),Y(3),Y(4),Y(5),Y(6),Y(7),Y(8),
     *     Y(9),Y(10),Y(11),Y(12),Y(13),Y(14),Y(15)
     *     /1.4E-1,1.8E-1,2.2E-1,2.5E-1,2.9E-1,3.2E-1,3.5E-1,3.9E-1,
     *      3.7E-1,5.8E-1,7.3E-1,9.6E-1,1.34E0,2.1E0,4.39E0/
C
      IF (IFLAG .NE. 0) GO TO 5
C
C     Insert print statements here when NPRINT is positive.
C
      RETURN
    5 CONTINUE
      IF(IFLAG.NE.1) GO TO 20
      DO 10 I = 1, M
         TMP1 = I
         TMP2 = 16 - I
         TMP3 = TMP1
```

```
         IF (I .GT. 8) TMP3 = TMP2
         FVEC(I) = Y(I) - (X(1) + TMP1/(X(2)*TMP2 + X(3)*TMP3))
   10    CONTINUE
      RETURN
C
C     Below, calculate the full Jacobian.
C
   20    CONTINUE
C
      DO 30 I = 1, M
         TMP1 = I
         TMP2 = 16 - I
         TMP3 = TMP1
         IF (I .GT. 8) TMP3 = TMP2
         TMP4 = (X(2)*TMP2 + X(3)*TMP3)**2
         FJAC(I,1) = -1.E0
         FJAC(I,2) = TMP1*TMP2/TMP4
         FJAC(I,3) = TMP1*TMP3/TMP4
   30    CONTINUE
      RETURN
      END


      For IOPT = 3, FJAC would be dimensioned as FJAC(3,3),
        LDFJAC would be set to 3, and FCN would be written as
        follows to calculate a row of the Jacobian when IFLAG=3.

      SUBROUTINE FCN(IFLAG,M,N,X,FVEC,FJAC,LDFJAC)
C     This is the form of the FCN routine if IOPT=3,
C     that is, if the user calculates the Jacobian row by row.
      INTEGER M,N,IFLAG
      REAL X(N),FVEC(M)
      REAL FJAC(N)
      INTEGER I
      REAL TMP1,TMP2,TMP3,TMP4
      REAL Y(15)
      DATA Y(1),Y(2),Y(3),Y(4),Y(5),Y(6),Y(7),Y(8),
     *     Y(9),Y(10),Y(11),Y(12),Y(13),Y(14),Y(15)
     *     /1.4E-1,1.8E-1,2.2E-1,2.5E-1,2.9E-1,3.2E-1,3.5E-1,3.9E-1,
     *      3.7E-1,5.8E-1,7.3E-1,9.6E-1,1.34E0,2.1E0,4.39E0/
C
      IF (IFLAG .NE. 0) GO TO 5
C
C     Insert print statements here when NPRINT is positive.
C
      RETURN
    5 CONTINUE
      IF( IFLAG.NE.1) GO TO 20
      DO 10 I = 1, M
         TMP1 = I
         TMP2 = 16 - I
         TMP3 = TMP1
         IF (I .GT. 8) TMP3 = TMP2
         FVEC(I) = Y(I) - (X(1) + TMP1/(X(2)*TMP2 + X(3)*TMP3))
   10    CONTINUE
      RETURN
C
C     Below, calculate the LDFJAC-th row of the Jacobian.
C
   20 CONTINUE
```

```
      I = LDFJAC
         TMP1 = I
         TMP2 = 16 - I
         TMP3 = TMP1
         IF (I .GT. 8) TMP3 = TMP2
         TMP4 = (X(2)*TMP2 + X(3)*TMP3)**2
         FJAC(1) = -1.E0
         FJAC(2) = TMP1*TMP2/TMP4
         FJAC(3) = TMP1*TMP3/TMP4
      RETURN
      END
```

```
***REFERENCES  Jorge J. More, The Levenberg-Marquardt algorithm:
                 implementation and theory.  In Numerical Analysis
                 Proceedings (Dundee, June 28 - July 1, 1977, G. A.
                 Watson, Editor), Lecture Notes in Mathematics 630,
                 Springer-Verlag, 1978.
***ROUTINES CALLED  SNLS1, XERMSG
***REVISION HISTORY  (YYMMDD)
   800301  DATE WRITTEN
   890206  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SNRM2

```
      REAL FUNCTION SNRM2 (N, SX, INCX)
***BEGIN PROLOGUE  SNRM2
***PURPOSE  Compute the Euclidean length (L2 norm) of a vector.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A3B
***TYPE      SINGLE PRECISION (SNRM2-S, DNRM2-D, SCNRM2-C)
***KEYWORDS  BLAS, EUCLIDEAN LENGTH, EUCLIDEAN NORM, L2,
             LINEAR ALGEBRA, UNITARY, VECTOR
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
           Kincaid, D. R., (U. of Texas)
           Krogh, F. T., (JPL)
***DESCRIPTION

                B L A S  Subprogram
    Description of Parameters

     --Input--
        N  number of elements in input vector(s)
       SX  single precision vector with N elements
     INCX  storage spacing between elements of SX

     --Output--
    SNRM2  single precision result (zero if N .LE. 0)

     Euclidean norm of the N-vector stored in SX with storage
     increment INCX .
     If N .LE. 0, return with result = 0.
     If N .GE. 1, then INCX must be .GE. 1

     Four Phase Method using two built-in constants that are
     hopefully applicable to all machines.
         CUTLO = maximum of  SQRT(U/EPS)  over all known machines.
         CUTHI = minimum of  SQRT(V)      over all known machines.
     where
         EPS = smallest no. such that EPS + 1. .GT. 1.
         U   = smallest positive no.   (underflow limit)
         V   = largest  no.            (overflow  limit)

     Brief Outline of Algorithm.

     Phase 1 scans zero components.
     Move to phase 2 when a component is nonzero and .LE. CUTLO
     Move to phase 3 when a component is .GT. CUTLO
     Move to phase 4 when a component is .GE. CUTHI/M
     where M = N for X() real and M = 2*N for complex.

     Values for CUTLO and CUTHI.
     From the environmental parameters listed in the IMSL converter
     document the limiting values are as follows:
     CUTLO, S.P.   U/EPS = 2**(-102) for  Honeywell.  Close seconds are
                   Univac and DEC at 2**(-103)
                   Thus CUTLO = 2**(-51) = 4.44089E-16
     CUTHI, S.P.   V = 2**127 for Univac, Honeywell, and DEC.
                   Thus CUTHI = 2**(63.5) = 1.30438E19
     CUTLO, D.P.   U/EPS = 2**(-67) for Honeywell and DEC.
```

```
                    Thus CUTLO = 2**(-33.5) = 8.23181D-11
     CUTHI, D.P.   same as S.P.  CUTHI = 1.30438D19
     DATA CUTLO, CUTHI /8.232D-11,  1.304D19/
     DATA CUTLO, CUTHI /4.441E-16,  1.304E19/

***REFERENCES  C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T.
                 Krogh, Basic linear algebra subprograms for Fortran
                 usage, Algorithm No. 539, Transactions on Mathematical
                 Software 5, 3 (September 1979), pp. 308-323.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   791001  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SNSQ

```
      SUBROUTINE SNSQ (FCN, JAC, IOPT, N, X, FVEC, FJAC, LDFJAC, XTOL,
         MAXFEV, ML, MU, EPSFCN, DIAG, MODE, FACTOR, NPRINT, INFO, NFEV,
     +   NJEV, R, LR, QTF, WA1, WA2, WA3, WA4)
***BEGIN PROLOGUE  SNSQ
***PURPOSE  Find a zero of a system of a N nonlinear functions in N
            variables by a modification of the Powell hybrid method.
***LIBRARY   SLATEC
***CATEGORY  F2A
***TYPE      SINGLE PRECISION (SNSQ-S, DNSQ-D)
***KEYWORDS  NONLINEAR SQUARE SYSTEM, POWELL HYBRID METHOD, ZEROS
***AUTHOR  Hiebert, K. L., (SNLA)
***DESCRIPTION

 1. Purpose.

       The purpose of SNSQ is to find a zero of a system of N non-
       linear functions in N variables by a modification of the Powell
       hybrid method.  The user must provide a subroutine which calcu-
       lates the functions.  The user has the option of either to
       provide a subroutine which calculates the Jacobian or to let the
       code calculate it by a forward-difference approximation.
       This code is the combination of the MINPACK codes (Argonne)
       HYBRD and HYBRDJ.


 2. Subroutine and Type Statements.

       SUBROUTINE SNSQ(FCN,JAC,IOPT,N,X,FVEC,FJAC,LDFJAC,XTOL,MAXFEV,
      *                ML,MU,EPSFCN,DIAG,MODE,FACTOR,NPRINT,INFO,NFEV,
      *                NJEV,R,LR,QTF,WA1,WA2,WA3,WA4)
       INTEGER IOPT,N,MAXFEV,ML,MU,MODE,NPRINT,INFO,NFEV,LDFJAC,NJEV,LR
       REAL XTOL,EPSFCN,FACTOR
       REAL X(N),FVEC(N),DIAG(N),FJAC(LDFJAC,N),R(LR),QTF(N),
      *     WA1(N),WA2(N),WA3(N),WA4(N)
       EXTERNAL FCN,JAC


 3. Parameters.

       Parameters designated as input parameters must be specified on
       entry to SNSQ and are not changed on exit, while parameters
       designated as output parameters need not be specified on entry
       and are set to appropriate values on exit from SNSQ.

       FCN is the name of the user-supplied subroutine which calculates
         the functions.  FCN must be declared in an EXTERNAL statement
         in the user calling program, and should be written as follows.

         SUBROUTINE FCN(N,X,FVEC,IFLAG)
         INTEGER N,IFLAG
         REAL X(N),FVEC(N)
         ----------
         Calculate the functions at X and
         return this vector in FVEC.
         ----------
         RETURN
```

```
END
```

The value of IFLAG should not be changed by FCN unless the user wants to terminate execution of SNSQ. In this case, set IFLAG to a negative integer.

JAC is the name of the user-supplied subroutine which calculates the Jacobian. If IOPT=1, then JAC must be declared in an EXTERNAL statement in the user calling program, and should be written as follows.

```
SUBROUTINE JAC(N,X,FVEC,FJAC,LDFJAC,IFLAG)
INTEGER N,LDFJAC,IFLAG
REAL X(N),FVEC(N),FJAC(LDFJAC,N)
----------
Calculate the Jacobian at X and return this
matrix in FJAC.  FVEC contains the function
values at X and should not be altered.
----------
RETURN
END
```

The value of IFLAG should not be changed by JAC unless the user wants to terminate execution of SNSQ. In this case, set IFLAG to a negative integer.

If IOPT=2, JAC can be ignored (treat it as a dummy argument).

IOPT is an input variable which specifies how the Jacobian will be calculated. If IOPT=1, then the user must supply the Jacobian through the subroutine JAC. If IOPT=2, then the code will approximate the Jacobian by forward-differencing.

N is a positive integer input variable set to the number of functions and variables.

X is an array of length N. On input, X must contain an initial estimate of the solution vector. On output, X contains the final estimate of the solution vector.

FVEC is an output array of length N which contains the functions evaluated at the output X.

FJAC is an output N by N array which contains the orthogonal matrix Q produced by the QR factorization of the final approximate Jacobian.

LDFJAC is a positive integer input variable not less than N which specifies the leading dimension of the array FJAC.

XTOL is a non-negative input variable. Termination occurs when the relative error between two consecutive iterates is at most XTOL. Therefore, XTOL measures the relative error desired in the approximate solution. Section 4 contains more details about XTOL.

MAXFEV is a positive integer input variable. Termination occurs when the number of calls to FCN is at least MAXFEV by the end of an iteration.

ML is a non-negative integer input variable which specifies the number of subdiagonals within the band of the Jacobian matrix. If the Jacobian is not banded or IOPT=1, set ML to at least N - 1.

MU is a non-negative integer input variable which specifies the number of superdiagonals within the band of the Jacobian matrix. If the Jacobian is not banded or IOPT=1, set MU to at least N - 1.

EPSFCN is an input variable used in determining a suitable step for the forward-difference approximation. This approximation assumes that the relative errors in the functions are of the order of EPSFCN. If EPSFCN is less than the machine precision, it is assumed that the relative errors in the functions are of the order of the machine precision. If IOPT=1, then EPSFCN can be ignored (treat it as a dummy argument).

DIAG is an array of length N. If MODE = 1 (see below), DIAG is internally set. If MODE = 2, DIAG must contain positive entries that serve as implicit (multiplicative) scale factors for the variables.

MODE is an integer input variable. If MODE = 1, the variables will be scaled internally. If MODE = 2, the scaling is specified by the input DIAG. Other values of MODE are equivalent to MODE = 1.

FACTOR is a positive input variable used in determining the initial step bound. This bound is set to the product of FACTOR and the Euclidean norm of DIAG*X if nonzero, or else to FACTOR itself. In most cases FACTOR should lie in the interval (.1,100.). 100. is a generally recommended value.

NPRINT is an integer input variable that enables controlled printing of iterates if it is positive. In this case, FCN is called with IFLAG = 0 at the beginning of the first iteration and every NPRINT iteration thereafter and immediately prior to return, with X and FVEC available for printing. Appropriate print statements must be added to FCN(see example). If NPRINT is not positive, no special calls of FCN with IFLAG = 0 are made.

INFO is an integer output variable. If the user has terminated execution, INFO is set to the (negative) value of IFLAG. See description of FCN and JAC. Otherwise, INFO is set as follows.

INFO = 0 improper input parameters.

INFO = 1 relative error between two consecutive iterates is at most XTOL.

INFO = 2 number of calls to FCN has reached or exceeded MAXFEV.

INFO = 3 XTOL is too small. No further improvement in the approximate solution X is possible.

INFO = 4 iteration is not making good progress, as measured by the improvement from the last five Jacobian eval-

uations.

INFO = 5 iteration is not making good progress, as measured by the improvement from the last ten iterations.

Sections 4 and 5 contain more details about INFO.

NFEV is an integer output variable set to the number of calls to FCN.

NJEV is an integer output variable set to the number of calls to JAC. (If IOPT=2, then NJEV is set to zero.)

R is an output array of length LR which contains the upper triangular matrix produced by the QR factorization of the final approximate Jacobian, stored rowwise.

LR is a positive integer input variable not less than (N*(N+1))/2.

QTF is an output array of length N which contains the vector (Q TRANSPOSE)*FVEC.

WA1, WA2, WA3, and WA4 are work arrays of length N.

4. Successful Completion.

The accuracy of SNSQ is controlled by the convergence parameter XTOL. This parameter is used in a test which makes a comparison between the approximation X and a solution XSOL. SNSQ terminates when the test is satisfied. If the convergence parameter is less than the machine precision (as defined by the function R1MACH(4)), then SNSQ only attempts to satisfy the test defined by the machine precision. Further progress is not usually possible.

The test assumes that the functions are reasonably well behaved, and, if the Jacobian is supplied by the user, that the functions and the Jacobian are coded consistently. If these conditions are not satisfied, then SNSQ may incorrectly indicate convergence. The coding of the Jacobian can be checked by the subroutine CHKDER. If the Jacobian is coded correctly or IOPT=2, then the validity of the answer can be checked, for example, by rerunning SNSQ with a tighter tolerance.

Convergence Test. If ENORM(Z) denotes the Euclidean norm of a vector Z and D is the diagonal matrix whose entries are defined by the array DIAG, then this test attempts to guarantee that

```
ENORM(D*(X-XSOL)) .LE. XTOL*ENORM(D*XSOL).
```

If this condition is satisfied with XTOL = 10**(-K), then the larger components of D*X have K significant decimal digits and INFO is set to 1. There is a danger that the smaller components of D*X may have large relative errors, but the fast rate of convergence of SNSQ usually avoids this possibility. Unless high precision solutions are required, the recommended value for XTOL is the square root of the machine precision.

5. Unsuccessful Completion.

Unsuccessful termination of SNSQ can be due to improper input parameters, arithmetic interrupts, an excessive number of function evaluations, or lack of good progress.

Improper Input Parameters. INFO is set to 0 if IOPT .LT. 1, or IOPT .GT. 2, or N .LE. 0, or LDFJAC .LT. N, or XTOL .LT. 0.E0, or MAXFEV .LE. 0, or ML .LT. 0, or MU .LT. 0, or FACTOR .LE. 0.E0, or LR .LT. (N*(N+1))/2.

Arithmetic Interrupts. If these interrupts occur in the FCN subroutine during an early stage of the computation, they may be caused by an unacceptable choice of X by SNSQ. In this case, it may be possible to remedy the situation by rerunning SNSQ with a smaller value of FACTOR.

Excessive Number of Function Evaluations. A reasonable value for MAXFEV is 100*(N+1) for IOPT=1 and 200*(N+1) for IOPT=2. If the number of calls to FCN reaches MAXFEV, then this indicates that the routine is converging very slowly as measured by the progress of FVEC, and INFO is set to 2. This situation should be unusual because, as indicated below, lack of good progress is usually diagnosed earlier by SNSQ, causing termination with INFO = 4 or INFO = 5.

Lack of Good Progress. SNSQ searches for a zero of the system by minimizing the sum of the squares of the functions. In so doing, it can become trapped in a region where the minimum does not correspond to a zero of the system and, in this situation, the iteration eventually fails to make good progress. In particular, this will happen if the system does not have a zero. If the system has a zero, rerunning SNSQ from a different starting point may be helpful.

6. Characteristics of the Algorithm.

SNSQ is a modification of the Powell hybrid method. Two of its main characteristics involve the choice of the correction as a convex combination of the Newton and scaled gradient directions, and the updating of the Jacobian by the rank-1 method of Broyden. The choice of the correction guarantees (under reasonable conditions) global convergence for starting points far from the solution and a fast rate of convergence. The Jacobian is calculated at the starting point by either the user-supplied subroutine or a forward-difference approximation, but it is not recalculated until the rank-1 method fails to produce satisfactory progress.

Timing. The time required by SNSQ to solve a given problem depends on N, the behavior of the functions, the accuracy requested, and the starting point. The number of arithmetic operations needed by SNSQ is about 11.5*(N**2) to process each evaluation of the functions (call to FCN) and 1.3*(N**3) to process each evaluation of the Jacobian (call to JAC, if IOPT = 1). Unless FCN and JAC can be evaluated quickly, the timing of SNSQ will be strongly influenced by the time

spent in FCN and JAC.

Storage. SNSQ requires (3*N**2 + 17*N)/2 single precision storage locations, in addition to the storage required by the program. There are no internally declared storage arrays.

7. Example.

The problem is to determine the values of X(1), X(2), ..., X(9), which solve the system of tridiagonal equations

```
(3-2*X(1))*X(1)           -2*X(2)                    = -1
        -X(I-1) + (3-2*X(I))*X(I)         -2*X(I+1) = -1, I=2-8
                                -X(8) + (3-2*X(9))*X(9) = -1
```

```
C     **********

      PROGRAM TEST
C
C     Driver for SNSQ example.
C
      INTEGER J,IOPT,N,MAXFEV,ML,MU,MODE,NPRINT,INFO,NFEV,LDFJAC,LR,
     *        NWRITE
      REAL XTOL,EPSFCN,FACTOR,FNORM
      REAL X(9),FVEC(9),DIAG(9),FJAC(9,9),R(45),QTF(9),
     *     WA1(9),WA2(9),WA3(9),WA4(9)
      REAL ENORM,R1MACH
      EXTERNAL FCN
      DATA NWRITE /6/
C
      IOPT = 2
      N = 9
C
C     The following starting values provide a rough solution.
C
      DO 10 J = 1, 9
         X(J) = -1.E0
   10    CONTINUE
C
      LDFJAC = 9
      LR = 45
C
C     Set XTOL to the square root of the machine precision.
C     Unless high precision solutions are required,
C     this is the recommended setting.
C
      XTOL = SQRT(R1MACH(4))
C
      MAXFEV = 2000
      ML = 1
      MU = 1
      EPSFCN = 0.E0
      MODE = 2
      DO 20 J = 1, 9
         DIAG(J) = 1.E0
   20    CONTINUE
      FACTOR = 1.E2
      NPRINT = 0
C
      CALL SNSQ(FCN,JAC,IOPT,N,X,FVEC,FJAC,LDFJAC,XTOL,MAXFEV,ML,MU,
```

```
     *           EPSFCN,DIAG,MODE,FACTOR,NPRINT,INFO,NFEV,NJEV,
     *           R,LR,QTF,WA1,WA2,WA3,WA4)
      FNORM = ENORM(N,FVEC)
      WRITE (NWRITE,1000) FNORM,NFEV,INFO,(X(J),J=1,N)
      STOP
 1000 FORMAT (5X,' FINAL L2 NORM OF THE RESIDUALS',E15.7 //
     *        5X,' NUMBER OF FUNCTION EVALUATIONS',I10 //
     *        5X,' EXIT PARAMETER',16X,I10 //
     *        5X,' FINAL APPROXIMATE SOLUTION' // (5X,3E15.7))
      END
      SUBROUTINE FCN(N,X,FVEC,IFLAG)
      INTEGER N,IFLAG
      REAL X(N),FVEC(N)
      INTEGER K
      REAL ONE,TEMP,TEMP1,TEMP2,THREE,TWO,ZERO
      DATA ZERO,ONE,TWO,THREE /0.E0,1.E0,2.E0,3.E0/
C
      IF (IFLAG .NE. 0) GO TO 5
C
C     Insert print statements here when NPRINT is positive.
C
      RETURN
    5 CONTINUE
      DO 10 K = 1, N
         TEMP = (THREE - TWO*X(K))*X(K)
         TEMP1 = ZERO
         IF (K .NE. 1) TEMP1 = X(K-1)
         TEMP2 = ZERO
         IF (K .NE. N) TEMP2 = X(K+1)
         FVEC(K) = TEMP - TEMP1 - TWO*TEMP2 + ONE
   10    CONTINUE
      RETURN
      END

      Results obtained with different compilers or machines
      may be slightly different.

      FINAL L2 NORM OF THE RESIDUALS  0.1192636E-07

      NUMBER OF FUNCTION EVALUATIONS        14

      EXIT PARAMETER                         1

      FINAL APPROXIMATE SOLUTION

      -0.5706545E+00 -0.6816283E+00 -0.7017325E+00
      -0.7042129E+00 -0.7013690E+00 -0.6918656E+00
      -0.6657920E+00 -0.5960342E+00 -0.4164121E+00

***REFERENCES  M. J. D. Powell, A hybrid method for nonlinear equa-
                tions. In Numerical Methods for Nonlinear Algebraic
                Equations, P. Rabinowitz, Editor.  Gordon and Breach,
                1988.
***ROUTINES CALLED  DOGLEG, ENORM, FDJAC1, QFORM, QRFAC, R1MACH,
                    R1MPYQ, R1UPDT, XERMSG
***REVISION HISTORY  (YYMMDD)
   800301  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
```

```
891214  Prologue converted to Version 4.0 format.  (BAB)
900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

SNSQE

```
      SUBROUTINE SNSQE (FCN, JAC, IOPT, N, X, FVEC, TOL, NPRINT, INFO,
     +   WA, LWA)
***BEGIN PROLOGUE  SNSQE
***PURPOSE  An easy-to-use code to find a zero of a system of N
            nonlinear functions in N variables by a modification of
            the Powell hybrid method.
***LIBRARY   SLATEC
***CATEGORY  F2A
***TYPE      SINGLE PRECISION (SNSQE-S, DNSQE-D)
***KEYWORDS  EASY-TO-USE, NONLINEAR SQUARE SYSTEM,
             POWELL HYBRID METHOD, ZEROS
***AUTHOR  Hiebert, K. L., (SNLA)
***DESCRIPTION
```

1. Purpose.

The purpose of SNSQE is to find a zero of a system of N non-linear functions in N variables by a modification of the Powell hybrid method. This is done by using the more general nonlinear equation solver SNSQ. The user must provide a subroutine which calculates the functions. The user has the option of either to provide a subroutine which calculates the Jacobian or to let the code calculate it by a forward-difference approximation. This code is the combination of the MINPACK codes (Argonne) HYBRD1 and HYBRJ1.

2. Subroutine and Type Statements.

```
      SUBROUTINE SNSQE(FCN,JAC,IOPT,N,X,FVEC,TOL,NPRINT,INFO,
     *                 WA,LWA)
      INTEGER IOPT,N,NPRINT,INFO,LWA
      REAL TOL
      REAL X(N),FVEC(N),WA(LWA)
      EXTERNAL FCN,JAC
```

3. Parameters.

Parameters designated as input parameters must be specified on entry to SNSQE and are not changed on exit, while parameters designated as output parameters need not be specified on entry and are set to appropriate values on exit from SNSQE.

FCN is the name of the user-supplied subroutine which calculates the functions. FCN must be declared in an EXTERNAL statement in the user calling program, and should be written as follows.

```
SUBROUTINE FCN(N,X,FVEC,IFLAG)
INTEGER N,IFLAG
REAL X(N),FVEC(N)
----------
Calculate the functions at X and
return this vector in FVEC.
----------
```

```
RETURN
END
```

The value of IFLAG should not be changed by FCN unless the user wants to terminate execution of SNSQE. In this case, set IFLAG to a negative integer.

JAC is the name of the user-supplied subroutine which calculates the Jacobian. If IOPT=1, then JAC must be declared in an EXTERNAL statement in the user calling program, and should be written as follows.

```
SUBROUTINE JAC(N,X,FVEC,FJAC,LDFJAC,IFLAG)
INTEGER N,LDFJAC,IFLAG
REAL X(N),FVEC(N),FJAC(LDFJAC,N)
----------
Calculate the Jacobian at X and return this
matrix in FJAC.  FVEC contains the function
values at X and should not be altered.
----------
RETURN
END
```

The value of IFLAG should not be changed by JAC unless the user wants to terminate execution of SNSQE. In this case, set IFLAG to a negative integer.

If IOPT=2, JAC can be ignored (treat it as a dummy argument).

IOPT is an input variable which specifies how the Jacobian will be calculated. If IOPT=1, then the user must supply the Jacobian through the subroutine JAC. If IOPT=2, then the code will approximate the Jacobian by forward-differencing.

N is a positive integer input variable set to the number of functions and variables.

X is an array of length N. On input, X must contain an initial estimate of the solution vector. On output, X contains the final estimate of the solution vector.

FVEC is an output array of length N which contains the functions evaluated at the output X.

TOL is a non-negative input variable. Termination occurs when the algorithm estimates that the relative error between X and the solution is at most TOL. Section 4 contains more details about TOL.

NPRINT is an integer input variable that enables controlled printing of iterates if it is positive. In this case, FCN is called with IFLAG = 0 at the beginning of the first iteration and every NPRINT iteration thereafter and immediately prior to return, with X and FVEC available for printing. Appropriate print statements must be added to FCN (see example). If NPRINT is not positive, no special calls of FCN with IFLAG = 0 are made.

INFO is an integer output variable. If the user has terminated execution, INFO is set to the (negative) value of IFLAG. See

description of FCN and JAC. Otherwise, INFO is set as follows.

INFO = 0 improper input parameters.

INFO = 1 algorithm estimates that the relative error between X and the solution is at most TOL.

INFO = 2 number of calls to FCN has reached or exceeded 100*(N+1) for IOPT=1 or 200*(N+1) for IOPT=2.

INFO = 3 TOL is too small. No further improvement in the approximate solution X is possible.

INFO = 4 iteration is not making good progress.

Sections 4 and 5 contain more details about INFO.

WA is a work array of length LWA.

LWA is a positive integer input variable not less than (3*N**2+13*N))/2.

4. Successful Completion.

The accuracy of SNSQE is controlled by the convergence parameter TOL. This parameter is used in a test which makes a comparison between the approximation X and a solution XSOL. SNSQE terminates when the test is satisfied. If TOL is less than the machine precision (as defined by the function R1MACH(4)), then SNSQE attempts only to satisfy the test defined by the machine precision. Further progress is not usually possible. Unless high precision solutions are required, the recommended value for TOL is the square root of the machine precision.

The test assumes that the functions are reasonably well behaved, and, if the Jacobian is supplied by the user, that the functions and the Jacobian coded consistently. If these conditions are not satisfied, SNSQE may incorrectly indicate convergence. The coding of the Jacobian can be checked by the subroutine CHKDER. If the Jacobian is coded correctly or IOPT=2, then the validity of the answer can be checked, for example, by rerunning SNSQE with a tighter tolerance.

Convergence Test. If ENORM(Z) denotes the Euclidean norm of a vector Z, then this test attempts to guarantee that

```
ENORM(X-XSOL) .LE.  TOL*ENORM(XSOL).
```

If this condition is satisfied with TOL = 10**(-K), then the larger components of X have K significant decimal digits and INFO is set to 1. There is a danger that the smaller components of X may have large relative errors, but the fast rate of convergence of SNSQE usually avoids this possibility.

5. Unsuccessful Completion.

Unsuccessful termination of SNSQE can be due to improper input parameters, arithmetic interrupts, an excessive number of func-

tion evaluations, errors in the functions, or lack of good progress.

Improper Input Parameters. INFO is set to 0 if IOPT .LT. 1, or IOPT .GT. 2, or N .LE. 0, or TOL .LT. 0.E0, or LWA .LT. (3*N**2+13*N)/2.

Arithmetic Interrupts. If these interrupts occur in the FCN subroutine during an early stage of the computation, they may be caused by an unacceptable choice of X by SNSQE. In this case, it may be possible to remedy the situation by not evaluating the functions here, but instead setting the components of FVEC to numbers that exceed those in the initial FVEC.

Excessive Number of Function Evaluations. If the number of calls to FCN reaches 100*(N+1) for IOPT=1 or 200*(N+1) for IOPT=2, then this indicates that the routine is converging very slowly as measured by the progress of FVEC, and INFO is set to 2. This situation should be unusual because, as indicated below, lack of good progress is usually diagnosed earlier by SNSQE, causing termination with INFO = 4.

Errors in the Functions. When IOPT=2, the choice of step length in the forward-difference approximation to the Jacobian assumes that the relative errors in the functions are of the order of the machine precision. If this is not the case, SNSQE may fail (usually with INFO = 4). The user should then either use SNSQ and set the step length or use IOPT=1 and supply the Jacobian.

Lack of Good Progress. SNSQE searches for a zero of the system by minimizing the sum of the squares of the functions. In so doing, it can become trapped in a region where the minimum does not correspond to a zero of the system and, in this situation, the iteration eventually fails to make good progress. In particular, this will happen if the system does not have a zero. If the system has a zero, rerunning SNSQE from a different starting point may be helpful.

6. Characteristics of the Algorithm.

SNSQE is a modification of the Powell hybrid method. Two of its main characteristics involve the choice of the correction as a convex combination of the Newton and scaled gradient directions, and the updating of the Jacobian by the rank-1 method of Broyden. The choice of the correction guarantees (under reasonable conditions) global convergence for starting points far from the solution and a fast rate of convergence. The Jacobian is calculated at the starting point by either the user-supplied subroutine or a forward-difference approximation, but it is not recalculated until the rank-1 method fails to produce satisfactory progress.

Timing. The time required by SNSQE to solve a given problem depends on N, the behavior of the functions, the accuracy requested, and the starting point. The number of arithmetic operations needed by SNSQE is about 11.5*(N**2) to process each evaluation of the functions (call to FCN) and 1.3*(N**3) to process each evaluation of the Jacobian (call to JAC,

if IOPT = 1). Unless FCN and JAC can be evaluated quickly, the timing of SNSQE will be strongly influenced by the time spent in FCN and JAC.

Storage. SNSQE requires (3*N**2 + 17*N)/2 single precision storage locations, in addition to the storage required by the program. There are no internally declared storage arrays.

7. Example.

The problem is to determine the values of X(1), X(2), ..., X(9), which solve the system of tridiagonal equations

```
(3-2*X(1))*X(1)           -2*X(2)                    = -1
        -X(I-1) + (3-2*X(I))*X(I)         -2*X(I+1) = -1, I=2-8
                            -X(8) + (3-2*X(9))*X(9) = -1
```

```
**********

      PROGRAM TEST
C
C     Driver for SNSQE example.
C
      INTEGER J,N,IOPT,NPRINT,INFO,LWA,NWRITE
      REAL TOL,FNORM
      REAL X(9),FVEC(9),WA(180)
      REAL ENORM,R1MACH
      EXTERNAL FCN
      DATA NWRITE /6/
C
      IOPT = 2
      N = 9
C
C     The following starting values provide a rough solution.
C
      DO 10 J = 1, 9
         X(J) = -1.E0
   10    CONTINUE

      LWA = 180
      NPRINT = 0
C
C     Set TOL to the square root of the machine precision.
C     Unless high precision solutions are required,
C     this is the recommended setting.
C
      TOL = SQRT(R1MACH(4))
C
      CALL SNSQE(FCN,JAC,IOPT,N,X,FVEC,TOL,NPRINT,INFO,WA,LWA)
      FNORM = ENORM(N,FVEC)
      WRITE (NWRITE,1000) FNORM,INFO,(X(J),J=1,N)
      STOP
 1000 FORMAT (5X,' FINAL L2 NORM OF THE RESIDUALS',E15.7 //
     *        5X,' EXIT PARAMETER',16X,I10 //
     *        5X,' FINAL APPROXIMATE SOLUTION' // (5X,3E15.7))
      END
      SUBROUTINE FCN(N,X,FVEC,IFLAG)
      INTEGER N,IFLAG
      REAL X(N),FVEC(N)
```

```
      INTEGER K
      REAL ONE,TEMP,TEMP1,TEMP2,THREE,TWO,ZERO
      DATA ZERO,ONE,TWO,THREE /0.E0,1.E0,2.E0,3.E0/
C
      DO 10 K = 1, N
         TEMP = (THREE - TWO*X(K))*X(K)
         TEMP1 = ZERO
         IF (K .NE. 1) TEMP1 = X(K-1)
         TEMP2 = ZERO
         IF (K .NE. N) TEMP2 = X(K+1)
         FVEC(K) = TEMP - TEMP1 - TWO*TEMP2 + ONE
   10    CONTINUE
      RETURN
      END

      Results obtained with different compilers or machines
      may be slightly different.

      FINAL L2 NORM OF THE RESIDUALS  0.1192636E-07

      EXIT PARAMETER                        1

      FINAL APPROXIMATE SOLUTION

      -0.5706545E+00 -0.6816283E+00 -0.7017325E+00
      -0.7042129E+00 -0.7013690E+00 -0.6918656E+00
      -0.6657920E+00 -0.5960342E+00 -0.4164121E+00

***REFERENCES  M. J. D. Powell, A hybrid method for nonlinear equa-
                tions. In Numerical Methods for Nonlinear Algebraic
                Equations, P. Rabinowitz, Editor.  Gordon and Breach,
                1988.
***ROUTINES CALLED  SNSQ, XERMSG
***REVISION HISTORY  (YYMMDD)
   800301  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SOMN

```
      SUBROUTINE SOMN(N, B, X, NELT, IA, JA, A, ISYM, MATVEC, MSOLVE,
     +     NSAVE, ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT, R,
     +     Z, P, AP, EMAP, DZ, CSAV, RWORK, IWORK)
***BEGIN PROLOGUE  SOMN
***PURPOSE  Preconditioned Orthomin Sparse Iterative Ax=b Solver.
            Routine to solve a general linear system  Ax = b  using
            the Preconditioned Orthomin method.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SOMN-S, DOMN-D)
***KEYWORDS  ITERATIVE PRECONDITION, NON-SYMMETRIC LINEAR SYSTEM,
             ORTHOMIN, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER  N, NELT, IA(NELT), JA(NELT), ISYM, NSAVE, ITOL, ITMAX
     INTEGER  ITER, IERR, IUNIT, IWORK(USER DEFINED)
     REAL     B(N), X(N), A(NELT), TOL, ERR, R(N), Z(N)
     REAL     P(N,0:NSAVE), AP(N,0:NSAVE), EMAP(N,0:NSAVE)
     REAL     DZ(N), CSAV(NSAVE), RWORK(USER DEFINED)
     EXTERNAL MATVEC, MSOLVE

     CALL SOMN(N, B, X, NELT, IA, JA, A, ISYM, MATVEC, MSOLVE,
    $     NSAVE, ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT, R,
    $     Z, P, AP, EMAP, DZ, CSAV, RWORK, IWORK)

 *Arguments:
 N      :IN       Integer.
         Order of the Matrix.
 B      :IN       Real B(N).
         Right-hand side vector.
 X      :INOUT    Real X(N).
         On input X is your initial guess for solution vector.
         On output X is the final approximate solution.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :IN       Integer IA(NELT).
 JA     :IN       Integer JA(NELT).
 A      :IN       Real A(NELT).
         These arrays contain the matrix data structure for A.
         It could take any form.  See "Description", below, for more
         details.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 MATVEC :EXT      External.
         Name of a routine which performs the matrix vector multiply
         Y = A*X given A and X.  The name of the MATVEC routine must
```

```
        be declared external in the calling program.  The calling
        sequence to MATVEC is:
            CALL MATVEC( N, X, Y, NELT, IA, JA, A, ISYM )
        Where N is the number of unknowns, Y is the product A*X
        upon return X is an input vector, NELT is the number of
        non-zeros in the SLAP IA, JA, A storage for the matrix A.
        ISYM is a flag which, if non-zero, denotest that A is
        symmetric and only the lower or upper triangle is stored.
MSOLVE :EXT      External.
        Name of a routine which solves a linear system MZ = R for
        Z given R with the preconditioning matrix M (M is supplied via
        RWORK and IWORK arrays).  The name of the MSOLVE routine must
        be declared external in the calling program.  The calling
        sequence to MSOLVE is:
            CALL MSOLVE(N, R, Z, NELT, IA, JA, A, ISYM, RWORK, IWORK)
        Where N is the number of unknowns, R is the right-hand side
        vector and Z is the solution upon return.  NELT, IA, JA, A and
        ISYM are defined as above.  RWORK is a real array that can
        be used to pass necessary preconditioning information and/or
        workspace to MSOLVE.  IWORK is an integer work array for
        the same purpose as RWORK.
NSAVE  :IN       Integer.
        Number of  direction vectors to save and orthogonalize
        against.  NSAVE >= 0.
ITOL   :IN       Integer.
        Flag to indicate type of convergence criterion.
        If ITOL=1, iteration stops when the 2-norm of the residual
        divided by the 2-norm of the right-hand side is less than TOL.
        If ITOL=2, iteration stops when the 2-norm of M-inv times the
        residual divided by the 2-norm of M-inv times the right hand
        side is less than TOL, where M-inv is the inverse of the
        diagonal of A.
        ITOL=11 is often useful for checking and comparing different
        routines.  For this case, the user must supply the "exact"
        solution or a very accurate approximation (one with an error
        much less than TOL) through a common block,
            COMMON /SSLBLK/ SOLN( )
        If ITOL=11, iteration stops when the 2-norm of the difference
        between the iterative approximation and the user-supplied
        solution divided by the 2-norm of the user-supplied solution
        is less than TOL.  Note that this requires the user to set up
        the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine.
        The routine with this declaration should be loaded before the
        stop test so that the correct length is used by the loader.
        This procedure is not standard Fortran and may not work
        correctly on your system (although it has worked on every
        system the authors have tried).  If ITOL is not 11 then this
        common block is indeed standard Fortran.
TOL    :INOUT    Real.
        Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX  :IN       Integer.
        Maximum number of iterations.
ITER   :OUT      Integer.
        Number of iterations required to reach convergence, or
        ITMAX+1 if convergence criterion could not be achieved in
        ITMAX iterations.
ERR    :OUT      Real.
        Error estimate of error in final approximate solution, as
        defined by ITOL.
IERR   :OUT      Integer.
```

```
        Return error flag.
          IERR = 0 => All went well.
          IERR = 1 => Insufficient space allocated for WORK or IWORK.
          IERR = 2 => Method failed to converge in ITMAX steps.
          IERR = 3 => Error in user input.
                      Check input values of N, ITOL.
          IERR = 4 => User error tolerance set too tight.
                      Reset to 500*R1MACH(3).  Iteration proceeded.
          IERR = 5 => Preconditioning matrix, M, is not positive
                      definite.  (r,z) < 0.
          IERR = 6 => Breakdown of method detected.
                      (p,Ap) < epsilon**2.
IUNIT  :IN       Integer.
        Unit number on which to write the error at each iteration,
        if this is desired for monitoring convergence.  If unit
        number is 0, no writing will occur.
R      :WORK     Real R(N).
Z      :WORK     Real Z(N).
P      :WORK     Real P(N,0:NSAVE).
AP     :WORK     Real AP(N,0:NSAVE).
EMAP   :WORK     Real EMAP(N,0:NSAVE).
DZ     :WORK     Real DZ(N).
CSAV   :WORK     Real CSAV(NSAVE)
        Real arrays used for workspace.
RWORK  :WORK     Real RWORK(USER DEFINED).
        Real array that can be used for workspace in MSOLVE.
IWORK  :WORK     Integer IWORK(USER DEFINED).
        Integer array that can be used for workspace in MSOLVE.

*Description
      This routine does  not care  what matrix data   structure is
      used for  A and M.  It simply   calls  the MATVEC and MSOLVE
      routines, with  the arguments as  described above.  The user
      could write any type of structure and the appropriate MATVEC
      and MSOLVE routines.  It is assumed  that A is stored in the
      IA, JA, A  arrays in some fashion and  that M (or INV(M)) is
      stored  in  IWORK  and  RWORK)  in  some fashion.   The SLAP
      routines SSDOMN and SSLUOM are examples of this procedure.

      Two  examples  of  matrix  data structures  are the: 1) SLAP
      Triad  format and 2) SLAP Column format.

      ==================== S L A P Triad format ====================
      In  this   format only the  non-zeros are  stored.  They may
      appear  in *ANY* order.   The user  supplies three arrays of
      length NELT, where  NELT  is the number  of non-zeros in the
      matrix:  (IA(NELT), JA(NELT),  A(NELT)).  For each  non-zero
      the  user puts   the row  and  column index   of that matrix
      element in the IA and JA arrays.  The  value of the non-zero
      matrix  element is  placed in  the corresponding location of
      the A  array.  This is  an extremely easy data  structure to
      generate.  On  the other hand it  is  not too  efficient  on
      vector  computers   for the  iterative  solution  of  linear
      systems.  Hence, SLAP  changes this input  data structure to
      the SLAP   Column  format for the  iteration (but   does not
      change it back).

      Here is an example of the  SLAP Triad   storage format for a
      5x5 Matrix.  Recall that the entries may appear in any order.
```

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Cautions:

This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO SSDOMN, SSLUOM, ISSOMN

***REFERENCES 1. Mark K. Seager, A SLAP for the Masses, in G. F. Carey, Ed., Parallel Supercomputing: Methods, Algorithms and Applications, Wiley, 1989, pp.135-155.

***ROUTINES CALLED ISSOMN, R1MACH, SAXPY, SCOPY, SDOT

***REVISION HISTORY (YYMMDD)

```
871119  DATE WRITTEN
881213  Previous REVISION DATE
890915  Made changes requested at July 1989 CML Meeting.  (MKS)
890921  Removed TeX from comments.  (FNF)
890922  Numerous changes to prologue to make closer to SLATEC
        standard.  (FNF)
890929  Numerous changes to reduce SP/DP differences.  (FNF)
891004  Added new reference.
910411  Prologue converted to Version 4.0 format.  (BAB)
910502  Removed MATVEC and MSOLVE from ROUTINES CALLED list.  (FNF)
```

```
920407  COMMON BLOCK renamed SSLBLK.  (WRB)
920511  Added complete declaration section.  (WRB)
920929  Corrected format of reference.  (FNF)
921019  Changed 500.0 to 500 to reduce SP/DP differences.  (FNF)
921113  Corrected C***CATEGORY line.  (FNF)
930326  Removed unused variable.  (FNF)
END PROLOGUE
```

SOS

```
      SUBROUTINE SOS (FNC, NEQ, X, RTOLX, ATOLX, TOLF, IFLAG, RW, LRW,
     +   IW, LIW)
***BEGIN PROLOGUE  SOS
***PURPOSE  Solve a square system of nonlinear equations.
***LIBRARY   SLATEC
***CATEGORY  F2A
***TYPE      SINGLE PRECISION (SOS-S, DSOS-D)
***KEYWORDS  BROWN'S METHOD, NEWTON'S METHOD, NONLINEAR EQUATIONS,
             ROOTS, SOLUTIONS
***AUTHOR  Watts, H. A., (SNLA)
***DESCRIPTION

      SOS solves a system of NEQ simultaneous nonlinear equations in
      NEQ unknowns.  That is, it solves the problem   F(X)=0
      where X is a vector with components  X(1),...,X(NEQ)  and  F
      is a vector of nonlinear functions.  Each equation is of the form

                F (X(1),...,X(NEQ))=0     for K=1,...,NEQ.
                 K

      The algorithm is based on an iterative method which is a
      variation of Newton's method using Gaussian elimination
      in a manner similar to the Gauss-Seidel process.  Convergence
      is roughly quadratic.  All partial derivatives required by
      the algorithm are approximated by first difference quotients.
      The convergence behavior of this code is affected by the
      ordering of the equations, and it is advantageous to place linear
      and mildly nonlinear equations first in the ordering.

      Actually, SOS is merely an interfacing routine for
      calling subroutine SOSEQS which embodies the solution
      algorithm.  The purpose of this is to add greater
      flexibility and ease of use for the prospective user.

      SOSEQS calls the accompanying routine SOSSOL, which solves special
      triangular linear systems by back-substitution.

      The user must supply a function subprogram which evaluates the
      K-th equation only (K specified by SOSEQS) for each call
      to the subprogram.

      SOS represents an implementation of the mathematical algorithm
      described in the references below.  It is a modification of the
      code SOSNLE written by H. A. Watts in 1973.

 ***********************************************************************
   -Input-

      FNC -Name of the function program which evaluates the equations.
           This name must be in an EXTERNAL statement in the calling
           program.  The user must supply FNC in the form FNC(X,K),
           where X is the solution vector (which must be dimensioned
           in FNC) and FNC returns the value of the K-th function.

      NEQ -Number of equations to be solved.
```

```
    X   -Solution vector.  Initial guesses must be supplied.

    RTOLX -Relative error tolerance used in the convergence criteria.
         Each solution component X(I) is checked by an accuracy test
         of the form   ABS(X(I)-XOLD(I)) .LE. RTOLX*ABS(X(I))+ATOLX,
         where XOLD(I) represents the previous iteration value.
         RTOLX must be non-negative.

    ATOLX -Absolute error tolerance used in the convergence criteria.
         ATOLX must be non-negative.  If the user suspects some
         solution component may be zero, he should set ATOLX to an
         appropriate (depends on the scale of the remaining variables)
         positive value for better efficiency.

    TOLF -Residual error tolerance used in the convergence criteria.
         Convergence will be indicated if all residuals (values of the
         functions or equations) are not bigger than TOLF in
         magnitude.  Note that extreme care must be given in assigning
         an appropriate value for TOLF because this convergence test
         is dependent on the scaling of the equations.  An
         inappropriate value can cause premature termination of the
         iteration process.

    IFLAG -Optional input indicator.  You must set  IFLAG=-1  if you
         want to use any of the optional input items listed below.
         Otherwise set it to zero.

    RW  -A REAL work array which is split apart by SOS and used
         internally by SOSEQS.

    LRW -Dimension of the RW array.  LRW must be at least
                   1 + 6*NEQ + NEQ*(NEQ+1)/2

    IW  -An INTEGER work array which is split apart by SOS and used
         internally by SOSEQS.

    LIW -Dimension of the IW array. LIW must be at least  3 + NEQ.

  -Optional Input-

    IW(1) -Internal printing parameter.  You must set  IW(1)=-1  if
         you want the intermediate solution iterates to be printed.

    IW(2) -Iteration limit.  The maximum number of allowable
         iterations can be specified, if desired.  To override the
         default value of 50, set IW(2) to the number wanted.

    Remember, if you tell the code that you are using one of the
              options (by setting IFLAG=-1), you must supply values
              for both IW(1) and IW(2).

**********************************************************************
  -Output-

    X   -Solution vector.

    IFLAG -Status indicator

                         *** Convergence to a Solution ***
```

```
          1 Means satisfactory convergence to a solution was achieved.
            Each solution component X(I) satisfies the error tolerance
            test   ABS(X(I)-XOLD(I)) .LE. RTOLX*ABS(X(I))+ATOLX.

          2 Means procedure converged to a solution such that all
            residuals are at most TOLF in magnitude,
            ABS(FNC(X,I)) .LE. TOLF.

          3 Means that conditions for both IFLAG=1 and IFLAG=2 hold.

          4 Means possible numerical convergence.  Behavior indicates
            limiting precision calculations as a result of user asking
            for too much accuracy or else convergence is very slow.
            Residual norms and solution increment norms have
            remained roughly constant over several consecutive
            iterations.

                        *** Task Interrupted ***

          5 Means the allowable number of iterations has been met
            without obtaining a solution to the specified accuracy.
            Very slow convergence may be indicated.  Examine the
            approximate solution returned and see if the error
            tolerances seem appropriate.

          6 Means the allowable number of iterations has been met and
            the iterative process does not appear to be converging.
            A local minimum may have been encountered or there may be
            limiting precision difficulties.

          7 Means that the iterative scheme appears to be diverging.
            Residual norms and solution increment norms have
            increased over several consecutive iterations.

                        *** Task Cannot Be Continued ***

          8 Means that a Jacobian-related matrix was singular.

          9 Means improper input parameters.

          *** IFLAG should be examined after each call to   ***
          *** SOS with the appropriate action being taken.  ***


     RW(1) -Contains a norm of the residual.

     IW(3) -Contains the number of iterations used by the process.

 ***********************************************************************
***REFERENCES  K. M. Brown, Solution of simultaneous nonlinear
                equations, Algorithm 316, Communications of the
                A.C.M. 10, (1967), pp. 728-729.
              K. M. Brown, A quadratically convergent Newton-like
                method based upon Gaussian elimination, SIAM Journal
                on Numerical Analysis 6, (1969), pp. 560-569.
***ROUTINES CALLED  SOSEQS, XERMSG
***REVISION HISTORY  (YYMMDD)
   801001  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
```

```
891214  Prologue converted to Version 4.0 format.  (BAB)
900510  Convert XERRWV calls to XERMSG calls, changed Prologue
        comments to agree with DSOS.  (RWC)
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

SPBCO

```
      SUBROUTINE SPBCO (ABD, LDA, N, M, RCOND, Z, INFO)
***BEGIN PROLOGUE  SPBCO
***PURPOSE  Factor a real symmetric positive definite matrix stored in
            band form and estimate the condition number of the matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B2
***TYPE      SINGLE PRECISION (SPBCO-S, DPBCO-D, CPBCO-C)
***KEYWORDS  BANDED, CONDITION NUMBER, LINEAR ALGEBRA, LINPACK,
             MATRIX FACTORIZATION, POSITIVE DEFINITE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SPBCO factors a real symmetric positive definite matrix
      stored in band form and estimates the condition of the matrix.

      If  RCOND  is not needed, SPBFA is slightly faster.
      To solve  A*X = B , follow SPBCO by SPBSL.
      To compute  INVERSE(A)*C , follow SPBCO by SPBSL.
      To compute  DETERMINANT(A) , follow SPBCO by SPBDI.

      On Entry

         ABD     REAL(LDA, N)
                 the matrix to be factored.  The columns of the upper
                 triangle are stored in the columns of ABD and the
                 diagonals of the upper triangle are stored in the
                 rows of ABD .  See the comments below for details.

         LDA     INTEGER
                 the leading dimension of the array  ABD .
                 LDA must be .GE. M + 1 .

         N       INTEGER
                 the order of the matrix  A .

         M       INTEGER
                 the number of diagonals above the main diagonal.
                 0 .LE. M .LT. N .

      On Return

         ABD     an upper triangular matrix  R , stored in band
                 form, so that  A = TRANS(R)*R .
                 If  INFO .NE. 0 , the factorization is not complete.

         RCOND   REAL
                 an estimate of the reciprocal condition of  A .
                 For the system  A*X = B , relative perturbations
                 in  A  and  B  of size  EPSILON  may cause
                 relative perturbations in  X  of size  EPSILON/RCOND .
                 If  RCOND  is so small that the logical expression
                           1.0 + RCOND .EQ. 1.0
                 is true, then  A  may be singular to working
                 precision.  In particular,  RCOND  is zero  if
                 exact singularity is detected or the estimate
                 underflows.  If INFO .NE. 0 , RCOND is unchanged.
```

```
         Z       REAL(N)
                 a work vector whose contents are usually unimportant.
                 If  A  is singular to working precision, then  Z  is
                 an approximate null vector in the sense that
                 NORM(A*Z) = RCOND*NORM(A)*NORM(Z) .
                 If  INFO .NE. 0 , Z  is unchanged.

         INFO    INTEGER
                 = 0  for normal return.
                 = K  signals an error condition.  The leading minor
                      of order  K  is not positive definite.

      Band Storage

            If  A  is a symmetric positive definite band matrix,
            the following program segment will set up the input.

                    M = (band width above diagonal)
                    DO 20 J = 1, N
                       I1 = MAX(1, J-M)
                       DO 10 I = I1, J
                          K = I-J+M+1
                          ABD(K,J) = A(I,J)
                 10    CONTINUE
                 20 CONTINUE

            This uses  M + 1  rows of  A , except for the  M by M
            upper left triangle, which is ignored.

      Example:  If the original matrix is

            11 12 13  0  0  0
            12 22 23 24  0  0
            13 23 33 34 35  0
             0 24 34 44 45 46
             0  0 35 45 55 56
             0  0  0 46 56 66

      then  N = 6 , M = 2  and  ABD  should contain

             *  * 13 24 35 46
             * 12 23 34 45 56
            11 22 33 44 55 66

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SASUM, SAXPY, SDOT, SPBFA, SSCAL
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SPBDI

```
      SUBROUTINE SPBDI (ABD, LDA, N, M, DET)
***BEGIN PROLOGUE  SPBDI
***PURPOSE  Compute the determinant of a symmetric positive definite
            band matrix using the factors computed by SPBCO or SPBFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D3B2
***TYPE      SINGLE PRECISION (SPBDI-S, DPBDI-D, CPBDI-C)
***KEYWORDS  BANDED, DETERMINANT, INVERSE, LINEAR ALGEBRA, LINPACK,
             MATRIX, POSITIVE DEFINITE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SPBDI computes the determinant
      of a real symmetric positive definite band matrix
      using the factors computed by SPBCO or SPBFA.
      If the inverse is needed, use SPBSL  N  times.

      On Entry

         ABD     REAL(LDA, N)
                 the output from SPBCO or SPBFA.

         LDA     INTEGER
                 the leading dimension of the array  ABD .

         N       INTEGER
                 the order of the matrix  A .

         M       INTEGER
                 the number of diagonals above the main diagonal.

      On Return

         DET     REAL(2)
                 determinant of original matrix in the form
                 Determinant = DET(1) * 10.0**DET(2)
                 with  1.0 .LE. DET(1) .LT. 10.0
                 or  DET(1) .EQ. 0.0 .

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SPBFA

```
      SUBROUTINE SPBFA (ABD, LDA, N, M, INFO)
***BEGIN PROLOGUE  SPBFA
***PURPOSE  Factor a real symmetric positive definite matrix stored in
            band form.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B2
***TYPE      SINGLE PRECISION (SPBFA-S, DPBFA-D, CPBFA-C)
***KEYWORDS  BANDED, LINEAR ALGEBRA, LINPACK, MATRIX FACTORIZATION,
             POSITIVE DEFINITE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SPBFA factors a real symmetric positive definite matrix
      stored in band form.

      SPBFA is usually called by SPBCO, but it can be called
      directly with a saving in time if  RCOND  is not needed.

      On Entry

         ABD     REAL(LDA, N)
                 the matrix to be factored.  The columns of the upper
                 triangle are stored in the columns of ABD and the
                 diagonals of the upper triangle are stored in the
                 rows of ABD .  See the comments below for details.

         LDA     INTEGER
                 the leading dimension of the array  ABD .
                 LDA must be .GE. M + 1 .

         N       INTEGER
                 the order of the matrix  A .

         M       INTEGER
                 the number of diagonals above the main diagonal.
                 0 .LE. M .LT. N .

      On Return

         ABD     an upper triangular matrix  R , stored in band
                 form, so that  A = TRANS(R)*R .

         INFO    INTEGER
                 = 0  for normal return.
                 = K  if the leading minor of order  K  is not
                      positive definite.

      Band Storage

            If  A  is a symmetric positive definite band matrix,
            the following program segment will set up the input.

                    M = (band width above diagonal)
                    DO 20 J = 1, N
                       I1 = MAX(1, J-M)
                       DO 10 I = I1, J
```

```
                         K = I-J+M+1
                         ABD(K,J) = A(I,J)
                10    CONTINUE
                20 CONTINUE

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SDOT
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SPBSL

```
      SUBROUTINE SPBSL (ABD, LDA, N, M, B)
***BEGIN PROLOGUE  SPBSL
***PURPOSE  Solve a real symmetric positive definite band system
            using the factors computed by SPBCO or SPBFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B2
***TYPE      SINGLE PRECISION (SPBSL-S, DPBSL-D, CPBSL-C)
***KEYWORDS  BANDED, LINEAR ALGEBRA, LINPACK, MATRIX,
             POSITIVE DEFINITE, SOLVE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SPBSL solves the real symmetric positive definite band
      system  A*X = B
      using the factors computed by SPBCO or SPBFA.

      On Entry

         ABD     REAL(LDA, N)
                 the output from SPBCO or SPBFA.

         LDA     INTEGER
                 the leading dimension of the array  ABD .

         N       INTEGER
                 the order of the matrix  A .

         M       INTEGER
                 the number of diagonals above the main diagonal.

         B       REAL(N)
                 the right hand side vector.

      On Return

         B       the solution vector  X .

      Error Condition

         A division by zero will occur if the input factor contains
         a zero on the diagonal.  Technically, this indicates
         singularity, but it is usually caused by improper subroutine
         arguments.  It will not occur if the subroutines are called
         correctly and  INFO .EQ. 0 .

      To compute  INVERSE(A) * C  where  C  is a matrix
      with  P  columns
            CALL SPBCO(ABD,LDA,N,RCOND,Z,INFO)
            IF (RCOND is too small .OR. INFO .NE. 0) GO TO ...
            DO 10 J = 1, P
               CALL SPBSL(ABD,LDA,N,C(1,J))
         10 CONTINUE

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SDOT
```

```
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SPENC

```
      FUNCTION SPENC (X)
***BEGIN PROLOGUE  SPENC
***PURPOSE  Compute a form of Spence's integral due to K. Mitchell.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C5
***TYPE      SINGLE PRECISION (SPENC-S, DSPENC-D)
***KEYWORDS  FNLIB, SPECIAL FUNCTIONS, SPENCE'S INTEGRAL
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 Evaluate a form of Spence's function defined by
        integral from 0 to X of  -LOG(1-Y)/Y  DY.
 For ABS(X) .LE. 1, the uniformly convergent expansion
        SPENC = sum K=1,infinity  X**K / K**2     is valid.

 Spence's function can be used to evaluate much more general integral
 forms.  For example,
        integral from 0 to Z of  LOG(A*X+B)/(C*X+D)  DX  =
             LOG(ABS(B-A*D/C))*LOG(ABS(A*(C*X+D)/(A*D-B*C)))/C
             - SPENC (A*(C*Z+D)/(A*D-B*C)) / C.

 Ref -- K. Mitchell, Philosophical Magazine, 40, p. 351 (1949).
        Stegun and Abromowitz, AMS 55, p. 1004.


 Series for SPEN       on the interval  0.          to  5.00000D-01
                                       with weighted error   6.82E-17
                                        log weighted error  16.17
                                 significant figures required  15.22
                                      decimal places required  16.81

***REFERENCES  (NONE)
***ROUTINES CALLED  CSEVL, INITS, R1MACH
***REVISION HISTORY  (YYMMDD)
   780201  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890531  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   END PROLOGUE
```

SPLP

```
      SUBROUTINE SPLP (DUSRMT, MRELAS, NVARS, COSTS, PRGOPT, DATTRV,
     +   BL, BU, IND, INFO, PRIMAL, DUALS, IBASIS, WORK, LW, IWORK, LIW)
***BEGIN PROLOGUE  SPLP
***PURPOSE  Solve linear programming problems involving at
            most a few thousand constraints and variables.
            Takes advantage of sparsity in the constraint matrix.
***LIBRARY   SLATEC
***CATEGORY  G2A2
***TYPE      SINGLE PRECISION (SPLP-S, DSPLP-D)
***KEYWORDS  LINEAR CONSTRAINTS, LINEAR OPTIMIZATION,
             LINEAR PROGRAMMING, LP, SPARSE CONSTRAINTS
***AUTHOR  Hanson, R. J., (SNLA)
           Hiebert, K. L., (SNLA)
***DESCRIPTION

      These are the short usage instructions; for details about
      other features, options and methods for defining the matrix
      A, see the extended usage instructions which are contained in
      the Long Description section below.

    |------------|
    |Introduction|
    |------------|
      The subprogram SPLP( ) solves a linear optimization problem.
      The problem statement is as follows

                           minimize (transpose of costs)*x
                           subject to A*x=w.

      The entries of the unknowns x and w may have simple lower or
      upper bounds (or both), or be free to take on any value.  By
      setting the bounds for x and w, the user is imposing the con-
      straints of the problem.  The matrix A has MRELAS rows and
      NVARS columns.  The vectors costs, x, and w respectively
      have NVARS, NVARS, and MRELAS number of entries.

      The input for the problem includes the problem dimensions,
      MRELAS and NVARS, the array COSTS(*), data for the matrix
      A, and the bound information for the unknowns x and w, BL(*),
      BU(*), and IND(*).  Only the nonzero entries of the matrix A
      are passed to SPLP( ).

      The output from the problem (when output flag INFO=1) includes
      optimal values for x and w in PRIMAL(*), optimal values for
      dual variables of the equations A*x=w and the simple bounds
      on x in  DUALS(*), and the indices of the basic columns,
      IBASIS(*).

   |-------------------------------|
   |Fortran Declarations Required:|
   |-------------------------------|

      DIMENSION COSTS(NVARS),PRGOPT(*),DATTRV(*),
     *BL(NVARS+MRELAS),BU(NVARS+MRELAS),IND(NVARS+MRELAS),
     *PRIMAL(NVARS+MRELAS),DUALS(MRELAS+NVARS),IBASIS(NVARS+MRELAS),
     *WORK(LW),IWORK(LIW)
```

EXTERNAL USRMAT

The dimensions of PRGOPT(*) and DATTRV(*) must be at least 1. The exact lengths will be determined by user-required options and data transferred to the subprogram USRMAT( ).

The values of LW and LIW, the lengths of the arrays WORK(*) and IWORK(*), must satisfy the inequalities

```
          LW .GE. 4*NVARS+ 8*MRELAS+LAMAT+  LBM
          LIW.GE.   NVARS+11*MRELAS+LAMAT+2*LBM
```

It is an error if they do not both satisfy these inequalities. (The subprogram will inform the user of the required lengths if either LW or LIW is wrong.) The values of LAMAT and LBM nominally are

```
          LAMAT=4*NVARS+7
and       LBM  =8*MRELAS
```

LAMAT determines the length of the sparse matrix storage area. The value of LBM determines the amount of storage available to decompose and update the active basis matrix.

```
|------|
|Input:|
|------|
```

MRELAS,NVARS

These parameters are respectively the number of constraints (the linear relations A*x=w that the unknowns x and w are to satisfy) and the number of entries in the vector x. Both must be .GE. 1. Other values are errors.

COSTS(*)

The NVARS entries of this array are the coefficients of the linear objective function. The value COSTS(J) is the multiplier for variable J of the unknown vector x. Each entry of this array must be defined.

USRMAT

This is the name of a specific subprogram in the SPLP( ) package used to define the matrix A. In this usage mode of SPLP( ) the user places the nonzero entries of A in the array DATTRV(*) as given in the description of that parameter. The name USRMAT must appear in a Fortran EXTERNAL statement.

DATTRV(*)

The array DATTRV(*) contains data for the matrix A as follows: Each column (numbered J) requires (floating point) data consisting of the value (-J) followed by pairs of values. Each pair consists of the row index immediately followed by the value of the matrix at that entry. A value of J=0 signals that there are no more columns. The required length of DATTRV(*) is 2*no. of nonzeros + NVARS + 1.

BL(*),BU(*),IND(*)

The values of IND(*) are input parameters that define the form of the bounds for the unknowns x and w. The values for the bounds are found in the arrays BL(*) and BU(*) as follows.

```
For values of J between 1 and NVARS,
     if IND(J)=1, then X(J) .GE. BL(J); BU(J) is not used.
     if IND(J)=2, then X(J) .LE. BU(J); BL(J) is not used.
     if IND(J)=3, then BL(J) .LE. X(J) .LE. BU(J),(BL(J)=BU(J) ok)
     if IND(J)=4, then X(J) is free to have any value,
     and BL(J), BU(J) are not used.

For values of I between NVARS+1 and NVARS+MRELAS,
     if IND(I)=1, then W(I-NVARS) .GE. BL(I); BU(I) is not used.
     if IND(I)=2, then W(I-NVARS) .LE. BU(I); BL(I) is not used.
     if IND(I)=3, then BL(I) .LE. W(I-NVARS) .LE. BU(I),
     (BL(I)=BU(I) is ok).
     if IND(I)=4, then W(I-NVARS) is free to have any value,
     and BL(I), BU(I) are not used.
```

A value of IND(*) not equal to 1,2,3 or 4 is an error. When IND(I)=3, BL(I) must be .LE. BU(I). The condition BL(I).GT. BU(I) indicates infeasibility and is an error.

PRGOPT(*)

This array is used to redefine various parameters within SPLP( ). Frequently, perhaps most of the time, a user will be satisfied and obtain the solutions with no changes to any of these parameters. To try this, simply set PRGOPT(1)=1.E0.

For users with more sophisticated needs, SPLP( ) provides several options that may be used to take advantage of more detailed knowledge of the problem or satisfy other utilitarian needs. The complete description of how to use this option array to utilize additional subprogram features is found under the heading of SPLP( ) Subprogram Options in the Extended Usage Instructions.

Briefly, the user should note the following value of the parameter KEY and the corresponding task or feature desired before turning to that document.

Value of KEY	Brief Statement of Purpose for Option
50	Change from a minimization problem to a maximization problem.
51	Change the amount of printed output. Normally, no printed output is obtained.
52	Redefine the line length and precision used for the printed output.
53	Redefine the values of LAMAT and LBM that were discussed above under the heading Fortran Declarations Required.
54	Redefine the unit number where pages of the sparse data matrix A are stored. Normally, the unit number is 1.

```
   55        A computation, partially completed, is
             being continued.  Read the up-to-date
             partial results from unit number 2.
   56        Redefine the unit number where the partial results
             are stored.  Normally, the unit number is 2.
   57        Save partial results on unit 2 either after
             maximum iterations or at the optimum.
   58        Redefine the value for the maximum number of
             iterations.  Normally, the maximum number of
             iterations is 3*(NVARS+MRELAS).
   59        Provide SPLP( ) with a starting (feasible)
             nonsingular basis.  Normally, SPLP( ) starts
             with the identity matrix columns corresponding
             to the vector w.
   60        The user has provided scale factors for the
             columns of A.  Normally, SPLP( ) computes scale
             factors that are the reciprocals of the max. norm
             of each column.
   61        The user has provided a scale factor
             for the vector costs.  Normally, SPLP( ) computes
             a scale factor equal to the reciprocal of the
             max. norm of the vector costs after the column
             scaling for the data matrix has been applied.
   62        Size parameters, namely the smallest and
             largest magnitudes of nonzero entries in
             the matrix A, are provided.  Values noted
             outside this range are to be considered errors.
   63        Redefine the tolerance required in
             evaluating residuals for feasibility.
             Normally, this value is set to RELPR,
             where RELPR = relative precision of the arithmetic.
   64        Change the criterion for bringing new variables
             into the basis from the steepest edge (best
             local move) to the minimum reduced cost.
   65        Redefine the value for the number of iterations
             between recalculating the error in the primal
             solution.  Normally, this value is equal to ten.
   66        Perform "partial pricing" on variable selection.
             Redefine the value for the number of negative
             reduced costs to compute (at most) when finding
             a variable to enter the basis.  Normally this
             value is set to NVARS.  This implies that no
             "partial pricing" is used.
   67        Adjust the tuning factor (normally one) to apply
             to the primal and dual error estimates.
   68        Pass  information to the  subprogram  FULMAT(),
             provided with the SPLP() package, so that a Fortran
             two-dimensional array can be used as the argument
             DATTRV(*).
   69        Pass an absolute tolerance to use for the feasibility
             test when the usual relative error test indicates
             infeasibility.  The nominal value of this tolerance,
             TOLABS, is zero.


|---------------|
|Working Arrays:|
|---------------|

   WORK(*),LW,
```

```
    IWORK(*),LIW
    ------------
    The arrays WORK(*) and IWORK(*) are respectively floating point
    and type INTEGER working arrays for SPLP( ) and its
    subprograms.  The lengths of these arrays are respectively
    LW and LIW.  These parameters must satisfy the inequalities
    noted above under the heading "Fortran Declarations Required:"
    It is an error if either value is too small.

 |-----------------------------|
 |Input/Output files required:|
 |-----------------------------|

    Fortran unit 1 is used by SPLP( ) to store the sparse matrix A
    out of high-speed memory.  A crude
    upper bound for the amount of information written on unit 1
    is 6*nz, where nz is the number of nonzero entries in A.

 |-------|
 |Output:|
 |-------|

    INFO,PRIMAL(*),DUALS(*)
    -----------------------
    The integer flag INFO indicates why SPLP( ) has returned to the
    user.  If INFO=1 the solution has been computed.  In this case
    X(J)=PRIMAL(J) and W(I)=PRIMAL(I+NVARS).  The dual variables
    for the equations A*x=w are in the array DUALS(I)=dual for
    equation number I.  The dual value for the component X(J) that
    has an upper or lower bound (or both) is returned in
    DUALS(J+MRELAS).  The only other values for INFO are .LT. 0.
    The meaning of these values can be found by reading
    the diagnostic message in the output file, or by looking for
    error number = (-INFO) in the Extended Usage Instructions
    under the heading:

         List of SPLP( ) Error and Diagnostic Messages.

    BL(*),BU(*),IND(*)
    ------------------
    These arrays are output parameters only under the (unusual)
    circumstances where the stated problem is infeasible, has an
    unbounded optimum value, or both.  These respective conditions
    correspond to INFO=-1,-2 or -3.    See the Extended
    Usage Instructions for further details.

    IBASIS(I),I=1,...,MRELAS
    ------------------------
    This array contains the indices of the variables that are
    in the active basis set at the solution (INFO=1).  A value
    of IBASIS(I) between 1 and NVARS corresponds to the variable
    X(IBASIS(I)).  A value of IBASIS(I) between NVARS+1 and NVARS+
    MRELAS corresponds to the variable W(IBASIS(I)-NVARS).

*Long Description:

    SUBROUTINE SPLP(USRMAT,MRELAS,NVARS,COSTS,PRGOPT,DATTRV,
   *           BL,BU,IND,INFO,PRIMAL,DUALS,IBASIS,WORK,LW,IWORK,LIW)

    |------------|
```

```
|Introduction|
|------------|
The subprogram SPLP( ) solves a linear optimization problem.
The problem statement is as follows

                    minimize (transpose of costs)*x
                    subject to A*x=w.

The entries of the unknowns x and w may have simple lower or
upper bounds (or both), or be free to take on any value.  By
setting the bounds for x and w, the user is imposing the con-
straints of the problem.

(The problem may also be stated as a maximization
problem.  This is done by means of input in the option array
PRGOPT(*).)  The matrix A has MRELAS rows and NVARS columns.  The
vectors costs, x, and w respectively have NVARS, NVARS, and
MRELAS number of entries.

The input for the problem includes the problem dimensions,
MRELAS and NVARS, the array COSTS(*), data for the matrix
A, and the bound information for the unknowns x and w, BL(*),
BU(*), and IND(*).

The output from the problem (when output flag INFO=1) includes
optimal values for x and w in PRIMAL(*), optimal values for
dual variables of the equations A*x=w and the simple bounds
on x in  DUALS(*), and the indices of the basic columns in
IBASIS(*).

|------------------------------|
|Fortran Declarations Required:|
|------------------------------|

 DIMENSION COSTS(NVARS),PRGOPT(*),DATTRV(*),
*BL(NVARS+MRELAS),BU(NVARS+MRELAS),IND(NVARS+MRELAS),
*PRIMAL(NVARS+MRELAS),DUALS(MRELAS+NVARS),IBASIS(NVARS+MRELAS),
*WORK(LW),IWORK(LIW)

 EXTERNAL USRMAT (or 'NAME', if user provides the subprogram)

 The dimensions of PRGOPT(*) and DATTRV(*) must be at least 1.
 The exact lengths will be determined by user-required options and
 data transferred to the subprogram USRMAT( ) ( or 'NAME').

 The values of LW and LIW, the lengths of the arrays WORK(*)
 and IWORK(*), must satisfy the inequalities

           LW .GE. 4*NVARS+ 8*MRELAS+LAMAT+  LBM
           LIW.GE.   NVARS+11*MRELAS+LAMAT+2*LBM

 It is an error if they do not both satisfy these inequalities.
 (The subprogram will inform the user of the required lengths
 if either LW or LIW is wrong.)  The values of LAMAT and LBM
 nominally are

           LAMAT=4*NVARS+7
 and       LBM  =8*MRELAS

 These values will be as shown unless the user changes them by
```

means of input in the option array PRGOPT(*). The value of LAMAT determines the length of the sparse matrix "staging" area. For reasons of efficiency the user may want to increase the value of LAMAT. The value of LBM determines the amount of storage available to decompose and update the active basis matrix. Due to exhausting the working space because of fill-in, it may be necessary for the user to increase the value of LBM. (If this situation occurs an informative diagnostic is printed and a value of INFO=-28 is obtained as an output parameter.)

```
|------|
|Input:|
|------|
```

MRELAS,NVARS

These parameters are respectively the number of constraints (the linear relations A*x=w that the unknowns x and w are to satisfy) and the number of entries in the vector x. Both must be .GE. 1. Other values are errors.

COSTS(*)

The NVARS entries of this array are the coefficients of the linear objective function. The value COSTS(J) is the multiplier for variable J of the unknown vector x. Each entry of this array must be defined. This array can be changed by the user between restarts. See options with KEY=55,57 for details of checkpointing and restarting.

USRMAT

This is the name of a specific subprogram in the SPLP( ) package that is used to define the matrix entries when this data is passed to SPLP( ) as a linear array. In this usage mode of SPLP( ) the user gives information about the nonzero entries of A in DATTRV(*) as given under the description of that parameter. The name USRMAT must appear in a Fortran EXTERNAL statement. Users who are passing the matrix data with USRMAT( ) can skip directly to the description of the input parameter DATTRV(*). Also see option 68 for passing the constraint matrix data using a standard Fortran two-dimensional array.

If the user chooses to provide a subprogram 'NAME'( ) to define the matrix A, then DATTRV(*) may be used to pass floating point data from the user's program unit to the subprogram 'NAME'( ). The content of DATTRV(*) is not changed in any way.

The subprogram 'NAME'( ) can be of the user's choice but it must meet Fortran standards and it must appear in a Fortran EXTERNAL statement. The first statement of the subprogram has the form

```
     SUBROUTINE 'NAME'(I,J,AIJ, INDCAT, PRGOPT, DATTRV, IFLAG)
```

The variables I,J, INDCAT, IFLAG(10) are type INTEGER,
while AIJ, PRGOPT(*),DATTRV(*) are type REAL.

The user interacts with the contents of IFLAG(*) to direct the appropriate action. The algorithmic steps are

as follows.

```
     Test IFLAG(1).

        IF(IFLAG(1).EQ.1) THEN

          Initialize the necessary pointers and data
          for defining the matrix A.  The contents
          of IFLAG(K), K=2,...,10, may be used for
          storage of the pointers.  This array remains intact
          between calls to 'NAME'( ) by SPLP( ).
          RETURN

        END IF

        IF(IFLAG(1).EQ.2) THEN

          Define one set of values for I,J,AIJ, and INDCAT.
          Each nonzero entry of A must be defined this way.
          These values can be defined in any convenient order.
          (It is most efficient to define the data by
          columns in the order 1,...,NVARS; within each
          column define the entries in the order 1,...,MRELAS.)
          If this is the last matrix value to be
          defined or updated, then set IFLAG(1)=3.
          (When I and J are positive and respectively no larger
          than MRELAS and NVARS, the value of AIJ is used to
          define (or update) row I and column J of A.)
          RETURN

        END IF

          END
```

Remarks: The values of I and J are the row and column indices for the nonzero entries of the matrix A. The value of this entry is AIJ.
Set INDCAT=0 if this value defines that entry.
Set INDCAT=1 if this entry is to be updated,
new entry=old entry+AIJ.
A value of I not between 1 and MRELAS, a value of J not between 1 and NVARS, or a value of INDCAT not equal to 0 or 1 are each errors.

The contents of IFLAG(K), K=2,...,10, can be used to remember the status (of the process of defining the matrix entries) between calls to 'NAME'( ) by SPLP( ). On entry to 'NAME'( ), only the values 1 or 2 will be in IFLAG(1). More than 2*NVARS*MRELAS definitions of the matrix elements is considered an error because it suggests an infinite loop in the user-written subprogram 'NAME'( ). Any matrix element not provided by 'NAME'( ) is defined to be zero.

The REAL arrays PRGOPT(*) and DATTRV(*) are passed as arguments directly from SPLP( ) to 'NAME'( ). The array PRGOPT(*) contains any user-defined program options. In this usage mode the array DATTRV(*) may now contain any (type REAL) data that the user needs to define the matrix A. Both arrays PRGOPT(*) and

```
          DATTRV(*) remain intact between calls to 'NAME'( )
          by SPLP( ).
Here is a subprogram that communicates the matrix values for A,
as represented in DATTRV(*), to SPLP( ).  This subprogram,
called USRMAT( ), is included as part of the SPLP( ) package.
This subprogram 'decodes' the array DATTRV(*) and defines the
nonzero entries of the matrix  A for SPLP( ) to store.  This
listing is presented here as a guide and example
for the users who find it necessary to write their own subroutine
for this purpose.  The contents of DATTRV(*) are given below in
the description of that parameter.

SUBROUTINE USRMAT(I,J,AIJ, INDCAT,PRGOPT,DATTRV,IFLAG)
DIMENSION PRGOPT(*),DATTRV(*),IFLAG(10)

IF(IFLAG(1).EQ.1) THEN

THIS IS THE INITIALIZATION STEP.  THE VALUES OF IFLAG(K),K=2,3,4,
ARE RESPECTIVELY THE COLUMN INDEX, THE ROW INDEX (OR THE NEXT COL.
INDEX), AND THE POINTER TO THE MATRIX ENTRY'S VALUE WITHIN
DATTRV(*).  ALSO CHECK (DATTRV(1)=0.) SIGNIFYING NO DATA.
     IF(DATTRV(1).EQ.0.) THEN
     I = 0
     J = 0
     IFLAG(1) = 3
     ELSE
     IFLAG(2)=-DATTRV(1)
     IFLAG(3)= DATTRV(2)
     IFLAG(4)= 3
     END IF

     RETURN
ELSE
     J=IFLAG(2)
     I=IFLAG(3)
     L=IFLAG(4)
     IF(I.EQ.0) THEN

SIGNAL THAT ALL OF THE NONZERO ENTRIES HAVE BEEN DEFINED.
          IFLAG(1)=3
          RETURN
     ELSE IF(I.LT.0) THEN

SIGNAL THAT A SWITCH IS MADE TO A NEW COLUMN.
          J=-I
          I=DATTRV(L)
          L=L+1
     END IF

     AIJ=DATTRV(L)

UPDATE THE INDICES AND POINTERS FOR THE NEXT ENTRY.
     IFLAG(2)=J
     IFLAG(3)=DATTRV(L+1)
     IFLAG(4)=L+2

INDCAT=0 DENOTES THAT ENTRIES OF THE MATRIX ARE ASSIGNED THE
VALUES FROM DATTRV(*).  NO ACCUMULATION IS PERFORMED.
     INDCAT=0
     RETURN
```

```
END IF
END
```

DATTRV(*)

If the user chooses to use the provided subprogram USRMAT( ) then the array DATTRV(*) contains data for the matrix A as follows: Each column (numbered J) requires (floating point) data consisting of the value (-J) followed by pairs of values. Each pair consists of the row index immediately followed by the value of the matrix at that entry. A value of J=0 signals that there are no more columns. (See "Example of SPLP( ) Usage," below.) The dimension of DATTRV(*) must be 2*no. of nonzeros + NVARS + 1 in this usage. No checking of the array length is done by the subprogram package.

If the Save/Restore feature is in use (see options with KEY=55,57 for details of checkpointing and restarting) USRMAT( ) can be used to redefine entries of the matrix. The matrix entries are redefined or overwritten. No accumulation is performed.
Any other nonzero entry of A, defined in a previous call to SPLP( ), remain intact.

BL(*),BU(*),IND(*)

The values of IND(*) are input parameters that define the form of the bounds for the unknowns x and w. The values for the bounds are found in the arrays BL(*) and BU(*) as follows.

```
For values of J between 1 and NVARS,
     if IND(J)=1, then X(J) .GE. BL(J); BU(J) is not used.
     if IND(J)=2, then X(J) .LE. BU(J); BL(J) is not used.
     if IND(J)=3, then BL(J) .LE. X(J) .LE. BU(J),(BL(J)=BU(J) ok)
     if IND(J)=4, then X(J) is free to have any value,
     and BL(J), BU(J) are not used.

For values of I between NVARS+1 and NVARS+MRELAS,
     if IND(I)=1, then W(I-NVARS) .GE. BL(I); BU(I) is not used.
     if IND(I)=2, then W(I-NVARS) .LE. BU(I); BL(I) is not used.
     if IND(I)=3, then BL(I) .LE. W(I-NVARS) .LE. BU(I),
     (BL(I)=BU(I) is ok).
     if IND(I)=4, then W(I-NVARS) is free to have any value,
     and BL(I), BU(I) are not used.
```

A value of IND(*) not equal to 1,2,3 or 4 is an error. When IND(I)=3, BL(I) must be .LE. BU(I). The condition BL(I).GT. BU(I) indicates infeasibility and is an error. These arrays can be changed by the user between restarts. See options with KEY=55,57 for details of checkpointing and restarting.

PRGOPT(*)

This array is used to redefine various parameters within SPLP( ). Frequently, perhaps most of the time, a user will be satisfied and obtain the solutions with no changes to any of these parameters. To try this, simply set PRGOPT(1)=1.E0.

For users with more sophisticated needs, SPLP( ) provides several

options that may be used to take advantage of more detailed knowledge of the problem or satisfy other utilitarian needs. The complete description of how to use this option array to utilize additional subprogram features is found under the heading "Usage of SPLP( ) Subprogram Options."

Briefly, the user should note the following value of the parameter KEY and the corresponding task or feature desired before turning to that section.

Value of KEY	Brief Statement of Purpose for Option
50	Change from a minimization problem to a maximization problem.
51	Change the amount of printed output. Normally, no printed output is obtained.
52	Redefine the line length and precision used for the printed output.
53	Redefine the values of LAMAT and LBM that were discussed above under the heading Fortran Declarations Required.
54	Redefine the unit number where pages of the sparse data matrix A are stored. Normally, the unit number is 1.
55	A computation, partially completed, is being continued. Read the up-to-date partial results from unit number 2.
56	Redefine the unit number where the partial results are stored. Normally, the unit number is 2.
57	Save partial results on unit 2 either after maximum iterations or at the optimum.
58	Redefine the value for the maximum number of iterations. Normally, the maximum number of iterations is 3*(NVARS+MRELAS).
59	Provide SPLP( ) with a starting (feasible) nonsingular basis. Normally, SPLP( ) starts with the identity matrix columns corresponding to the vector w.
60	The user has provided scale factors for the columns of A. Normally, SPLP( ) computes scale factors that are the reciprocals of the max. norm of each column.
61	The user has provided a scale factor for the vector costs. Normally, SPLP( ) computes a scale factor equal to the reciprocal of the max. norm of the vector costs after the column scaling for the data matrix has been applied.
62	Size parameters, namely the smallest and largest magnitudes of nonzero entries in the matrix A, are provided. Values noted outside this range are to be considered errors.
63	Redefine the tolerance required in evaluating residuals for feasibility. Normally, this value is set to the value RELPR, where RELPR = relative precision of the arithmetic.
64	Change the criterion for bringing new variables into the basis from the steepest edge (best local move) to the minimum reduced cost.
65	Redefine the value for the number of iterations

between recalculating the error in the primal solution. Normally, this value is equal to ten.

66 Perform "partial pricing" on variable selection. Redefine the value for the number of negative reduced costs to compute (at most) when finding a variable to enter the basis. Normally this value is set to NVARS. This implies that no "partial pricing" is used.

67 Adjust the tuning factor (normally one) to apply to the primal and dual error estimates.

68 Pass information to the subprogram FULMAT(), provided with the SPLP() package, so that a Fortran two-dimensional array can be used as the argument DATTRV(*).

69 Pass an absolute tolerance to use for the feasibility test when the usual relative error test indicates infeasibility. The nominal value of this tolerance, TOLABS, is zero.

Working Arrays:

WORK(*),LW,
IWORK(*),LIW

The arrays WORK(*) and IWORK(*) are respectively floating point and type INTEGER working arrays for SPLP( ) and its subprograms. The lengths of these arrays are respectively LW and LIW. These parameters must satisfy the inequalities noted above under the heading "Fortran Declarations Required." It is an error if either value is too small.

Input/Output files required:

Fortran unit 1 is used by SPLP( ) to store the sparse matrix A out of high-speed memory. This direct access file is opened within the package under the following two conditions. 1. When the Save/Restore feature is used. 2. When the constraint matrix is so large that storage out of high-speed memory is required. The user may need to close unit 1 (with deletion from the job step) in the main program unit when several calls are made to SPLP( ). A crude upper bound for the amount of information written on unit 1 is 6*nz, where nz is the number of nonzero entries in A. The unit number may be redefined to any other positive value by means of input in the option array PRGOPT(*).

Fortran unit 2 is used by SPLP( ) only when the Save/Restore feature is desired. Normally this feature is not used. It is activated by means of input in the option array PRGOPT(*). On some computer systems the user may need to open unit 2 before executing a call to SPLP( ). This file is type sequential and is unformatted.

Fortran unit=I1MACH(2) (check local setting) is used by SPLP( ) when the printed output feature (KEY=51) is used. Normally

this feature is not used. It is activated by input in the options array PRGOPT(*). For many computer systems I1MACH(2)=6.

```
|-------|
|Output:|
|-------|
```

INFO,PRIMAL(*),DUALS(*)

The integer flag INFO indicates why SPLP( ) has returned to the user. If INFO=1 the solution has been computed. In this case X(J)=PRIMAL(J) and W(I)=PRIMAL(I+NVARS). The dual variables for the equations A*x=w are in the array DUALS(I)=dual for equation number I. The dual value for the component X(J) that has an upper or lower bound (or both) is returned in DUALS(J+MRELAS). The only other values for INFO are .LT. 0. The meaning of these values can be found by reading the diagnostic message in the output file, or by looking for error number = (-INFO) under the heading "List of SPLP( ) Error and Diagnostic Messages."
The diagnostic messages are printed using the error processing subprogram XERMSG( ) with error category LEVEL=1.
See the document "Brief Instr. for Using the Sandia Math. Subroutine Library," SAND79-2382, Nov., 1980, for further information about resetting the usual response to a diagnostic message.

BL(*),BU(*),IND(*)

These arrays are output parameters only under the (unusual) circumstances where the stated problem is infeasible, has an unbounded optimum value, or both. These respective conditions correspond to INFO=-1,-2 or -3. For INFO=-1 or -3 certain components of the vectors x or w will not satisfy the input bounds. If component J of X or component I of W does not satisfy its input bound because of infeasibility, then IND(J)=-4 or IND(I+NVARS)=-4, respectively. For INFO=-2 or -3 certain components of the vector x could not be used as basic variables because the objective function would have become unbounded. In particular if component J of x corresponds to such a variable, then IND(J)=-3. Further, if the input value of IND(J)

```
                =1, then BU(J)=BL(J);
                =2, then BL(J)=BU(J);
                =4, then BL(J)=0.,BU(J)=0.
```

(The J-th variable in x has been restricted to an appropriate feasible value.)
The negative output value for IND(*) allows the user to identify those constraints that are not satisfied or those variables that would cause unbounded values of the objective function. Note that the absolute value of IND(*), together with BL(*) and BU(*), are valid input to SPLP( ). In the case of infeasibility the sum of magnitudes of the infeasible values is minimized. Thus one could reenter SPLP( ) with these components of x or w now fixed at their present values. This involves setting the appropriate components of IND(*) = 3, and BL(*) = BU(*).

IBASIS(I),I=1,...,MRELAS

This array contains the indices of the variables that are in the active basis set at the solution (INFO=1). A value

of IBASIS(I) between 1 and NVARS corresponds to the variable X(IBASIS(I)). A value of IBASIS(I) between NVARS+1 and NVARS+ MRELAS corresponds to the variable W(IBASIS(I)-NVARS).

Computing with the Matrix A after Calling SPLP( )

Following the return from SPLP( ), nonzero entries of the MRELAS by NVARS matrix A are available for usage by the user. The method for obtaining the next nonzero in column J with a row index strictly greater than I in value, is completed by executing

```
CALL PNNZRS(I,AIJ,IPLACE,WORK,IWORK,J)
```

The value of I is also an output parameter. If I.LE.0 on output, then there are no more nonzeroes in column J. If I.GT.0, the output value for component number I of column J is in AIJ. The parameters WORK(*) and IWORK(*) are the same arguments as in the call to SPLP( ). The parameter IPLACE is a single INTEGER working variable.

The data structure used for storage of the matrix A within SPLP( ) corresponds to sequential storage by columns as defined in SAND78-0785. Note that the names of the subprograms LNNZRS(), LCHNGS(),LINITM(),LLOC(),LRWPGE(), and LRWVIR() have been changed to PNNZRS(),PCHNGS(),PINITM(),IPLOC(),PRWPGE(), and PRWVIR() respectively. The error processing subprogram LERROR() is no longer used; XERMSG() is used instead.

|---|
Subprograms Required by SPLP( )

Called by SPLP() are SPLPMN(),SPLPUP(),SPINIT(),SPOPT(),
SPLPDM(),SPLPCE(),SPINCW(),SPLPFL(),
SPLPFE(),SPLPMU().

Error Processing Subprograms XERMSG(),I1MACH(),R1MACH()

Sparse Matrix Subprograms PNNZRS(),PCHNGS(),PRWPGE(),PRWVIR(),
PINITM(),IPLOC()

Mass Storage File Subprograms SOPENM(),SCLOSM(),SREADP(),SWRITP()

Basic Linear Algebra Subprograms SCOPY(),SASUM(),SDOT()

Sparse Matrix Basis Handling Subprograms LA05AS(),LA05BS(),
LA05CS(),LA05ED(),MC20AS()

Vector Output Subprograms SVOUT(),IVOUT()

Machine-sensitive Subprograms I1MACH( ),R1MACH( ),
SOPENM(),SCLOSM(),SREADP(),SWRITP().

COMMON Block Used

/LA05DS/ SMALL,LP,LENL,LENU,NCP,LROW,LCOL
See the document AERE-R8269 for further details.

|---|
Example of SPLP( ) Usage

```
PROGRAM LPEX
THE OPTIMIZATION PROBLEM IS TO FIND X1, X2, X3 THAT
```

```
 MINIMIZE X1 + X2 + X3, X1.GE.0, X2.GE.0, X3 UNCONSTRAINED.

 THE UNKNOWNS X1,X2,X3 ARE TO SATISFY CONSTRAINTS

    X1 -3*X2 +4*X3 = 5
    X1 -2*X2     .LE.3
        2*X2 - X3.GE.4

 WE FIRST DEFINE THE DEPENDENT VARIABLES
      W1=X1 -3*X2 +4*X3
      W2=X1- 2*X2
      W3=    2*X2 -X3

 WE NOW SHOW HOW TO USE SPLP( ) TO SOLVE THIS LINEAR OPTIMIZATION
 PROBLEM.  EACH REQUIRED STEP WILL BE SHOWN IN THIS EXAMPLE.
 DIMENSION COSTS(03),PRGOPT(01),DATTRV(18),BL(06),BU(06),IND(06),
*PRIMAL(06),DUALS(06),IBASIS(06),WORK(079),IWORK(103)

 EXTERNAL USRMAT
 MRELAS=3
 NVARS=3

 DEFINE THE ARRAY COSTS(*) FOR THE OBJECTIVE FUNCTION.
 COSTS(01)=1.
 COSTS(02)=1.
 COSTS(03)=1.

 PLACE THE NONZERO INFORMATION ABOUT THE MATRIX IN DATTRV(*).
 DEFINE COL. 1:
 DATTRV(01)=-1
 DATTRV(02)=1
 DATTRV(03)=1.
 DATTRV(04)=2
 DATTRV(05)=1.

 DEFINE COL. 2:
 DATTRV(06)=-2
 DATTRV(07)=1
 DATTRV(08)=-3.
 DATTRV(09)=2
 DATTRV(10)=-2.
 DATTRV(11)=3
 DATTRV(12)=2.

 DEFINE COL. 3:
 DATTRV(13)=-3
 DATTRV(14)=1
 DATTRV(15)=4.
 DATTRV(16)=3
 DATTRV(17)=-1.

 DATTRV(18)=0

 CONSTRAIN X1,X2 TO BE NONNEGATIVE. LET X3 HAVE NO BOUNDS.
 BL(1)=0.
 IND(1)=1
 BL(2)=0.
 IND(2)=1
 IND(3)=4
```

```
 CONSTRAIN W1=5,W2.LE.3, AND W3.GE.4.
 BL(4)=5.
 BU(4)=5.
 IND(4)=3
 BU(5)=3.
 IND(5)=2
 BL(6)=4.
 IND(6)=1

 INDICATE THAT NO MODIFICATIONS TO OPTIONS ARE IN USE.
 PRGOPT(01)=1

 DEFINE THE WORKING ARRAY LENGTHS.
 LW=079
 LIW=103
 CALL SPLP(USRMAT,MRELAS,NVARS,COSTS,PRGOPT,DATTRV,
*BL,BU,IND,INFO,PRIMAL,DUALS,IBASIS,WORK,LW,IWORK,LIW)

 CALCULATE VAL, THE MINIMAL VALUE OF THE OBJECTIVE FUNCTION.
 VAL=SDOT(NVARS,COSTS,1,PRIMAL,1)

 STOP
 END
|------------------------|
|End of Example of Usage |
|------------------------|
```

|------------------------------------|
Usage of SPLP( ) Subprogram Options.

Users frequently have a large variety of requirements for linear optimization software. Allowing for these varied requirements is at cross purposes with the desire to keep the usage of SPLP( ) as simple as possible. One solution to this dilemma is as follows. (1) Provide a version of SPLP( ) that solves a wide class of problems and is easy to use. (2) Identify parameters within SPLP() that certain users may want to change. (3) Provide a means of changing any selected number of these parameters that does not require changing all of them.

Changing selected parameters is done by requiring that the user provide an option array, PRGOPT(*), to SPLP( ). The contents of PRGOPT(*) inform SPLP( ) of just those options that are going to be modified within the total set of possible parameters that can be modified. The array PRGOPT(*) is a linked list consisting of groups of data of the following form

```
     LINK
     KEY
     SWITCH
     data set
```

that describe the desired options. The parameters LINK, KEY and switch are each one word and are always required. The data set can be comprised of several words or can be empty. The number of words in the data set for each option depends on the value of the parameter KEY.

The value of LINK points to the first entry of the next group

of data within PRGOPT(*). The exception is when there are no more options to change. In that case, LINK=1 and the values for KEY, SWITCH and data set are not referenced. The general layout of PRGOPT(*) is as follows:

```
     ...PRGOPT(1)=LINK1 (link to first entry of next group)
     .  PRGOPT(2)=KEY1 (KEY to the option change)
     .  PRGOPT(3)=SWITCH1 (on/off switch for the option)
     .  PRGOPT(4)=data value
     .       .
     .       .
     .       .
     ...PRGOPT(LINK1)=LINK2 (link to first entry of next group)
     .  PRGOPT(LINK1+1)=KEY2 (KEY to option change)
     .  PRGOPT(LINK1+2)=SWITCH2 (on/off switch for the option)
     .  PRGOPT(LINK1+3)=data value
     ...     .
     .       .
     .       .
     ...PRGOPT(LINK)=1 (no more options to change)
```

A value of LINK that is .LE.0 or .GT. 10000 is an error. In this case SPLP( ) returns with an error message, INFO=-14. This helps prevent using invalid but positive values of LINK that will probably extend beyond the program limits of PRGOPT(*). Unrecognized values of KEY are ignored. If the value of SWITCH is zero then the option is turned off. For any other value of SWITCH the option is turned on. This is used to allow easy changing of options without rewriting PRGOPT(*). The order of the options is arbitrary and any number of options can be changed with the following restriction. To prevent cycling in processing of the option array PRGOPT(*), a count of the number of options changed is maintained. Whenever this count exceeds 1000 an error message (INFO=-15) is printed and the subprogram returns.

In the following description of the options, the value of LATP indicates the amount of additional storage that a particular option requires. The sum of all of these values (plus one) is the minimum dimension for the array PRGOPT(*).

If a user is satisfied with the nominal form of SPLP( ), set PRGOPT(1)=1 (or PRGOPT(1)=1.E0).

Options:

-----KEY = 50. Change from a minimization problem to a maximization problem.
If SWITCH=0 option is off; solve minimization problem.
=1 option is on; solve maximization problem.
data set =empty
LATP=3

-----KEY = 51. Change the amount of printed output. The nominal form of SPLP( ) has no printed output.
The first level of output (SWITCH=1) includes

(1) Minimum dimensions for the arrays COSTS(*),BL(*),BU(*),IND(*), PRIMAL(*),DUALS(*),IBASIS(*), and PRGOPT(*).
(2) Problem dimensions MRELAS,NVARS.
(3) The types of and values for the bounds on x and w, and the values of the components of the vector costs.

```
    (4) Whether optimization problem is minimization or
        maximization.
    (5) Whether steepest edge or smallest reduced cost criteria used
        for exchanging variables in the revised simplex method.

    Whenever a solution has been found, (INFO=1),

    (6) the value of the objective function,
    (7) the values of the vectors x and w,
    (8) the dual variables for the constraints A*x=w and the
        bounded components of x,
    (9) the indices of the basic variables,
   (10) the number of revised simplex method iterations,
   (11) the number of full decompositions of the basis matrix.

    The second level of output (SWITCH=2) includes all for SWITCH=1
    plus

   (12) the iteration number,
   (13) the column number to enter the basis,
   (14) the column number to leave the basis,
   (15) the length of the step taken.

    The third level of output (SWITCH=3) includes all for SWITCH=2
    plus
   (16) critical quantities required in the revised simplex method.
         This output is rather voluminous.  It is intended to be used
         as a diagnostic tool in case of a failure in SPLP( ).

    If SWITCH=0  option is off; no printed output.
             =1  summary output.
             =2  lots of output.
             =3  even more output.
    data set =empty
    LATP=3

-----KEY = 52.  Redefine the parameter, IDIGIT, which determines the
    format and precision used for the printed output.  In the printed
    output, at least ABS(IDIGIT) decimal digits per number is printed.
    If IDIGIT.LT.0, 72 printing columns are used.  IF IDIGIT.GT.0, 133
    printing columns are used.
    If SWITCH=0  option is off; IDIGIT=-4.
             =1  option is on.
    data set =IDIGIT
    LATP=4

-----KEY = 53.  Redefine LAMAT and LBM, the lengths of the portions of
    WORK(*) and IWORK(*) that are allocated to the sparse matrix
    storage and the sparse linear equation solver, respectively.
    LAMAT must be .GE. NVARS+7 and LBM must be positive.
    If SWITCH=0  option is off; LAMAT=4*NVARS+7
                                LBM  =8*MRELAS.
             =1  option is on.
    data set =LAMAT
              LBM
    LATP=5

-----KEY = 54. Redefine IPAGEF, the file number where the pages of the
    sparse data matrix are stored.  IPAGEF must be positive and
    different from ISAVE (see option 56).
```

```
    If SWITCH=0  option is off; IPAGEF=1.
             =1  option is on.
    data set =IPAGEF
    LATP=4

-----KEY = 55.  Partial results have been computed and stored on unit
    number ISAVE (see option 56), during a previous run of
    SPLP( ).  This is a continuation from these partial results.
    The arrays COSTS(*),BL(*),BU(*),IND(*) do not have to have
    the same values as they did when the checkpointing occurred.
    This feature makes it possible for the user to do certain
    types of parameter studies such as changing costs and varying
    the constraints of the problem.  This file is rewound both be-
    fore and after reading the partial results.
    If SWITCH=0  option is off; start a new problem.
             =1  option is on; continue from partial results
                 that are stored in file ISAVE.
    data set = empty
    LATP=3

-----KEY = 56.  Redefine ISAVE, the file number where the partial
    results are stored (see option 57).  ISAVE must be positive and
    different from IPAGEF (see option 54).
    If SWITCH=0  option is off; ISAVE=2.
             =1  option is on.
    data set =ISAVE
    LATP=4

-----KEY = 57.  Save the partial results after maximum number of
    iterations, MAXITR, or at the optimum.  When this option is on,
    data essential to continuing the calculation is saved on a file
    using a Fortran binary write operation.  The data saved includes
    all the information about the sparse data matrix A.  Also saved
    is information about the current basis.  Nominally the partial
    results are saved on Fortran unit 2.  This unit number can be
    redefined (see option 56).  If the save option is on,
    this file must be opened (or declared) by the user prior to the
    call to SPLP( ).  A crude upper bound for the number of words
    written to this file is 6*nz.  Here nz= number of nonzeros in A.
    If SWITCH=0  option is off; do not save partial results.
             =1  option is on; save partial results.
    data set = empty
    LATP=3

-----KEY = 58.  Redefine the maximum number of iterations, MAXITR, to
    be taken before returning to the user.
    If SWITCH=0  option is off; MAXITR=3*(NVARS+MRELAS).
             =1  option is on.
    data set =MAXITR
    LATP=4

-----KEY = 59.  Provide SPLP( ) with exactly MRELAS indices which
    comprise a feasible, nonsingular basis.  The basis must define a
    feasible point: values for x and w such that A*x=w and all the
    stated bounds on x and w are satisfied.  The basis must also be
    nonsingular.  The failure of either condition will cause an error
    message (INFO=-23 or =-24, respectively).  Normally, SPLP( ) uses
    identity matrix columns which correspond to the components of w.
    This option would normally not be used when restarting from
    a previously saved run (KEY=57).
```

```
    In numbering the unknowns,
    the components of x are numbered (1-NVARS) and the components
    of w are numbered (NVARS+1)-(NVARS+MRELAS).  A value for an
    index .LE. 0 or .GT. (NVARS+MRELAS) is an error (INFO=-16).
    If SWITCH=0  option is off; SPLP( ) chooses the initial basis.
             =1  option is on; user provides the initial basis.
    data set =MRELAS indices of basis; order is arbitrary.
    LATP=MRELAS+3

-----KEY = 60.  Provide the scale factors for the columns of the data
    matrix A.  Normally, SPLP( ) computes the scale factors as the
    reciprocals of the max. norm of each column.
    If SWITCH=0  option is off; SPLP( ) computes the scale factors.
             =1  option is on; user provides the scale factors.
    data set =scaling for column J, J=1,NVARS; order is sequential.
    LATP=NVARS+3

-----KEY = 61.  Provide a scale factor, COSTSC, for the vector of
    costs.  Normally, SPLP( ) computes this scale factor to be the
    reciprocal of the max. norm of the vector costs after the column
    scaling has been applied.
    If SWITCH=0  option is off; SPLP( ) computes COSTSC.
             =1  option is on; user provides COSTSC.
    data set =COSTSC
    LATP=4

-----KEY = 62.  Provide size parameters, ASMALL and ABIG, the smallest
    and largest magnitudes of nonzero entries in the data matrix A,
    respectively.  When this option is on, SPLP( ) will check the
    nonzero entries of A to see if they are in the range of ASMALL and
    ABIG.  If an entry of A is not within this range, SPLP( ) returns
    an error message, INFO=-22. Both ASMALL and ABIG must be positive
    with ASMALL .LE. ABIG.  Otherwise,  an error message is returned,
    INFO=-17.
    If SWITCH=0  option is off; no checking of the data matrix is done
             =1  option is on; checking is done.
    data set =ASMALL
              ABIG
    LATP=5

-----KEY = 63.  Redefine the relative tolerance, TOLLS, used in
    checking if the residuals are feasible.  Normally,
    TOLLS=RELPR, where RELPR is the machine precision.
    If SWITCH=0  option is off; TOLLS=RELPR.
             =1  option is on.
    data set =TOLLS
    LATP=4

-----KEY = 64. Use the minimum reduced cost pricing strategy to choose
    columns to enter the basis.  Normally, SPLP( ) uses the steepest
    edge pricing strategy which is the best local move.  The steepest
    edge pricing strategy generally uses fewer iterations than the
    minimum reduced cost pricing, but each iteration costs more in the
    number of calculations done.  The steepest edge pricing is
    considered to be more efficient.  However, this is very problem
    dependent.  That is why SPLP( ) provides the option of either
    pricing strategy.
    If SWITCH=0  option is off; steepest option edge pricing is used.
             =1  option is on; minimum reduced cost pricing is used.
    data set =empty
```

```
    LATP=3

-----KEY = 65.  Redefine MXITBR, the number of iterations between
    recalculating the error in the primal solution.  Normally, MXITBR
    is set to 10.  The error in the primal solution is used to monitor
    the error in solving the linear system.  This is an expensive
    calculation and every tenth iteration is generally often enough.
    If SWITCH=0  option is off; MXITBR=10.
             =1  option is on.
    data set =MXITBR
    LATP=4

-----KEY = 66.  Redefine NPP, the number of negative reduced costs
    (at most) to be found at each iteration of choosing
    a variable to enter the basis.  Normally NPP is set
    to NVARS which implies that all of the reduced costs
    are computed at each such step.  This "partial
    pricing" may very well increase the total number
    of iterations required.  However it decreases the
    number of calculations at each iteration.
    therefore the effect on overall efficiency is quite
    problem-dependent.

    if SWITCH=0 option is off; NPP=NVARS
             =1 option is on.
    data set =NPP
    LATP=4

-----KEY =  67.  Redefine the tuning factor (PHI) used to scale the
    error estimates for the primal and dual linear algebraic systems
    of equations.  Normally, PHI = 1.E0, but in some environments it
    may be necessary to reset PHI to the range 0.001-0.01.  This is
    particularly important for machines with short word lengths.

    if SWITCH = 0 option is off; PHI=1.E0.
              = 1 option is on.
    Data Set  = PHI
    LATP=4

-----KEY = 68.  Used together with the subprogram FULMAT(), provided
    with the SPLP() package, for passing a standard Fortran two-
    dimensional array containing the constraint matrix.  Thus the sub-
    program FULMAT must be declared in a Fortran EXTERNAL statement.
    The two-dimensional array is passed as the argument DATTRV.
    The information about the array and problem dimensions are passed
    in the option array PRGOPT(*).  It is an error if FULMAT() is
    used and this information is not passed in PRGOPT(*).

    if SWITCH = 0 option is off; this is an error is FULMAT() is
                                 used.
              = 1 option is on.
    Data Set  = IA = row dimension of two-dimensional array.
                MRELAS = number of constraint equations.
                NVARS  = number of dependent variables.
    LATP = 6
-----KEY = 69.  Normally a relative tolerance (TOLLS, see option 63)
    is used to decide if the problem is feasible.  If this test fails
    an absolute test will be applied using the value TOLABS.
    Nominally TOLABS = zero.
    If SWITCH = 0 option is off; TOLABS = zero.
```

```
          = 1 option is on.
 Data set  = TOLABS
 LATP = 4

|-----------------------------|
|Example of Option array Usage|
|-----------------------------|
 To illustrate the usage of the option array, let us suppose that
 the user has the following nonstandard requirements:

      a) Wants to change from minimization to maximization problem.
      b) Wants to limit the number of simplex steps to 100.
      c) Wants to save the partial results after 100 steps on
         Fortran unit 2.

 After these 100 steps are completed the user wants to continue the
 problem (until completed) using the partial results saved on
 Fortran unit 2.  Here are the entries of the array PRGOPT(*)
 that accomplish these tasks.  (The definitions of the other
 required input parameters are not shown.)

 CHANGE TO A MAXIMIZATION PROBLEM; KEY=50.
 PRGOPT(01)=4
 PRGOPT(02)=50
 PRGOPT(03)=1

 LIMIT THE NUMBER OF SIMPLEX STEPS TO 100; KEY=58.
 PRGOPT(04)=8
 PRGOPT(05)=58
 PRGOPT(06)=1
 PRGOPT(07)=100

 SAVE THE PARTIAL RESULTS, AFTER 100 STEPS, ON FORTRAN
 UNIT 2; KEY=57.
 PRGOPT(08)=11
 PRGOPT(09)=57
 PRGOPT(10)=1

 NO MORE OPTIONS TO CHANGE.
 PRGOPT(11)=1
 The user makes the CALL statement for SPLP( ) at this point.
 Now to restart, using the partial results after 100 steps, define
 new values for the array PRGOPT(*):

 AGAIN INFORM SPLP( ) THAT THIS IS A MAXIMIZATION PROBLEM.
 PRGOPT(01)=4
 PRGOPT(02)=50
 PRGOPT(03)=1

 RESTART, USING SAVED PARTIAL RESULTS; KEY=55.
 PRGOPT(04)=7
 PRGOPT(05)=55
 PRGOPT(06)=1

 NO MORE OPTIONS TO CHANGE.  THE SUBPROGRAM SPLP( ) IS NO LONGER
 LIMITED TO 100 SIMPLEX STEPS BUT WILL RUN UNTIL COMPLETION OR
 MAX.=3*(MRELAS+NVARS) ITERATIONS.
 PRGOPT(07)=1
 The user now makes a CALL to subprogram SPLP( ) to compute the
 solution.
```

```
|-------------------------------------------|
|End of Usage of SPLP( ) Subprogram Options.|
|-------------------------------------------|

 |-----------------------------------------------|
 |List of SPLP( ) Error and Diagnostic Messages.|
 |-----------------------------------------------|
  This section may be required to understand the meanings of the
 error flag =-INFO  that may be returned from SPLP( ).

-----1. There is no set of values for x and w that satisfy A*x=w and
    the stated bounds.  The problem can be made feasible by ident-
    ifying components of w that are now infeasible and then rede-
    signating them as free variables.  Subprogram SPLP( ) only
    identifies an infeasible problem; it takes no other action to
    change this condition.  Message:
    SPLP( ). THE PROBLEM APPEARS TO BE INFEASIBLE.
    ERROR NUMBER =         1

    2. One of the variables in either the vector x or w was con-
    strained at a bound.  Otherwise the objective function value,
    (transpose of costs)*x, would not have a finite optimum.
    Message:
    SPLP( ). THE PROBLEM APPEARS TO HAVE NO FINITE SOLN.
    ERROR NUMBER =         2

    3.  Both of the conditions of 1. and 2. above have occurred.
    Message:
    SPLP( ). THE PROBLEM APPEARS TO BE INFEASIBLE AND TO
    HAVE NO FINITE SOLN.
    ERROR NUMBER =         3

-----4.  The REAL and INTEGER working arrays, WORK(*) and IWORK(*),
    are not long enough. The values (I1) and (I2) in the message
    below will give you the minimum length required.  Also redefine
    LW and LIW, the lengths of these arrays.  Message:
    SPLP( ). WORK OR IWORK IS NOT LONG ENOUGH. LW MUST BE (I1)
    AND LIW MUST BE (I2).
              IN ABOVE MESSAGE, I1=         0
              IN ABOVE MESSAGE, I2=         0
    ERROR NUMBER =        4

-----5. and 6.  These error messages often mean that one or more
    arguments were left out of the call statement to SPLP( ) or
    that the values of MRELAS and NVARS have been over-written
    by garbage.  Messages:
    SPLP( ). VALUE OF MRELAS MUST BE .GT.0. NOW=(I1).
              IN ABOVE MESSAGE, I1=         0
    ERROR NUMBER =         5

    SPLP( ). VALUE OF NVARS MUST BE .GT.0. NOW=(I1).
              IN ABOVE MESSAGE, I1=         0
    ERROR NUMBER =         6

-----7.,8., and 9.  These error messages can occur as the data matrix
    is being defined by either USRMAT( ) or the user-supplied sub-
    program, 'NAME'( ). They would indicate a mistake in the contents
    of DATTRV(*), the user-written subprogram or that data has been
    over-written.
    Messages:
```

```
    SPLP( ). MORE THAN 2*NVARS*MRELAS ITERS. DEFINING OR UPDATING
    MATRIX DATA.
    ERROR NUMBER =        7

    SPLP( ). ROW INDEX (I1) OR COLUMN INDEX (I2) IS OUT OF RANGE.
              IN ABOVE MESSAGE, I1=         1
              IN ABOVE MESSAGE, I2=        12
    ERROR NUMBER =        8

    SPLP( ). INDICATION FLAG (I1) FOR MATRIX DATA MUST BE
    EITHER 0 OR 1.
              IN ABOVE MESSAGE, I1=        12
    ERROR NUMBER =        9

-----10. and 11.  The type of bound (even no bound) and the bounds
    must be specified for each independent variable. If an independent
    variable has both an upper and lower bound, the bounds must be
    consistent.  The lower bound must be .LE. the upper bound.
    Messages:
    SPLP( ). INDEPENDENT VARIABLE (I1) IS NOT DEFINED.
              IN ABOVE MESSAGE, I1=         1
    ERROR NUMBER =        10

    SPLP( ).  LOWER BOUND (R1) AND UPPER BOUND (R2) FOR INDEP.
    VARIABLE (I1) ARE NOT CONSISTENT.
              IN ABOVE MESSAGE, I1=         1
              IN ABOVE MESSAGE, R1=    0.
              IN ABOVE MESSAGE, R2=    -.1000000000E+01
    ERROR NUMBER =        11

-----12. and 13.  The type of bound (even no bound) and the bounds
    must be specified for each dependent variable.  If a dependent
    variable has both an upper and lower bound, the bounds must be
    consistent. The lower bound must be .LE. the upper bound.
    Messages:
    SPLP( ). DEPENDENT VARIABLE (I1) IS NOT DEFINED.
              IN ABOVE MESSAGE, I1=         1
    ERROR NUMBER =        12

    SPLP( ).  LOWER BOUND (R1) AND UPPER BOUND (R2) FOR DEP.
     VARIABLE (I1) ARE NOT CONSISTENT.
              IN ABOVE MESSAGE, I1=         1
              IN ABOVE MESSAGE, R1=    0.
              IN ABOVE MESSAGE, R2=    -.1000000000E+01
    ERROR NUMBER =        13

-----14. - 21.  These error messages can occur when processing the
    option array, PRGOPT(*), supplied by the user.  They would
    indicate a mistake in defining PRGOPT(*) or that data has been
    over-written.  See heading Usage of SPLP( )
    Subprogram Options, for details on how to define PRGOPT(*).
    Messages:
    SPLP( ). THE USER OPTION ARRAY HAS UNDEFINED DATA.
    ERROR NUMBER =        14

    SPLP( ). OPTION ARRAY PROCESSING IS CYCLING.
    ERROR NUMBER =        15

    SPLP( ). AN INDEX OF USER-SUPPLIED BASIS IS OUT OF RANGE.
    ERROR NUMBER =        16
```

```
    SPLP( ). SIZE PARAMETERS FOR MATRIX MUST BE SMALLEST AND LARGEST
    MAGNITUDES OF NONZERO ENTRIES.
    ERROR NUMBER =        17

    SPLP( ). THE NUMBER OF REVISED SIMPLEX STEPS BETWEEN CHECK-POINTS
    MUST BE POSITIVE.
    ERROR NUMBER =        18

    SPLP( ). FILE NUMBERS FOR SAVED DATA AND MATRIX PAGES MUST BE
    POSITIVE AND NOT EQUAL.
    ERROR NUMBER =        19

    SPLP( ). USER-DEFINED VALUE OF LAMAT (I1)
    MUST BE .GE. NVARS+7.
              IN ABOVE MESSAGE, I1=         1
    ERROR NUMBER =         20

    SPLP( ). USER-DEFINED VALUE OF LBM MUST BE .GE. 0.
    ERROR NUMBER =         21

-----22.  The user-option, number 62, to check the size of the matrix
    data has been used.  An element of the matrix does not lie within
    the range of ASMALL and ABIG, parameters provided by the user.
    (See the heading: Usage of SPLP( ) Subprogram Options,
    for details about this feature.)  Message:
    SPLP( ). A MATRIX ELEMENT'S SIZE IS OUT OF THE SPECIFIED RANGE.
    ERROR NUMBER =        22

-----23.  The user has provided an initial basis that is singular.
    In this case, the user can remedy this problem by letting
    subprogram SPLP( ) choose its own initial basis.  Message:
    SPLP( ). A SINGULAR INITIAL BASIS WAS ENCOUNTERED.
    ERROR NUMBER =         23

-----24.  The user has provided an initial basis which is infeasible.
    The x and w values it defines do not satisfy A*x=w and the stated
    bounds.  In this case, the user can let subprogram SPLP( )
    choose its own initial basis.  Message:
    SPLP( ). AN INFEASIBLE INITIAL BASIS WAS ENCOUNTERED.
    ERROR NUMBER =        24

-----25. Subprogram SPLP( ) has completed the maximum specified number
    of iterations.  (The nominal maximum number is 3*(MRELAS+NVARS).)
    The results, necessary to continue on from
    this point, can be saved on Fortran unit 2 by activating option
    KEY=57.  If the user anticipates continuing the calculation, then
    the contents of Fortran unit 2 must be retained intact.  This
    is not done by subprogram SPLP( ), so the user needs to save unit
    2 by using the appropriate system commands.  Message:
    SPLP( ). MAX. ITERS. (I1) TAKEN. UP-TO-DATE RESULTS
    SAVED ON FILE (I2). IF(I2)=0, NO SAVE.
              IN ABOVE MESSAGE, I1=       500
              IN ABOVE MESSAGE, I2=         2
    ERROR NUMBER =        25

-----26.  This error should never happen.  Message:
    SPLP( ). MOVED TO A SINGULAR POINT. THIS SHOULD NOT HAPPEN.
    ERROR NUMBER =        26
```

```
 -----27.  The subprogram LA05A( ), which decomposes the basis matrix,
     has returned with an error flag (R1).  (See the document,
     "Fortran subprograms for handling sparse linear programming
     bases", AERE-R8269, J.K. Reid, Jan., 1976, H.M. Stationery Office,
     for an explanation of this error.)  Message:
     SPLP( ). LA05A( ) RETURNED ERROR FLAG (R1) BELOW.
               IN ABOVE MESSAGE, R1=    -.5000000000E+01
     ERROR NUMBER =        27

 -----28.  The sparse linear solver package, LA05*( ), requires more
     space.  The value of LBM must be increased.  See the companion
     document, Usage of SPLP( ) Subprogram Options, for details on how
     to increase the value of LBM.  Message:
     SPLP( ). SHORT ON STORAGE FOR LA05*( ) PACKAGE. USE PRGOPT(*)
     TO GIVE MORE.
     ERROR NUMBER =        28

 -----29.  The row dimension of the two-dimensional Fortran array,
     the number of constraint equations (MRELAS), and the number
     of variables (NVARS), were not passed to the subprogram
     FULMAT().  See KEY = 68 for details.  Message:
     FULMAT() OF SPLP() PACKAGE. ROW DIM., MRELAS, NVARS ARE
     MISSING FROM PRGOPT(*).
     ERROR NUMBER =        29

     |-------------------------------------------------------|
     |End of List of SPLP( ) Error and Diagnostic Messages. |
     |-------------------------------------------------------|
***REFERENCES  R. J. Hanson and K. L. Hiebert, A sparse linear
                 programming subprogram, Report SAND81-0297, Sandia
                 National Laboratories, 1981.
***ROUTINES CALLED  SPLPMN, XERMSG
***REVISION HISTORY  (YYMMDD)
   811215  DATE WRITTEN
   890605  Corrected references to XERRWV.  (WRB)
   890605  Removed unreferenced labels.  (WRB)
   890605  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900510  Convert XERRWV calls to XERMSG calls.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SPOCO

```
      SUBROUTINE SPOCO (A, LDA, N, RCOND, Z, INFO)
***BEGIN PROLOGUE  SPOCO
***PURPOSE  Factor a real symmetric positive definite matrix
            and estimate the condition number of the matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B
***TYPE      SINGLE PRECISION (SPOCO-S, DPOCO-D, CPOCO-C)
***KEYWORDS  CONDITION NUMBER, LINEAR ALGEBRA, LINPACK,
             MATRIX FACTORIZATION, POSITIVE DEFINITE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SPOCO factors a real symmetric positive definite matrix
      and estimates the condition of the matrix.

      If  RCOND  is not needed, SPOFA is slightly faster.
      To solve  A*X = B , follow SPOCO by SPOSL.
      To compute  INVERSE(A)*C , follow SPOCO by SPOSL.
      To compute  DETERMINANT(A) , follow SPOCO by SPODI.
      To compute  INVERSE(A) , follow SPOCO by SPODI.

      On Entry

         A       REAL(LDA, N)
                 the symmetric matrix to be factored.  Only the
                 diagonal and upper triangle are used.

         LDA     INTEGER
                 the leading dimension of the array  A .

         N       INTEGER
                 the order of the matrix  A .

      On Return

         A       an upper triangular matrix  R  so that  A = TRANS(R)*R
                 where  TRANS(R)  is the transpose.
                 The strict lower triangle is unaltered.
                 If  INFO .NE. 0 , the factorization is not complete.

         RCOND   REAL
                 an estimate of the reciprocal condition of  A .
                 For the system  A*X = B , relative perturbations
                 in  A  and  B  of size  EPSILON  may cause
                 relative perturbations in  X  of size  EPSILON/RCOND .
                 If  RCOND  is so small that the logical expression
                           1.0 + RCOND .EQ. 1.0
                 is true, then  A  may be singular to working
                 precision.  In particular,  RCOND  is zero  if
                 exact singularity is detected or the estimate
                 underflows.  If INFO .NE. 0 , RCOND is unchanged.

         Z       REAL(N)
                 a work vector whose contents are usually unimportant.
                 If  A  is close to a singular matrix, then  Z  is
                 an approximate null vector in the sense that
```

```
                NORM(A*Z) = RCOND*NORM(A)*NORM(Z) .
                If  INFO .NE. 0 , Z  is unchanged.

        INFO    INTEGER
                = 0  for normal return.
                = K  signals an error condition.  The leading minor
                     of order  K  is not positive definite.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SASUM, SAXPY, SDOT, SPOFA, SSCAL
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SPODI

```
      SUBROUTINE SPODI (A, LDA, N, DET, JOB)
***BEGIN PROLOGUE  SPODI
***PURPOSE  Compute the determinant and inverse of a certain real
            symmetric positive definite matrix using the factors
            computed by SPOCO, SPOFA or SQRDC.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B, D3B1B
***TYPE      SINGLE PRECISION (SPODI-S, DPODI-D, CPODI-C)
***KEYWORDS  DETERMINANT, INVERSE, LINEAR ALGEBRA, LINPACK, MATRIX,
             POSITIVE DEFINITE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SPODI computes the determinant and inverse of a certain
      real symmetric positive definite matrix (see below)
      using the factors computed by SPOCO, SPOFA or SQRDC.

      On Entry

         A       REAL(LDA, N)
                 the output  A  from SPOCO or SPOFA
                 or the output  X  from SQRDC.

         LDA     INTEGER
                 the leading dimension of the array  A .

         N       INTEGER
                 the order of the matrix  A .

         JOB     INTEGER
                 = 11   both determinant and inverse.
                 = 01   inverse only.
                 = 10   determinant only.

      On Return

         A       If SPOCO or SPOFA was used to factor  A , then
                 SPODI produces the upper half of INVERSE(A) .
                 If SQRDC was used to decompose  X , then
                 SPODI produces the upper half of INVERSE(TRANS(X)*X),
                 where TRANS(X) is the transpose.
                 Elements of  A  below the diagonal are unchanged.
                 If the units digit of JOB is zero,  A  is unchanged.

         DET     REAL(2)
                 determinant of  A  or of  TRANS(X)*X  if requested.
                 Otherwise not referenced.
                 Determinant = DET(1) * 10.0**DET(2)
                 with  1.0 .LE. DET(1) .LT. 10.0
                 or  DET(1) .EQ. 0.0 .

      Error Condition

         A division by zero will occur if the input factor contains
         a zero on the diagonal and the inverse is requested.
         It will not occur if the subroutines are called correctly
```

```
        and if SPOCO or SPOFA has set INFO .EQ. 0 .

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SSCAL
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SPOFA

```
      SUBROUTINE SPOFA (A, LDA, N, INFO)
***BEGIN PROLOGUE  SPOFA
***PURPOSE  Factor a real symmetric positive definite matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B
***TYPE      SINGLE PRECISION (SPOFA-S, DPOFA-D, CPOFA-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX FACTORIZATION,
             POSITIVE DEFINITE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SPOFA factors a real symmetric positive definite matrix.

      SPOFA is usually called by SPOCO, but it can be called
      directly with a saving in time if  RCOND  is not needed.
      (Time for SPOCO) = (1 + 18/N)*(Time for SPOFA) .

      On Entry

         A       REAL(LDA, N)
                 the symmetric matrix to be factored.  Only the
                 diagonal and upper triangle are used.

         LDA     INTEGER
                 the leading dimension of the array  A .

         N       INTEGER
                 the order of the matrix  A .

      On Return

         A       an upper triangular matrix  R  so that  A = TRANS(R)*R
                 where  TRANS(R)  is the transpose.
                 The strict lower triangle is unaltered.
                 If  INFO .NE. 0 , the factorization is not complete.

         INFO    INTEGER
                 = 0  for normal return.
                 = K  signals an error condition.  The leading minor
                      of order  K  is not positive definite.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SDOT
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SPOFS

```
      SUBROUTINE SPOFS (A, LDA, N, V, ITASK, IND, WORK)
***BEGIN PROLOGUE  SPOFS
***PURPOSE  Solve a positive definite symmetric system of linear
            equations.
***LIBRARY   SLATEC
***CATEGORY  D2B1B
***TYPE      SINGLE PRECISION (SPOFS-S, DPOFS-D, CPOFS-C)
***KEYWORDS  HERMITIAN, LINEAR EQUATIONS, POSITIVE DEFINITE, SYMMETRIC
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

    Subroutine SPOFS solves a real positive definite symmetric
    NxN system of single precision linear equations using
    LINPACK subroutines SPOCO and SPOSL.  That is, if A is an
    NxN real positive definite symmetric matrix and if X and B
    are real N-vectors, then SPOFS solves the equation

                           A*X=B.

    The matrix A is first factored into upper and lower tri-
    angular matrices R and R-TRANSPOSE.  These factors are used to
    find the solution vector X.  An approximate condition number is
    calculated to provide a rough estimate of the number of
    digits of accuracy in the computed solution.

    If the equation A*X=B is to be solved for more than one vector
    B, the factoring of A does not need to be performed again and
    the option to solve only (ITASK .GT. 1) will be faster for
    the succeeding solutions.  In this case, the contents of A,
    LDA, and N must not have been altered by the user following
    factorization (ITASK=1).  IND will not be changed by SPOFS
    in this case.

  Argument Description ***

    A      REAL(LDA,N)
             on entry, the doubly subscripted array with dimension
               (LDA,N) which contains the coefficient matrix.  Only
               the upper triangle, including the diagonal, of the
               coefficient matrix need be entered and will subse-
               quently be referenced and changed by the routine.
             on return, contains in its upper triangle an upper
               triangular matrix R such that  A = (R-TRANSPOSE) * R .
    LDA    INTEGER
             the leading dimension of the array A.  LDA must be great-
             er than or equal to N.  (Terminal error message IND=-1)
    N      INTEGER
             the order of the matrix A.  N must be greater
             than or equal to 1.  (Terminal error message IND=-2)
    V      REAL(N)
             on entry, the singly subscripted array(vector) of di-
               mension N which contains the right hand side B of a
               system of simultaneous linear equations A*X=B.
             on return, V contains the solution vector, X .
    ITASK  INTEGER
             If ITASK = 1, the matrix A is factored and then the
```

```
              linear equation is solved.
            If ITASK .GT. 1, the equation is solved using the existing
              factored matrix A.
            If ITASK .LT. 1, then terminal error message IND=-3 is
              printed.
   IND    INTEGER
            GT. 0  IND is a rough estimate of the number of digits
                    of accuracy in the solution, X.
            LT. 0  see error message corresponding to IND below.
   WORK   REAL(N)
            a singly subscripted array of dimension at least N.

  Error Messages Printed ***

   IND=-1  terminal   N is greater than LDA.
   IND=-2  terminal   N is less than 1.
   IND=-3  terminal   ITASK is less than 1.
   IND=-4  Terminal   The matrix A is computationally singular or
                        is not positive definite.  A solution
                        has not been computed.
   IND=-10 warning    The solution has no apparent significance.
                        The solution may be inaccurate or the
                        matrix A may be poorly scaled.

              Note-  The above terminal(*fatal*) error messages are
                     designed to be handled by XERMSG in which
                     LEVEL=1 (recoverable) and IFLAG=2 .  LEVEL=0
                     for warning error messages from XERMSG.  Unless
                     the user provides otherwise, an error message
                     will be printed followed by an abort.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  R1MACH, SPOCO, SPOSL, XERMSG
***REVISION HISTORY  (YYMMDD)
   800509  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900510  Convert XERRWV calls to XERMSG calls.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SPOIR

```
      SUBROUTINE SPOIR (A, LDA, N, V, ITASK, IND, WORK)
***BEGIN PROLOGUE  SPOIR
***PURPOSE  Solve a positive definite symmetric system of linear
            equations.  Iterative refinement is used to obtain an error
            estimate.
***LIBRARY   SLATEC
***CATEGORY  D2B1B
***TYPE      SINGLE PRECISION (SPOIR-S, CPOIR-C)
***KEYWORDS  HERMITIAN, LINEAR EQUATIONS, POSITIVE DEFINITE, SYMMETRIC
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

    Subroutine SPOIR solves a real positive definite symmetric
    NxN system of single precision linear equations using LINPACK
    subroutines SPOFA and SPOSL.  One pass of iterative refine-
    ment is used only to obtain an estimate of the accuracy.  That
    is, if A is an NxN real positive definite symmetric matrix
    and if X and B are real N-vectors, then SPOIR solves the
    equation

                           A*X=B.

    The matrix A is first factored into upper and lower
    triangular matrices R and R-TRANSPOSE.  These
    factors are used to calculate the solution, X.
    Then the residual vector is found and used
    to calculate an estimate of the relative error, IND.
    IND estimates the accuracy of the solution only when the
    input matrix and the right hand side are represented
    exactly in the computer and does not take into account
    any errors in the input data.

    If the equation A*X=B is to be solved for more than one vector
    B, the factoring of A does not need to be performed again and
    the option to only solve (ITASK .GT. 1) will be faster for
    the succeeding solutions.  In this case, the contents of A,
    LDA, N, and WORK must not have been altered by the user
    following factorization (ITASK=1).  IND will not be changed
    by SPOIR in this case.

  Argument Description ***
    A      REAL(LDA,N)
             the doubly subscripted array with dimension (LDA,N)
             which contains the coefficient matrix.  Only the
             upper triangle, including the diagonal, of the
             coefficient matrix need be entered.  A is not
             altered by the routine.
    LDA    INTEGER
             the leading dimension of the array A.  LDA must be great-
             er than or equal to N.  (Terminal error message IND=-1)
    N      INTEGER
             the order of the matrix A.  N must be greater than
             or equal to one.  (Terminal error message IND=-2)
    V      REAL(N)
             on entry, the singly subscripted array(vector) of di-
               mension N which contains the right hand side B of a
```

```
                system of simultaneous linear equations A*X=B.
              on return, V contains the solution vector, X .
    ITASK  INTEGER
              If ITASK = 1, the matrix A is factored and then the
                linear equation is solved.
              If ITASK .GT. 1, the equation is solved using the existing
                factored matrix A (stored in WORK).
              If ITASK .LT. 1, then terminal terminal error IND=-3 is
                printed.
    IND    INTEGER
              GT. 0  IND is a rough estimate of the number of digits
                      of accuracy in the solution, X.  IND=75 means
                      that the solution vector X is zero.
              LT. 0  See error message corresponding to IND below.
    WORK   REAL(N*(N+1))
              a singly subscripted array of dimension at least N*(N+1).

  Error Messages Printed ***

    IND=-1  terminal   N is greater than LDA.
    IND=-2  terminal   N is less than one.
    IND=-3  terminal   ITASK is less than one.
    IND=-4  Terminal   The matrix A is computationally singular
                         or is not positive definite.
                         A solution has not been computed.
    IND=-10 warning    The solution has no apparent significance.
                         The solution may be inaccurate or the matrix
                         A may be poorly scaled.

                Note-  The above terminal(*fatal*) error messages are
                       designed to be handled by XERMSG in which
                       LEVEL=1 (recoverable) and IFLAG=2 .  LEVEL=0
                       for warning error messages from XERMSG.  Unless
                       the user provides otherwise, an error message
                       will be printed followed by an abort.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  DSDOT, R1MACH, SASUM, SCOPY, SPOFA, SPOSL, XERMSG
***REVISION HISTORY  (YYMMDD)
   800528  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900510  Convert XERRWV calls to XERMSG calls.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SPOSL

```
      SUBROUTINE SPOSL (A, LDA, N, B)
***BEGIN PROLOGUE  SPOSL
***PURPOSE  Solve the real symmetric positive definite linear system
            using the factors computed by SPOCO or SPOFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B
***TYPE      SINGLE PRECISION (SPOSL-S, DPOSL-D, CPOSL-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX, POSITIVE DEFINITE, SOLVE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SPOSL solves the real symmetric positive definite system
      A * X = B
      using the factors computed by SPOCO or SPOFA.

      On Entry

         A       REAL(LDA, N)
                 the output from SPOCO or SPOFA.

         LDA     INTEGER
                 the leading dimension of the array  A .

         N       INTEGER
                 the order of the matrix  A .

         B       REAL(N)
                 the right hand side vector.

      On Return

         B       the solution vector  X .

      Error Condition

         A division by zero will occur if the input factor contains
         a zero on the diagonal.  Technically, this indicates
         singularity, but it is usually caused by improper subroutine
         arguments.  It will not occur if the subroutines are called
         correctly and  INFO .EQ. 0 .

      To compute  INVERSE(A) * C  where  C  is a matrix
      with  P  columns
            CALL SPOCO(A,LDA,N,RCOND,Z,INFO)
            IF (RCOND is too small .OR. INFO .NE. 0) GO TO ...
            DO 10 J = 1, P
               CALL SPOSL(A,LDA,N,C(1,J))
         10 CONTINUE

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SDOT
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
```

```
891214  Prologue converted to Version 4.0 format.  (BAB)
900326  Removed duplicate information from DESCRIPTION section.
        (WRB)
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

SPPCO

```
      SUBROUTINE SPPCO (AP, N, RCOND, Z, INFO)
***BEGIN PROLOGUE  SPPCO
***PURPOSE  Factor a symmetric positive definite matrix stored in
            packed form and estimate the condition number of the
            matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B
***TYPE      SINGLE PRECISION (SPPCO-S, DPPCO-D, CPPCO-C)
***KEYWORDS  CONDITION NUMBER, LINEAR ALGEBRA, LINPACK,
             MATRIX FACTORIZATION, PACKED, POSITIVE DEFINITE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SPPCO factors a real symmetric positive definite matrix
      stored in packed form
      and estimates the condition of the matrix.

      If  RCOND  is not needed, SPPFA is slightly faster.
      To solve  A*X = B , follow SPPCO by SPPSL.
      To compute  INVERSE(A)*C , follow SPPCO by SPPSL.
      To compute  DETERMINANT(A) , follow SPPCO by SPPDI.
      To compute  INVERSE(A) , follow SPPCO by SPPDI.

      On Entry

         AP      REAL (N*(N+1)/2)
                 the packed form of a symmetric matrix  A .  The
                 columns of the upper triangle are stored sequentially
                 in a one-dimensional array of length  N*(N+1)/2 .
                 See comments below for details.

         N       INTEGER
                 the order of the matrix  A .

      On Return

         AP      an upper triangular matrix  R , stored in packed
                 form, so that  A = TRANS(R)*R .
                 If  INFO .NE. 0 , the factorization is not complete.

         RCOND   REAL
                 an estimate of the reciprocal condition of  A .
                 For the system  A*X = B , relative perturbations
                 in  A  and  B  of size  EPSILON  may cause
                 relative perturbations in  X  of size  EPSILON/RCOND .
                 If  RCOND  is so small that the logical expression
                           1.0 + RCOND .EQ. 1.0
                 is true, then  A  may be singular to working
                 precision.  In particular,  RCOND  is zero  if
                 exact singularity is detected or the estimate
                 underflows.  If INFO .NE. 0 , RCOND is unchanged.

         Z       REAL(N)
                 a work vector whose contents are usually unimportant.
                 If  A  is singular to working precision, then  Z  is
                 an approximate null vector in the sense that
```

```
                NORM(A*Z) = RCOND*NORM(A)*NORM(Z) .
                If  INFO .NE. 0 , Z  is unchanged.

        INFO    INTEGER
                = 0  for normal return.
                = K  signals an error condition.  The leading minor
                     of order  K  is not positive definite.

     Packed Storage

          The following program segment will pack the upper
          triangle of a symmetric matrix.

                K = 0
                DO 20 J = 1, N
                   DO 10 I = 1, J
                      K = K + 1
                      AP(K) = A(I,J)
             10    CONTINUE
             20 CONTINUE

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SASUM, SAXPY, SDOT, SPPFA, SSCAL
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SPPDI

```
      SUBROUTINE SPPDI (AP, N, DET, JOB)
***BEGIN PROLOGUE  SPPDI
***PURPOSE  Compute the determinant and inverse of a real symmetric
            positive definite matrix using factors from SPPCO or SPPFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B, D3B1B
***TYPE      SINGLE PRECISION (SPPDI-S, DPPDI-D, CPPDI-C)
***KEYWORDS  DETERMINANT, INVERSE, LINEAR ALGEBRA, LINPACK, MATRIX,
             PACKED, POSITIVE DEFINITE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SPPDI computes the determinant and inverse
      of a real symmetric positive definite matrix
      using the factors computed by SPPCO or SPPFA .

      On Entry

         AP      REAL (N*(N+1)/2)
                 the output from SPPCO or SPPFA.

         N       INTEGER
                 the order of the matrix  A .

         JOB     INTEGER
                 = 11   both determinant and inverse.
                 = 01   inverse only.
                 = 10   determinant only.

      On Return

         AP      the upper triangular half of the inverse .
                 The strict lower triangle is unaltered.

         DET     REAL(2)
                 determinant of original matrix if requested.
                 Otherwise not referenced.
                 Determinant = DET(1) * 10.0**DET(2)
                 with  1.0 .LE. DET(1) .LT. 10.0
                 or  DET(1) .EQ. 0.0 .

      Error Condition

         A division by zero will occur if the input factor contains
         a zero on the diagonal and the inverse is requested.
         It will not occur if the subroutines are called correctly
         and if SPOCO or SPOFA has set INFO .EQ. 0 .

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SSCAL
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
```

```
900326  Removed duplicate information from DESCRIPTION section.
        (WRB)
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

SPPERM

```
      SUBROUTINE SPPERM (X, N, IPERM, IER)
***BEGIN PROLOGUE  SPPERM
***PURPOSE  Rearrange a given array according to a prescribed
            permutation vector.
***LIBRARY   SLATEC
***CATEGORY  N8
***TYPE      SINGLE PRECISION (SPPERM-S, DPPERM-D, IPPERM-I, HPPERM-H)
***KEYWORDS  APPLICATION OF PERMUTATION TO DATA VECTOR
***AUTHOR  McClain, M. A., (NIST)
           Rhoads, G. S., (NBS)
***DESCRIPTION

         SPPERM rearranges the data vector X according to the
         permutation IPERM: X(I) <--- X(IPERM(I)).  IPERM could come
         from one of the sorting routines IPSORT, SPSORT, DPSORT or
         HPSORT.

      Description of Parameters
         X - input/output -- real array of values to be rearranged.
         N - input -- number of values in real array X.
         IPERM - input -- permutation vector.
         IER - output -- error indicator:
             =  0  if no error,
             =  1  if N is zero or negative,
             =  2  if IPERM is not a valid permutation.

***REFERENCES  (NONE)
***ROUTINES CALLED  XERMSG
***REVISION HISTORY  (YYMMDD)
   901004  DATE WRITTEN
   920507  Modified by M. McClain to revise prologue text.
   END PROLOGUE
```

SPPFA

```
      SUBROUTINE SPPFA (AP, N, INFO)
***BEGIN PROLOGUE  SPPFA
***PURPOSE  Factor a real symmetric positive definite matrix stored in
            packed form.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B
***TYPE      SINGLE PRECISION (SPPFA-S, DPPFA-D, CPPFA-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX FACTORIZATION, PACKED,
             POSITIVE DEFINITE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SPPFA factors a real symmetric positive definite matrix
      stored in packed form.

      SPPFA is usually called by SPPCO, but it can be called
      directly with a saving in time if  RCOND  is not needed.
      (Time for SPPCO) = (1 + 18/N)*(Time for SPPFA) .

      On Entry

         AP      REAL (N*(N+1)/2)
                 the packed form of a symmetric matrix  A .  The
                 columns of the upper triangle are stored sequentially
                 in a one-dimensional array of length  N*(N+1)/2 .
                 See comments below for details.

         N       INTEGER
                 the order of the matrix  A .

      On Return

         AP      an upper triangular matrix  R , stored in packed
                 form, so that  A = TRANS(R)*R .

         INFO    INTEGER
                 = 0  for normal return.
                 = K  if the leading minor of order  K  is not
                      positive definite.


      Packed Storage

           The following program segment will pack the upper
           triangle of a symmetric matrix.

                 K = 0
                 DO 20 J = 1, N
                    DO 10 I = 1, J
                       K = K + 1
                       AP(K) = A(I,J)
              10    CONTINUE
              20 CONTINUE

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
```

```
***ROUTINES CALLED  SDOT
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SPPSL

```
      SUBROUTINE SPPSL (AP, N, B)
***BEGIN PROLOGUE  SPPSL
***PURPOSE  Solve the real symmetric positive definite system using
            the factors computed by SPPCO or SPPFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B
***TYPE      SINGLE PRECISION (SPPSL-S, DPPSL-D, CPPSL-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX, PACKED,
             POSITIVE DEFINITE, SOLVE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SPPSL solves the real symmetric positive definite system
      A * X = B
      using the factors computed by SPPCO or SPPFA.

      On Entry

         AP      REAL (N*(N+1)/2)
                 the output from SPPCO or SPPFA.

         N       INTEGER
                 the order of the matrix  A .

         B       REAL(N)
                 the right hand side vector.

      On Return

         B       the solution vector  X .

      Error Condition

         A division by zero will occur if the input factor contains
         a zero on the diagonal.  Technically, this indicates
         singularity, but it is usually caused by improper subroutine
         arguments.  It will not occur if the subroutines are called
         correctly and  INFO .EQ. 0 .

      To compute  INVERSE(A) * C  where  C  is a matrix
      with  P  columns
            CALL SPPCO(AP,N,RCOND,Z,INFO)
            IF (RCOND is too small .OR. INFO .NE. 0) GO TO ...
            DO 10 J = 1, P
               CALL SPPSL(AP,N,C(1,J))
         10 CONTINUE

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SDOT
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
```

```
        (WRB)
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

SPSORT

```
      SUBROUTINE SPSORT (X, N, IPERM, KFLAG, IER)
***BEGIN PROLOGUE  SPSORT
***PURPOSE  Return the permutation vector generated by sorting a given
            array and, optionally, rearrange the elements of the array.
            The array may be sorted in increasing or decreasing order.
            A slightly modified quicksort algorithm is used.
***LIBRARY   SLATEC
***CATEGORY  N6A1B, N6A2B
***TYPE      SINGLE PRECISION (SPSORT-S, DPSORT-D, IPSORT-I, HPSORT-H)
***KEYWORDS  NUMBER SORTING, PASSIVE SORTING, SINGLETON QUICKSORT, SORT
***AUTHOR  Jones, R. E., (SNLA)
           Rhoads, G. S., (NBS)
           Wisniewski, J. A., (SNLA)
***DESCRIPTION

   SPSORT returns the permutation vector IPERM generated by sorting
   the array X and, optionally, rearranges the values in X.  X may
   be sorted in increasing or decreasing order.  A slightly modified
   quicksort algorithm is used.

   IPERM is such that X(IPERM(I)) is the Ith value in the rearrangement
   of X.  IPERM may be applied to another array by calling IPPERM,
   SPPERM, DPPERM or HPPERM.

   The main difference between SPSORT and its active sorting equivalent
   SSORT is that the data are referenced indirectly rather than
   directly.  Therefore, SPSORT should require approximately twice as
   long to execute as SSORT.  However, SPSORT is more general.

   Description of Parameters
      X - input/output -- real array of values to be sorted.
          If ABS(KFLAG) = 2, then the values in X will be
          rearranged on output; otherwise, they are unchanged.
      N - input -- number of values in array X to be sorted.
      IPERM - output -- permutation array such that IPERM(I) is the
              index of the value in the original order of the
              X array that is in the Ith location in the sorted
              order.
      KFLAG - input -- control parameter:
            =  2  means return the permutation vector resulting from
                  sorting X in increasing order and sort X also.
            =  1  means return the permutation vector resulting from
                  sorting X in increasing order and do not sort X.
            = -1  means return the permutation vector resulting from
                  sorting X in decreasing order and do not sort X.
            = -2  means return the permutation vector resulting from
                  sorting X in decreasing order and sort X also.
      IER - output -- error indicator:
          =  0  if no error,
          =  1  if N is zero or negative,
          =  2  if KFLAG is not 2, 1, -1, or -2.
***REFERENCES  R. C. Singleton, Algorithm 347, An efficient algorithm
                 for sorting with minimal storage, Communications of
                 the ACM, 12, 3 (1969), pp. 185-187.
***ROUTINES CALLED  XERMSG
***REVISION HISTORY  (YYMMDD)
```

```
761101  DATE WRITTEN
761118  Modified by John A. Wisniewski to use the Singleton
        quicksort algorithm.
870423  Modified by Gregory S. Rhoads for passive sorting with the
        option for the rearrangement of the original data.
890620  Algorithm for rearranging the data vector corrected by R.
        Boisvert.
890622  Prologue upgraded to Version 4.0 style by D. Lozier.
891128  Error when KFLAG.LT.0 and N=1 corrected by R. Boisvert.
920507  Modified by M. McClain to revise prologue text.
920818  Declarations section rebuilt and code restructured to use
        IF-THEN-ELSE-ENDIF.  (SMR, WRB)
END PROLOGUE
```

SPTSL

```
      SUBROUTINE SPTSL (N, D, E, B)
***BEGIN PROLOGUE  SPTSL
***PURPOSE  Solve a positive definite tridiagonal linear system.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B2A
***TYPE      SINGLE PRECISION (SPTSL-S, DPTSL-D, CPTSL-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX, POSITIVE DEFINITE, SOLVE,
             TRIDIAGONAL
***AUTHOR  Dongarra, J., (ANL)
***DESCRIPTION

      SPTSL given a positive definite tridiagonal matrix and a right
      hand side will find the solution.

      On Entry

         N        INTEGER
                  is the order of the tridiagonal matrix.

         D        REAL(N)
                  is the diagonal of the tridiagonal matrix.
                  On output, D is destroyed.

         E        REAL(N)
                  is the offdiagonal of the tridiagonal matrix.
                  E(1) through E(N-1) should contain the
                  offdiagonal.

         B        REAL(N)
                  is the right hand side vector.

      On Return

         B        contains the solution.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                  Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890505  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SQRDC

```
      SUBROUTINE SQRDC (X, LDX, N, P, QRAUX, JPVT, WORK, JOB)
***BEGIN PROLOGUE  SQRDC
***PURPOSE  Use Householder transformations to compute the QR
            factorization of an N by P matrix.  Column pivoting is a
            users option.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D5
***TYPE      SINGLE PRECISION (SQRDC-S, DQRDC-D, CQRDC-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX, ORTHOGONAL TRIANGULAR,
             QR DECOMPOSITION
***AUTHOR  Stewart, G. W., (U. of Maryland)
***DESCRIPTION

      SQRDC uses Householder transformations to compute the QR
      factorization of an N by P matrix X.  Column pivoting
      based on the 2-norms of the reduced columns may be
      performed at the user's option.

      On Entry

         X       REAL(LDX,P), where LDX .GE. N.
                 X contains the matrix whose decomposition is to be
                 computed.

         LDX     INTEGER.
                 LDX is the leading dimension of the array X.

         N       INTEGER.
                 N is the number of rows of the matrix X.

         P       INTEGER.
                 P is the number of columns of the matrix X.

         JPVT    INTEGER(P).
                 JPVT contains integers that control the selection
                 of the pivot columns.  The K-th column X(K) of X
                 is placed in one of three classes according to the
                 value of JPVT(K).

                    If JPVT(K) .GT. 0, then X(K) is an initial
                                       column.

                    If JPVT(K) .EQ. 0, then X(K) is a free column.

                    If JPVT(K) .LT. 0, then X(K) is a final column.

                 Before the decomposition is computed, initial columns
                 are moved to the beginning of the array X and final
                 columns to the end.  Both initial and final columns
                 are frozen in place during the computation and only
                 free columns are moved.  At the K-th stage of the
                 reduction, if X(K) is occupied by a free column,
                 it is interchanged with the free column of largest
                 reduced norm.  JPVT is not referenced if
                 JOB .EQ. 0.
```

```
         WORK    REAL(P).
                 WORK is a work array.  WORK is not referenced if
                 JOB .EQ. 0.

         JOB     INTEGER.
                 JOB is an integer that initiates column pivoting.
                 If JOB .EQ. 0, no pivoting is done.
                 If JOB .NE. 0, pivoting is done.

      On Return

         X       X contains in its upper triangle the upper
                 triangular matrix R of the QR factorization.
                 Below its diagonal X contains information from
                 which the orthogonal part of the decomposition
                 can be recovered.  Note that if pivoting has
                 been requested, the decomposition is not that
                 of the original matrix X but that of X
                 with its columns permuted as described by JPVT.

         QRAUX   REAL(P).
                 QRAUX contains further information required to recover
                 the orthogonal part of the decomposition.

         JPVT    JPVT(K) contains the index of the column of the
                 original matrix that has been interchanged into
                 the K-th column, if pivoting was requested.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SDOT, SNRM2, SSCAL, SSWAP
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SQRSL

```
      SUBROUTINE SQRSL (X, LDX, N, K, QRAUX, Y, QY, QTY, B, RSD, XB,
     +   JOB, INFO)
***BEGIN PROLOGUE  SQRSL
***PURPOSE  Apply the output of SQRDC to compute coordinate transfor-
            mations, projections, and least squares solutions.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D9, D2A1
***TYPE      SINGLE PRECISION (SQRSL-S, DQRSL-D, CQRSL-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX, ORTHOGONAL TRIANGULAR,
             SOLVE
***AUTHOR  Stewart, G. W., (U. of Maryland)
***DESCRIPTION

      SQRSL applies the output of SQRDC to compute coordinate
      transformations, projections, and least squares solutions.
      For K .LE. MIN(N,P), let XK be the matrix

             XK = (X(JPVT(1)),X(JPVT(2)), ... ,X(JPVT(K)))

      formed from columns JPVT(1), ... ,JPVT(K) of the original
      N x P matrix X that was input to SQRDC (if no pivoting was
      done, XK consists of the first K columns of X in their
      original order).  SQRDC produces a factored orthogonal matrix Q
      and an upper triangular matrix R such that

               XK = Q * (R)
                        (0)

      This information is contained in coded form in the arrays
      X and QRAUX.

      On Entry

         X      REAL(LDX,P)
                X contains the output of SQRDC.

         LDX    INTEGER
                LDX is the leading dimension of the array X.

         N      INTEGER
                N is the number of rows of the matrix XK.  It must
                have the same value as N in SQRDC.

         K      INTEGER
                K is the number of columns of the matrix XK.  K
                must not be greater than MIN(N,P), where P is the
                same as in the calling sequence to SQRDC.

         QRAUX  REAL(P)
                QRAUX contains the auxiliary output from SQRDC.

         Y      REAL(N)
                Y contains an N-vector that is to be manipulated
                by SQRSL.

         JOB    INTEGER
```

JOB specifies what is to be computed. JOB has the decimal expansion ABCDE, with the following meaning.

```
If A .NE. 0, compute QY.
If B,C,D, or E .NE. 0, compute QTY.
If C .NE. 0, compute B.
If D .NE. 0, compute RSD.
If E .NE. 0, compute XB.
```

Note that a request to compute B, RSD, or XB automatically triggers the computation of QTY, for which an array must be provided in the calling sequence.

On Return

QY REAL(N).
QY contains Q*Y, if its computation has been requested.

QTY REAL(N).
QTY contains TRANS(Q)*Y, if its computation has been requested. Here TRANS(Q) is the transpose of the matrix Q.

B REAL(K)
B contains the solution of the least squares problem

```
minimize norm2(Y - XK*B),
```

if its computation has been requested. (Note that if pivoting was requested in SQRDC, the J-th component of B will be associated with column JPVT(J) of the original matrix X that was input into SQRDC.)

RSD REAL(N).
RSD contains the least squares residual Y - XK*B, if its computation has been requested. RSD is also the orthogonal projection of Y onto the orthogonal complement of the column space of XK.

XB REAL(N).
XB contains the least squares approximation XK*B, if its computation has been requested. XB is also the orthogonal projection of Y onto the column space of X.

INFO INTEGER.
INFO is zero unless the computation of B has been requested and R is exactly singular. In this case, INFO is the index of the first zero diagonal element of R and B is left unaltered.

The parameters QY, QTY, B, RSD, and XB are not referenced if their computation is not requested and in this case can be replaced by dummy variables in the calling program. To save storage, the user may in some cases use the same array for different parameters in the calling sequence. A frequently occurring example is when one wishes to compute

```
     any of B, RSD, or XB and does not need Y or QTY.  In this
     case one may identify Y, QTY, and one of B, RSD, or XB, while
     providing separate arrays for anything else that is to be
     computed.  Thus the calling sequence

          CALL SQRSL(X,LDX,N,K,QRAUX,Y,DUM,Y,B,Y,DUM,110,INFO)

     will result in the computation of B and RSD, with RSD
     overwriting Y.  More generally, each item in the following
     list contains groups of permissible identifications for
     a single calling sequence.

          1. (Y,QTY,B) (RSD) (XB) (QY)

          2. (Y,QTY,RSD) (B) (XB) (QY)

          3. (Y,QTY,XB) (B) (RSD) (QY)

          4. (Y,QY) (QTY,B) (RSD) (XB)

          5. (Y,QY) (QTY,RSD) (B) (XB)

          6. (Y,QY) (QTY,XB) (B) (RSD)

     In any group the value returned in the array allocated to
     the group corresponds to the last member of the group.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SCOPY, SDOT
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SROT

```
      SUBROUTINE SROT (N, SX, INCX, SY, INCY, SC, SS)
***BEGIN PROLOGUE  SROT
***PURPOSE  Apply a plane Givens rotation.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A8
***TYPE      SINGLE PRECISION (SROT-S, DROT-D, CSROT-C)
***KEYWORDS  BLAS, GIVENS ROTATION, GIVENS TRANSFORMATION,
             LINEAR ALGEBRA, PLANE ROTATION, VECTOR
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
           Kincaid, D. R., (U. of Texas)
           Krogh, F. T., (JPL)
***DESCRIPTION

                B L A S  Subprogram
    Description of Parameters

     --Input--
        N  number of elements in input vector(s)
       SX  single precision vector with N elements
     INCX  storage spacing between elements of SX
       SY  single precision vector with N elements
     INCY  storage spacing between elements of SY
       SC  element of rotation matrix
       SS  element of rotation matrix

     --Output--
       SX  rotated vector SX (unchanged if N .LE. 0)
       SY  rotated vector SY (unchanged if N .LE. 0)

     Multiply the 2 x 2 matrix  ( SC SS) times the 2 x N matrix (SX**T)
                                (-SS SC)                        (SY**T)
     where **T indicates transpose.  The elements of SX are in
     SX(LX+I*INCX), I = 0 to N-1, where LX = 1 if INCX .GE. 0, else
     LX = 1+(1-N)*INCX, and similarly for SY using LY and INCY.

***REFERENCES  C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T.
                 Krogh, Basic linear algebra subprograms for Fortran
                 usage, Algorithm No. 539, Transactions on Mathematical
                 Software 5, 3 (September 1979), pp. 308-323.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   791001  DATE WRITTEN
   861211  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920310  Corrected definition of LX in DESCRIPTION.  (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SROTG

```
      SUBROUTINE SROTG (SA, SB, SC, SS)
***BEGIN PROLOGUE  SROTG
***PURPOSE  Construct a plane Givens rotation.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B10
***TYPE      SINGLE PRECISION (SROTG-S, DROTG-D, CROTG-C)
***KEYWORDS  BLAS, GIVENS ROTATION, GIVENS TRANSFORMATION,
             LINEAR ALGEBRA, VECTOR
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
           Kincaid, D. R., (U. of Texas)
           Krogh, F. T., (JPL)
***DESCRIPTION

                B L A S  Subprogram
    Description of Parameters

      --Input--
        SA  single precision scalar
        SB  single precision scalar

      --Output--
        SA  single precision result R
        SB  single precision result Z
        SC  single precision result
        SS  single precision result

      Construct the Givens transformation

          ( SC  SS )
      G = (        ) ,    SC**2 + SS**2 = 1 ,
          (-SS  SC )

      which zeros the second entry of the 2-vector  (SA,SB)**T.

      The quantity R = (+/-)SQRT(SA**2 + SB**2) overwrites SA in
      storage.  The value of SB is overwritten by a value Z which
      allows SC and SS to be recovered by the following algorithm:

            If Z=1  set  SC=0.0  and  SS=1.0
            If ABS(Z) .LT. 1  set  SC=SQRT(1-Z**2)  and  SS=Z
            If ABS(Z) .GT. 1  set  SC=1/Z  and  SS=SQRT(1-SC**2)

      Normally, the subprogram SROT(N,SX,INCX,SY,INCY,SC,SS) will
      next be called to apply the transformation to a 2 by N matrix.

***REFERENCES  C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T.
                 Krogh, Basic linear algebra subprograms for Fortran
                 usage, Algorithm No. 539, Transactions on Mathematical
                 Software 5, 3 (September 1979), pp. 308-323.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   791001  DATE WRITTEN
   861211  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
```

```
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

SROTM

```
      SUBROUTINE SROTM (N, SX, INCX, SY, INCY, SPARAM)
***BEGIN PROLOGUE  SROTM
***PURPOSE  Apply a modified Givens transformation.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A8
***TYPE      SINGLE PRECISION (SROTM-S, DROTM-D)
***KEYWORDS  BLAS, LINEAR ALGEBRA, MODIFIED GIVENS ROTATION, VECTOR
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
           Kincaid, D. R., (U. of Texas)
           Krogh, F. T., (JPL)
***DESCRIPTION

                B L A S  Subprogram
    Description of Parameters

     --Input--
        N  number of elements in input vector(s)
       SX  single precision vector with N elements
     INCX  storage spacing between elements of SX
       SY  single precision vector with N elements
     INCY  storage spacing between elements of SY
   SPARAM  5-element vector. SPARAM(1) is SFLAG described below.
           Locations 2-5 of SPARAM contain elements of the
           transformation matrix H described below.

     --Output--
       SX  rotated vector (unchanged if N .LE. 0)
       SY  rotated vector (unchanged if N .LE. 0)

     Apply the modified Givens transformation, H, to the 2 by N matrix
     (SX**T)
     (SY**T) , where **T indicates transpose.  The elements of SX are
     in SX(LX+I*INCX), I = 0 to N-1, where LX = 1 if INCX .GE. 0, else
     LX = 1+(1-N)*INCX, and similarly for SY using LY and INCY.

     With SPARAM(1)=SFLAG, H has one of the following forms:

     SFLAG=-1.E0     SFLAG=0.E0        SFLAG=1.E0     SFLAG=-2.E0

       (SH11  SH12)    (1.E0  SH12)    (SH11  1.E0)    (1.E0  0.E0)
     H=(          )    (          )    (          )    (          )
       (SH21  SH22),   (SH21  1.E0),   (-1.E0 SH22),   (0.E0  1.E0).

     See SROTMG for a description of data storage in SPARAM.

***REFERENCES  C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T.
                 Krogh, Basic linear algebra subprograms for Fortran
                 usage, Algorithm No. 539, Transactions on Mathematical
                 Software 5, 3 (September 1979), pp. 308-323.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   791001  DATE WRITTEN
   861211  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920310  Corrected definition of LX in DESCRIPTION.  (WRB)
```

```
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

SROTMG

```
      SUBROUTINE SROTMG (SD1, SD2, SX1, SY1, SPARAM)
***BEGIN PROLOGUE  SROTMG
***PURPOSE  Construct a modified Givens transformation.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B10
***TYPE      SINGLE PRECISION (SROTMG-S, DROTMG-D)
***KEYWORDS  BLAS, LINEAR ALGEBRA, MODIFIED GIVENS ROTATION, VECTOR
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
           Kincaid, D. R., (U. of Texas)
           Krogh, F. T., (JPL)
***DESCRIPTION

                B L A S  Subprogram
    Description of Parameters

      --Input--
       SD1  single precision scalar
       SD2  single precision scalar
       SX1  single precision scalar
       SY2  single precision scalar
   SPARAM  S.P. 5-vector. SPARAM(1)=SFLAG defined below.
           Locations 2-5 contain the rotation matrix.

      --Output--
       SD1  changed to represent the effect of the transformation
       SD2  changed to represent the effect of the transformation
       SX1  changed to represent the effect of the transformation
       SY2  unchanged

      Construct the modified Givens transformation matrix H which zeros
      the second component of the 2-vector  (SQRT(SD1)*SX1,SQRT(SD2)*
      SY2)**T.
      With SPARAM(1)=SFLAG, H has one of the following forms:

      SFLAG=-1.E0     SFLAG=0.E0        SFLAG=1.E0     SFLAG=-2.E0

        (SH11  SH12)    (1.E0  SH12)    (SH11  1.E0)    (1.E0  0.E0)
      H=(          )    (          )    (          )    (          )
        (SH21  SH22),   (SH21  1.E0),   (-1.E0 SH22),   (0.E0  1.E0).

      Locations 2-5 of SPARAM contain SH11, SH21, SH12, and SH22,
      respectively.  (Values of 1.E0, -1.E0, or 0.E0 implied by the
      value of SPARAM(1) are not stored in SPARAM.)

***REFERENCES  C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T.
                 Krogh, Basic linear algebra subprograms for Fortran
                 usage, Algorithm No. 539, Transactions on Mathematical
                 Software 5, 3 (September 1979), pp. 308-323.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   780301  DATE WRITTEN
   861211  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920316  Prologue corrected.  (WRB)
```

```
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

SS2LT

```
      SUBROUTINE SS2LT (N, NELT, IA, JA, A, ISYM, NEL, IEL, JEL, EL)
***BEGIN PROLOGUE  SS2LT
***PURPOSE  Lower Triangle Preconditioner SLAP Set Up.
            Routine to store the lower triangle of a matrix stored
            in the SLAP Column format.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2E
***TYPE      SINGLE PRECISION (SS2LT-S, DS2LT-D)
***KEYWORDS  LINEAR SYSTEM, LOWER TRIANGLE, SLAP SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM
     INTEGER NEL, IEL(NEL), JEL(NEL)
     REAL    A(NELT), EL(NEL)

     CALL SS2LT( N, NELT, IA, JA, A, ISYM, NEL, IEL, JEL, EL )

 *Arguments:
 N       :IN       Integer
          Order of the Matrix.
 NELT    :IN       Integer.
          Number of non-zeros stored in A.
 IA      :IN       Integer IA(NELT).
 JA      :IN       Integer JA(NELT).
 A       :IN       Real A(NELT).
          These arrays should hold the matrix A in the SLAP Column
          format.  See "Description", below.
 ISYM    :IN       Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the lower
          triangle of the matrix is stored.
 NEL     :OUT      Integer.
          Number of non-zeros in the lower triangle of A.   Also
          corresponds to the length of the IEL, JEL, EL arrays.
 IEL     :OUT      Integer IEL(NEL).
 JEL     :OUT      Integer JEL(NEL).
 EL      :OUT      Real     EL(NEL).
          IEL, JEL, EL contain the lower triangle of the A matrix
          stored in SLAP Column format.  See "Description", below,
          for more details bout the SLAP Column format.

 *Description
       ================== S L A P Column format ==================
       This routine  requires that  the matrix A  be stored in  the
       SLAP Column format.  In this format the non-zeros are stored
       counting down columns (except for  the diagonal entry, which
       must appear first in each  "column")  and are stored  in the
       real array A.  In other words, for each column in the matrix
```

put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

```
***REFERENCES  (NONE)
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920511  Added complete declaration section.  (WRB)
   930701  Updated CATEGORY section.  (FNF, WRB)
   END PROLOGUE
```

SS2Y

```
      SUBROUTINE SS2Y (N, NELT, IA, JA, A, ISYM)
***BEGIN PROLOGUE  SS2Y
***PURPOSE  SLAP Triad to SLAP Column Format Converter.
            Routine to convert from the SLAP Triad to SLAP Column
            format.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D1B9
***TYPE      SINGLE PRECISION (SS2Y-S, DS2Y-D)
***KEYWORDS  LINEAR SYSTEM, SLAP SPARSE
***AUTHOR  Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM
     REAL    A(NELT)

     CALL SS2Y( N, NELT, IA, JA, A, ISYM )

 *Arguments:
 N       :IN       Integer
          Order of the Matrix.
 NELT    :IN       Integer.
          Number of non-zeros stored in A.
 IA      :INOUT    Integer IA(NELT).
 JA      :INOUT    Integer JA(NELT).
 A       :INOUT    Real A(NELT).
          These arrays should hold the matrix A in either the SLAP
          Triad format or the SLAP Column format.  See "Description",
          below.  If the SLAP Triad format is used, this format is
          translated to the SLAP Column format by this routine.
 ISYM    :IN       Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the lower
          triangle of the matrix is stored.

 *Description:
       The Sparse Linear Algebra Package (SLAP) utilizes two matrix
       data structures: 1) the  SLAP Triad  format or  2)  the SLAP
       Column format.  The user can hand this routine either of the
       of these data structures.  If the SLAP Triad format is give
       as input then this routine transforms it into SLAP Column
       format.  The way this routine tells which format is given as
       input is to look at JA(N+1).  If JA(N+1) = NELT+1 then we
       have the SLAP Column format.  If that equality does not hold
       then it is assumed that the IA, JA, A arrays contain the
       SLAP Triad format.

       ================== S L A P Triad format ==================
       This routine requires that the  matrix A be   stored in  the
       SLAP  Triad format.  In  this format only the non-zeros  are
       stored.  They may appear in  *ANY* order.  The user supplies
```

three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

```
***REFERENCES  (NONE)
***ROUTINES CALLED  QS2I1R
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
```

```
        standard.  (FNF)
890929  Numerous changes to reduce SP/DP differences.  (FNF)
910411  Prologue converted to Version 4.0 format.  (BAB)
910502  Corrected C***FIRST EXECUTABLE STATEMENT line.  (FNF)
920511  Added complete declaration section.  (WRB)
930701  Updated CATEGORY section.  (FNF, WRB)
END PROLOGUE
```

SSBMV

```
      SUBROUTINE SSBMV (UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y,
     $   INCY)
***BEGIN PROLOGUE  SSBMV
***PURPOSE  Multiply a real vector by a real symmetric band matrix.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (SSBMV-S, DSBMV-D, CSBMV-C)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  SSBMV  performs the matrix-vector  operation

     y := alpha*A*x + beta*y,

  where alpha and beta are scalars, x and y are n element vectors 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 supplied 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  - REAL             .
           On entry, ALPHA specifies the scalar alpha.
           Unchanged on exit.

  A      - REAL             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, 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 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, 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 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 - REAL 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 elements of X. INCX must not be zero.
Unchanged on exit.

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

Y - REAL 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 elements of
           Y. INCY must not be zero.
           Unchanged on exit.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

SSCAL

```
      SUBROUTINE SSCAL (N, SA, SX, INCX)
***BEGIN PROLOGUE  SSCAL
***PURPOSE  Multiply a vector by a constant.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A6
***TYPE      SINGLE PRECISION (SSCAL-S, DSCAL-D, CSCAL-C)
***KEYWORDS  BLAS, LINEAR ALGEBRA, SCALE, VECTOR
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
           Kincaid, D. R., (U. of Texas)
           Krogh, F. T., (JPL)
***DESCRIPTION

                B L A S  Subprogram
    Description of Parameters

     --Input--
        N  number of elements in input vector(s)
       SA  single precision scale factor
       SX  single precision vector with N elements
     INCX  storage spacing between elements of SX

     --Output--
       SX  single precision result (unchanged if N .LE. 0)

     Replace single precision SX by single precision SA*SX.
     For I = 0 to N-1, replace SX(IX+I*INCX) with  SA * SX(IX+I*INCX),
     where IX = 1 if INCX .GE. 0, else IX = 1+(1-N)*INCX.

***REFERENCES  C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T.
                 Krogh, Basic linear algebra subprograms for Fortran
                 usage, Algorithm No. 539, Transactions on Mathematical
                 Software 5, 3 (September 1979), pp. 308-323.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   791001  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900821  Modified to correct problem with a negative increment.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SSD2S

```
      SUBROUTINE SSD2S (N, NELT, IA, JA, A, ISYM, DINV)
***BEGIN PROLOGUE  SSD2S
***PURPOSE  Diagonal Scaling Preconditioner SLAP Normal Eqns Set Up.
            Routine to compute the inverse of the diagonal of the
            matrix A*A', where A is stored in SLAP-Column format.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2E
***TYPE      SINGLE PRECISION (SSD2S-S, DSD2S-D)
***KEYWORDS  DIAGONAL, SLAP SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM
     REAL    A(NELT), DINV(N)

     CALL SSD2S( N, NELT, IA, JA, A, ISYM, DINV )

 *Arguments:
 N       :IN       Integer
          Order of the Matrix.
 NELT    :IN       Integer.
          Number of elements in arrays IA, JA, and A.
 IA      :IN       Integer IA(NELT).
 JA      :IN       Integer JA(NELT).
 A       :IN       Real A(NELT).
          These arrays should hold the matrix A in the SLAP Column
          format.  See "Description", below.
 ISYM    :IN       Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the upper
          or lower triangle of the matrix is stored.
 DINV    :OUT      Real DINV(N).
          Upon return this array holds 1./DIAG(A*A').

 *Description
       ================== S L A P Column format ==================
       This routine  requires that  the matrix A  be stored in  the
       SLAP Column format.  In this format the non-zeros are stored
       counting down columns (except for  the diagonal entry, which
       must appear first in each  "column")  and are stored  in the
       real array A.  In other words, for each column in the matrix
       put the diagonal entry in A.  Then put in the other non-zero
       elements going down   the  column (except  the diagonal)  in
       order.  The IA array holds the row  index for each non-zero.
       The JA array holds the offsets into the IA, A arrays for the
       beginning of   each    column.    That  is,    IA(JA(ICOL)),
       A(JA(ICOL)) points to the beginning of the ICOL-th column in
       IA and  A.  IA(JA(ICOL+1)-1),  A(JA(ICOL+1)-1) points to the
       end  of   the ICOL-th  column.  Note   that  we  always have
```

JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

With the SLAP format all of the "inner loops" of this routine should vectorize on machines with hardware support for vector gather/scatter operations. Your compiler may require a compiler directive to convince it that there are no implicit vector dependencies. Compiler directives for the Alliant FX/Fortran and CRI CFT/CFT77 compilers are supplied with the standard SLAP distribution.

*Cautions:

This routine assumes that the diagonal of A is all non-zero and that the operation DINV = 1.0/DIAG(A*A') will not under-flow or overflow. This is done so that the loop vectorizes. Matrices with zero or near zero or very large entries will have numerical difficulties and must be fixed before this routine is called.

***SEE ALSO SSDCGN
***REFERENCES (NONE)
***ROUTINES CALLED (NONE)
***REVISION HISTORY (YYMMDD)
871119 DATE WRITTEN
881213 Previous REVISION DATE
890915 Made changes requested at July 1989 CML Meeting. (MKS)
890922 Numerous changes to prologue to make closer to SLATEC standard. (FNF)
890929 Numerous changes to reduce SP/DP differences. (FNF)
910411 Prologue converted to Version 4.0 format. (BAB)
920511 Added complete declaration section. (WRB)
921113 Corrected C***CATEGORY line. (FNF)
930701 Updated CATEGORY section. (FNF, WRB)
END PROLOGUE

SSDBCG

```
      SUBROUTINE SSDBCG(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
     +   ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)
***BEGIN PROLOGUE  SSDBCG
***PURPOSE  Diagonally Scaled BiConjugate Gradient Sparse Ax=b Solver.
            Routine to solve a linear system  Ax = b  using the
            BiConjugate Gradient method with diagonal scaling.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SSDBCG-S, DSDBCG-D)
***KEYWORDS  ITERATIVE PRECONDITION, NON-SYMMETRIC LINEAR SYSTEM, SLAP,
             SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
     INTEGER ITER, IERR, IUNIT, LENW, IWORK(10), LENIW
     REAL B(N), X(N), A(NELT), TOL, ERR, RWORK(8*N)

     CALL SSDBCG(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
    $     ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW )

 *Arguments:
 N       :IN       Integer
          Order of the Matrix.
 B       :IN       Real B(N).
          Right-hand side vector.
 X       :INOUT    Real X(N).
          On input X is your initial guess for solution vector.
          On output X is the final approximate solution.
 NELT    :IN       Integer.
          Number of Non-Zeros stored in A.
 IA      :INOUT    Integer IA(NELT).
 JA      :INOUT    Integer JA(NELT).
 A       :INOUT    Real A(NELT).
          These arrays should hold the matrix A in either the SLAP
          Triad format or the SLAP Column format.  See "Description",
          below.  If the SLAP Triad format is chosen it is changed
          internally to the SLAP Column format.
 ISYM    :IN       Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the upper
          or lower triangle of the matrix is stored.
 ITOL    :IN       Integer.
          Flag to indicate type of convergence criterion.
          If ITOL=1, iteration stops when the 2-norm of the residual
          divided by the 2-norm of the right-hand side is less than TOL.
          If ITOL=2, iteration stops when the 2-norm of M-inv times the
          residual divided by the 2-norm of M-inv times the right hand
          side is less than TOL, where M-inv is the inverse of the
```

```
         diagonal of A.
         ITOL=11 is often useful for checking and comparing different
         routines.  For this case, the user must supply the "exact"
         solution or a very accurate approximation (one with an error
         much less than TOL) through a common block,
             COMMON /SSLBLK/ SOLN( )
         If ITOL=11, iteration stops when the 2-norm of the difference
         between the iterative approximation and the user-supplied
         solution divided by the 2-norm of the user-supplied solution
         is less than TOL.  Note that this requires the user to set up
         the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine.
         The routine with this declaration should be loaded before the
         stop test so that the correct length is used by the loader.
         This procedure is not standard Fortran and may not work
         correctly on your system (although it has worked on every
         system the authors have tried).  If ITOL is not 11 then this
         common block is indeed standard Fortran.
TOL     :INOUT    Real.
         Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX   :IN       Integer.
         Maximum number of iterations.
ITER    :OUT      Integer.
         Number of iterations required to reach convergence, or
         ITMAX+1 if convergence criterion could not be achieved in
         ITMAX iterations.
ERR     :OUT      Real.
         Error estimate of error in final approximate solution, as
         defined by ITOL.
IERR    :OUT      Integer.
         Return error flag.
           IERR = 0 => All went well.
           IERR = 1 => Insufficient space allocated for WORK or IWORK.
           IERR = 2 => Method failed to converge in ITMAX steps.
           IERR = 3 => Error in user input.
                       Check input values of N, ITOL.
           IERR = 4 => User error tolerance set too tight.
                       Reset to 500*R1MACH(3).  Iteration proceeded.
           IERR = 5 => Preconditioning matrix, M, is not positive
                       definite.  (r,z) < 0.
           IERR = 6 => Matrix A is not positive definite.  (p,Ap) < 0.
IUNIT   :IN       Integer.
         Unit number on which to write the error at each iteration,
         if this is desired for monitoring convergence.  If unit
         number is 0, no writing will occur.
RWORK   :WORK     Real RWORK(LENW).
         Real array used for workspace.
LENW    :IN       Integer.
         Length of the real workspace, RWORK.  LENW >= 8*N.
IWORK   :WORK     Integer IWORK(LENIW).
         Used to hold pointers into the RWORK array.
LENIW   :IN       Integer.
         Length of the integer workspace, IWORK.  LENIW >= 10.
         Upon return the following locations of IWORK hold information
         which may be of use to the user:
         IWORK(9)  Amount of Integer workspace actually used.
         IWORK(10) Amount of Real workspace actually used.

*Description:
      This  routine performs  preconditioned  BiConjugate gradient
      method on the Non-Symmetric positive definite  linear system
```

Ax=b. The preconditioner is M = DIAG(A), the diagonal of the matrix A. This is the simplest of preconditioners and vectorizes very well.

The Sparse Linear Algebra Package (SLAP) utilizes two matrix data structures: 1) the SLAP Triad format or 2) the SLAP Column format. The user can hand this routine either of the of these data structures and SLAP will figure out which on is being used and act accordingly.

========== S L A P Triad format ==========

This routine requires that the matrix A be stored in the SLAP Triad format. In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
          5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                                1  2  3    4  5    6  7    8    9 10 11
       |11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
       |21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
       | 0  0 33  0 35|  JA:  1  4  6    8  9   12
       | 0  0  0 44  0|
       |51  0 53  0 55|
```

*Side Effects:
The SLAP Triad format (IA, JA, A) is modified internally to be the SLAP Column format. See above.

*Cautions:
This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO SBCG, SLUBCG
***REFERENCES (NONE)
***ROUTINES CALLED SBCG, SCHKW, SS2Y, SSDI, SSDS, SSMTV, SSMV
***REVISION HISTORY (YYMMDD)
871119 DATE WRITTEN
881213 Previous REVISION DATE
890915 Made changes requested at July 1989 CML Meeting. (MKS)
890921 Removed TeX from comments. (FNF)
890922 Numerous changes to prologue to make closer to SLATEC standard. (FNF)
890929 Numerous changes to reduce SP/DP differences. (FNF)
910411 Prologue converted to Version 4.0 format. (BAB)
920407 COMMON BLOCK renamed SSLBLK. (WRB)
920511 Added complete declaration section. (WRB)
921113 Corrected C***CATEGORY line. (FNF)
END PROLOGUE

SSDCG

```
    +   ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)
***BEGIN PROLOGUE  SSDCG
***PURPOSE  Diagonally Scaled Conjugate Gradient Sparse Ax=b Solver.
            Routine to solve a symmetric positive definite linear
            system  Ax = b  using the Preconditioned Conjugate
            Gradient method.  The preconditioner is diagonal scaling.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2B4
***TYPE      SINGLE PRECISION (SSDCG-S, DSDCG-D)
***KEYWORDS  ITERATIVE PRECONDITION, SLAP, SPARSE,
             SYMMETRIC LINEAR SYSTEM
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
     INTEGER ITER, IERR, IUNIT, LENW, IWORK(10), LENIW
     REAL B(N), X(N), A(NELT), TOL, ERR, RWORK(5*N)

     CALL SSDCG(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
    $     ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW )

 *Arguments:
 N      :IN       Integer.
         Order of the Matrix.
 B      :IN       Real B(N).
         Right-hand side vector.
 X      :INOUT    Real X(N).
         On input X is your initial guess for solution vector.
         On output X is the final approximate solution.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :INOUT    Integer IA(NELT).
 JA     :INOUT    Integer JA(NELT).
 A      :INOUT    Real A(NELT).
         These arrays should hold the matrix A in either the SLAP
         Triad format or the SLAP Column format.  See "Description",
         below.  If the SLAP Triad format is chosen it is changed
         internally to the SLAP Column format.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 ITOL   :IN       Integer.
         Flag to indicate type of convergence criterion.
         If ITOL=1, iteration stops when the 2-norm of the residual
         divided by the 2-norm of the right-hand side is less than TOL.
         If ITOL=2, iteration stops when the 2-norm of M-inv times the
         residual divided by the 2-norm of M-inv times the right hand
         side is less than TOL, where M-inv is the inverse of the
```

```
         diagonal of A.
         ITOL=11 is often useful for checking and comparing different
         routines.  For this case, the user must supply the "exact"
         solution or a very accurate approximation (one with an error
         much less than TOL) through a common block,
             COMMON /SSLBLK/ SOLN( )
         If ITOL=11, iteration stops when the 2-norm of the difference
         between the iterative approximation and the user-supplied
         solution divided by the 2-norm of the user-supplied solution
         is less than TOL.  Note that this requires the user to set up
         the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine.
         The routine with this declaration should be loaded before the
         stop test so that the correct length is used by the loader.
         This procedure is not standard Fortran and may not work
         correctly on your system (although it has worked on every
         system the authors have tried).  If ITOL is not 11 then this
         common block is indeed standard Fortran.
TOL     :INOUT    Real.
         Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX   :IN       Integer.
         Maximum number of iterations.
ITER    :OUT      Integer.
         Number of iterations required to reach convergence, or
         ITMAX+1 if convergence criterion could not be achieved in
         ITMAX iterations.
ERR     :OUT      Real.
         Error estimate of error in final approximate solution, as
         defined by ITOL.
IERR    :OUT      Integer.
         Return error flag.
           IERR = 0 => All went well.
           IERR = 1 => Insufficient space allocated for WORK or IWORK.
           IERR = 2 => Method failed to converge in ITMAX steps.
           IERR = 3 => Error in user input.
                       Check input values of N, ITOL.
           IERR = 4 => User error tolerance set too tight.
                       Reset to 500*R1MACH(3).  Iteration proceeded.
           IERR = 5 => Preconditioning matrix, M, is not positive
                       definite.  (r,z) < 0.
           IERR = 6 => Matrix A is not positive definite.  (p,Ap) < 0.
IUNIT   :IN       Integer.
         Unit number on which to write the error at each iteration,
         if this is desired for monitoring convergence.  If unit
         number is 0, no writing will occur.
RWORK   :WORK     Real RWORK(LENW).
         Real array used for workspace.
LENW    :IN       Integer.
         Length of the real workspace, RWORK.  LENW >= 5*N.
IWORK   :WORK     Integer IWORK(LENIW).
         Used to hold pointers into the real workspace, RWORK.
         Upon return the following locations of IWORK hold information
         which may be of use to the user:
         IWORK(9)  Amount of Integer workspace actually used.
         IWORK(10) Amount of Real workspace actually used.
LENIW   :IN       Integer.
         Length of the integer workspace, IWORK.  LENIW >= 10.

*Description:
      This  routine   performs preconditioned conjugate   gradient
      method on  the  symmetric positive definite   linear  system
```

Ax=b. The preconditioner is M = DIAG(A), the diagonal of the matrix A. This is the simplest of preconditioners and vectorizes very well. This routine is simply a driver for the SCG routine. It calls the SSDS routine to set up the preconditioning and then calls SCG with the appropriate MATVEC and MSOLVE routines.

The Sparse Linear Algebra Package (SLAP) utilizes two matrix data structures: 1) the SLAP Triad format or 2) the SLAP Column format. The user can hand this routine either of the of these data structures and SLAP will figure out which on is being used and act accordingly.

========== S L A P Triad format ==========

This routine requires that the matrix A be stored in the SLAP Triad format. In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a

5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Side Effects:
The SLAP Triad format (IA, JA, A) is modified internally to be the SLAP Column format. See above.

*Cautions:
This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO SCG, SSICCG

***REFERENCES

1. Louis Hageman and David Young, Applied Iterative Methods, Academic Press, New York, 1981.
2. Concus, Golub and O'Leary, A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations, in Sparse Matrix Computations, Bunch and Rose, Eds., Academic Press, New York, 1979.

***ROUTINES CALLED SCG, SCHKW, SS2Y, SSDI, SSDS, SSMV

***REVISION HISTORY (YYMMDD)

```
871119  DATE WRITTEN
881213  Previous REVISION DATE
890915  Made changes requested at July 1989 CML Meeting.  (MKS)
890921  Removed TeX from comments.  (FNF)
890922  Numerous changes to prologue to make closer to SLATEC
        standard.  (FNF)
890929  Numerous changes to reduce SP/DP differences.  (FNF)
910411  Prologue converted to Version 4.0 format.  (BAB)
920407  COMMON BLOCK renamed SSLBLK.  (WRB)
920511  Added complete declaration section.  (WRB)
920929  Corrected format of references.  (FNF)
END PROLOGUE
```

SSDCGN

```
      SUBROUTINE SSDCGN(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
     +   ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)
***BEGIN PROLOGUE  SSDCGN
***PURPOSE  Diagonally Scaled CG Sparse Ax=b Solver for Normal Eqn's.
            Routine to solve a general linear system  Ax = b  using
            diagonal scaling with the Conjugate Gradient method
            applied to the the normal equations, viz.,  AA'y = b,
            where  x = A'y.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SSDCGN-S, DSDCGN-D)
***KEYWORDS  ITERATIVE PRECONDITION, NON-SYMMETRIC LINEAR SYSTEM SOLVE,
             SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
     INTEGER ITER, IERR, IUNIT, LENW, IWORK(10), LENIW
     REAL B(N), X(N), A(NELT), TOL, ERR, RWORK(8*N)

     CALL SSDCGN(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
    $     ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)

 *Arguments:
 N       :IN       Integer
          Order of the Matrix.
 B       :IN       Real B(N).
          Right-hand side vector.
 X       :INOUT    Real X(N).
          On input X is your initial guess for solution vector.
          On output X is the final approximate solution.
 NELT    :IN       Integer.
          Number of Non-Zeros stored in A.
 IA      :INOUT    Integer IA(NELT).
 JA      :INOUT    Integer JA(NELT).
 A       :INOUT    Real A(NELT).
          These arrays should hold the matrix A in either the SLAP
          Triad format or the SLAP Column format.  See "Description",
          below.  If the SLAP Triad format is chosen it is changed
          internally to the SLAP Column format.
 ISYM    :IN       Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the upper
          or lower triangle of the matrix is stored.
 ITOL    :IN       Integer.
          Flag to indicate type of convergence criterion.
          If ITOL=1, iteration stops when the 2-norm of the residual
          divided by the 2-norm of the right-hand side is less than TOL.
          If ITOL=2, iteration stops when the 2-norm of M-inv times the
```

```
        residual divided by the 2-norm of M-inv times the right hand
        side is less than TOL, where M-inv is the inverse of the
        diagonal of A.
        ITOL=11 is often useful for checking and comparing different
        routines.  For this case, the user must supply the "exact"
        solution or a very accurate approximation (one with an error
        much less than TOL) through a common block,
            COMMON /SSLBLK/ SOLN( )
        If ITOL=11, iteration stops when the 2-norm of the difference
        between the iterative approximation and the user-supplied
        solution divided by the 2-norm of the user-supplied solution
        is less than TOL.  Note that this requires the user to set up
        the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine.
        The routine with this declaration should be loaded before the
        stop test so that the correct length is used by the loader.
        This procedure is not standard Fortran and may not work
        correctly on your system (although it has worked on every
        system the authors have tried).  If ITOL is not 11 then this
        common block is indeed standard Fortran.
TOL    :INOUT    Real.
        Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX  :IN       Integer.
        Maximum number of iterations.
ITER   :OUT      Integer.
        Number of iterations required to reach convergence, or
        ITMAX+1 if convergence criterion could not be achieved in
        ITMAX iterations.
ERR    :OUT      Real.
        Error estimate of error in final approximate solution, as
        defined by ITOL.
IERR   :OUT      Integer.
        Return error flag.
          IERR = 0 => All went well.
          IERR = 1 => Insufficient space allocated for WORK or IWORK.
          IERR = 2 => Method failed to converge in ITMAX steps.
          IERR = 3 => Error in user input.
                      Check input values of N, ITOL.
          IERR = 4 => User error tolerance set too tight.
                      Reset to 500*R1MACH(3).  Iteration proceeded.
          IERR = 5 => Preconditioning matrix, M, is not positive
                      definite.  (r,z) < 0.
          IERR = 6 => Matrix A is not positive definite.  (p,Ap) < 0.
IUNIT  :IN       Integer.
        Unit number on which to write the error at each iteration,
        if this is desired for monitoring convergence.  If unit
        number is 0, no writing will occur.
RWORK  :WORK     Real RWORK(LENW).
        Real array used for workspace.
LENW   :IN       Integer.
        Length of the real workspace, RWORK.  LENW >= 8*N.
IWORK  :WORK     Integer IWORK(LENIW).
        Used to hold pointers into the RWORK array.
        Upon return the following locations of IWORK hold information
        which may be of use to the user:
        IWORK(9)  Amount of Integer workspace actually used.
        IWORK(10) Amount of Real workspace actually used.
LENIW  :IN       Integer.
        Length of the integer workspace, IWORK.  LENIW >= 10.

*Description:
```

This routine is simply a driver for the SCGN routine. It calls the SSD2S routine to set up the preconditioning and then calls SCGN with the appropriate MATVEC and MSOLVE routines.

The Sparse Linear Algebra Package (SLAP) utilizes two matrix data structures: 1) the SLAP Triad format or 2) the SLAP Column format. The user can hand this routine either of the of these data structures and SLAP will figure out which on is being used and act accordingly.

========== S L A P Triad format ==========

This routine requires that the matrix A be stored in the SLAP Triad format. In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
          5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                                 1  2  3    4  5    6  7    8    9 10 11
       |11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
       |21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
       | 0  0 33  0 35|  JA:  1  4  6    8  9   12
       | 0  0  0 44  0|
       |51  0 53  0 55|
```

*Side Effects:

The SLAP Triad format (IA, JA, A) is modified internally to be the SLAP Column format. See above.

*Cautions:

This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

```
***SEE ALSO  SCGN, SSD2S, SSMV, SSMTV, SSDI
***REFERENCES  (NONE)
***ROUTINES CALLED  SCGN, SCHKW, SS2Y, SSD2S, SSDI, SSMTV, SSMV
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890921  Removed TeX from comments.  (FNF)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920407  COMMON BLOCK renamed SSLBLK.  (WRB)
   920511  Added complete declaration section.  (WRB)
   921113  Corrected C***CATEGORY line.  (FNF)
   END PROLOGUE
```

SSDCGS

```
      SUBROUTINE SSDCGS(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
     +   ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)
***BEGIN PROLOGUE  SSDCGS
***PURPOSE  Diagonally Scaled CGS Sparse Ax=b Solver.
            Routine to solve a linear system  Ax = b  using the
            BiConjugate Gradient Squared method with diagonal scaling.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SSDCGS-S, DSDCGS-D)
***KEYWORDS  ITERATIVE PRECONDITION, NON-SYMMETRIC LINEAR SYSTEM, SLAP,
             SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
     INTEGER ITER, IERR, IUNIT, LENW, IWORK(10), LENIW
     REAL B(N), X(N), A(NELT), TOL, ERR, RWORK(8*N)

     CALL SSDCGS(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
    $     ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW )

 *Arguments:
 N       :IN       Integer
          Order of the Matrix.
 B       :IN       Real B(N).
          Right-hand side vector.
 X       :INOUT    Real X(N).
          On input X is your initial guess for solution vector.
          On output X is the final approximate solution.
 NELT    :IN       Integer.
          Number of Non-Zeros stored in A.
 IA      :INOUT    Integer IA(NELT).
 JA      :INOUT    Integer JA(NELT).
 A       :INOUT    Real A(NELT).
          These arrays should hold the matrix A in either the SLAP
          Triad format or the SLAP Column format.  See "Description",
          below.  If the SLAP Triad format is chosen it is changed
          internally to the SLAP Column format.
 ISYM    :IN       Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the upper
          or lower triangle of the matrix is stored.
 ITOL    :IN       Integer.
          Flag to indicate type of convergence criterion.
          If ITOL=1, iteration stops when the 2-norm of the residual
          divided by the 2-norm of the right-hand side is less than TOL.
          This routine must calculate the residual from R = A*X - B.
          This is unnatural and hence expensive for this type of iter-
          ative method.  ITOL=2 is *STRONGLY* recommended.
```

```
         If ITOL=2, iteration stops when the 2-norm of M-inv times the
         residual divided by the 2-norm of M-inv times the right hand
         side is less than TOL, where M-inv time a vector is the pre-
         conditioning step.  This is the *NATURAL* stopping for this
         iterative method and is *STRONGLY* recommended.
         ITOL=11 is often useful for checking and comparing different
         routines.  For this case, the user must supply the "exact"
         solution or a very accurate approximation (one with an error
         much less than TOL) through a common block,
             COMMON /SSLBLK/ SOLN( )
         If ITOL=11, iteration stops when the 2-norm of the difference
         between the iterative approximation and the user-supplied
         solution divided by the 2-norm of the user-supplied solution
         is less than TOL.  Note that this requires the user to set up
         the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine.
         The routine with this declaration should be loaded before the
         stop test so that the correct length is used by the loader.
         This procedure is not standard Fortran and may not work
         correctly on your system (although it has worked on every
         system the authors have tried).  If ITOL is not 11 then this
         common block is indeed standard Fortran.
TOL     :INOUT    Real.
         Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX   :IN       Integer.
         Maximum number of iterations.
ITER    :OUT      Integer.
         Number of iterations required to reach convergence, or
         ITMAX+1 if convergence criterion could not be achieved in
         ITMAX iterations.
ERR     :OUT      Real.
         Error estimate of error in final approximate solution, as
         defined by ITOL.
IERR    :OUT      Integer.
         Return error flag.
           IERR = 0 => All went well.
           IERR = 1 => Insufficient space allocated for WORK or IWORK.
           IERR = 2 => Method failed to converge in ITMAX steps.
           IERR = 3 => Error in user input.
                       Check input values of N, ITOL.
           IERR = 4 => User error tolerance set too tight.
                       Reset to 500*R1MACH(3).  Iteration proceeded.
           IERR = 5 => Breakdown of the method detected.
                       (r0,r) approximately 0.
           IERR = 6 => Stagnation of the method detected.
                       (r0,v) approximately 0.
IUNIT   :IN       Integer.
         Unit number on which to write the error at each iteration,
         if this is desired for monitoring convergence.  If unit
         number is 0, no writing will occur.
RWORK   :WORK     Real RWORK(LENW).
         Real array used for workspace.
LENW    :IN       Integer.
         Length of the real workspace, RWORK.  LENW >= 8*N.
IWORK   :WORK     Integer IWORK(LENIW).
         Used to hold pointers into the RWORK array.
         Upon return the following locations of IWORK hold information
         which may be of use to the user:
         IWORK(9)  Amount of Integer workspace actually used.
         IWORK(10) Amount of Real workspace actually used.
LENIW   :IN       Integer.
```

Length of the integer workspace, IWORK. LENIW >= 10.

*Description:

This routine performs preconditioned BiConjugate gradient method on the Non-Symmetric positive definite linear system Ax=b. The preconditioner is M = DIAG(A), the diagonal of the matrix A. This is the simplest of preconditioners and vectorizes very well.

The Sparse Linear Algebra Package (SLAP) utilizes two matrix data structures: 1) the SLAP Triad format or 2) the SLAP Column format. The user can hand this routine either of the of these data structures and SLAP will figure out which on is being used and act accordingly.

========== S L A P Triad format ==========

This routine requires that the matrix A be stored in the SLAP Triad format. In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Side Effects:
The SLAP Triad format (IA, JA, A) is modified internally to be the SLAP Column format. See above.

*Cautions:
This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO SCGS, SLUBCG

***REFERENCES

1. P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric linear systems, Delft University of Technology Report 84-16, Department of Mathematics and Informatics, Delft, The Netherlands.
2. E. F. Kaasschieter, The solution of non-symmetric linear systems by biconjugate gradients or conjugate gradients squared, Delft University of Technology Report 86-21, Department of Mathematics and Informatics, Delft, The Netherlands.

***ROUTINES CALLED SCGS, SCHKW, SS2Y, SSDI, SSDS, SSMV

***REVISION HISTORY (YYMMDD)

```
871119  DATE WRITTEN
881213  Previous REVISION DATE
890915  Made changes requested at July 1989 CML Meeting.  (MKS)
890921  Removed TeX from comments.  (FNF)
890922  Numerous changes to prologue to make closer to SLATEC
        standard.  (FNF)
890929  Numerous changes to reduce SP/DP differences.  (FNF)
910411  Prologue converted to Version 4.0 format.  (BAB)
920407  COMMON BLOCK renamed SSLBLK.  (WRB)
920511  Added complete declaration section.  (WRB)
920929  Corrected format of references.  (FNF)
921113  Corrected C***CATEGORY line.  (FNF)
END PROLOGUE
```

SSDGMR

```
      SUBROUTINE SSDGMR(N, B, X, NELT, IA, JA, A, ISYM, NSAVE,
     +   TOL, ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)
***BEGIN PROLOGUE  SSDGMR
***PURPOSE  Diagonally Scaled GMRES Iterative Sparse Ax=b Solver.
            This routine uses the generalized minimum residual
            (GMRES) method with diagonal scaling to solve possibly
            non-symmetric linear systems of the form: Ax = b.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SSDGMR-S, DSDGMR-D)
***KEYWORDS  GENERALIZED MINIMUM RESIDUAL, ITERATIVE PRECONDITION,
             NON-SYMMETRIC LINEAR SYSTEM, SLAP, SPARSE
***AUTHOR  Brown, Peter, (LLNL), pnbrown@llnl.gov
           Hindmarsh, Alan, (LLNL), alanh@llnl.gov
           Seager, Mark K., (LLNL), seager@llnl.gov
             Lawrence Livermore National Laboratory
             PO Box 808, L-60
             Livermore, CA 94550 (510) 423-3141
***DESCRIPTION

 *Usage:
      INTEGER   N, NELT, IA(NELT), JA(NELT), ISYM, NSAVE, ITOL
      INTEGER   ITMAX, ITER, IERR, IUNIT, LENW, IWORK(LENIW), LENIW
      REAL      B(N), X(N), A(NELT), TOL, ERR, RWORK(LENW)

      CALL SSDGMR(N, B, X, NELT, IA, JA, A, ISYM, NSAVE,
     $     ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT,
     $     RWORK, LENW, IWORK, LENIW)

 *Arguments:
 N      :IN       Integer.
         Order of the Matrix.
 B      :IN       Real B(N).
         Right-hand side vector.
 X      :INOUT    Real X(N).
         On input X is your initial guess for solution vector.
         On output X is the final approximate solution.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :IN       Integer IA(NELT).
 JA     :IN       Integer JA(NELT).
 A      :IN       Real A(NELT).
         These arrays should hold the matrix A in either the SLAP
         Triad format or the SLAP Column format.  See "Description",
         below.  If the SLAP Triad format is chosen it is changed
         internally to the SLAP Column format.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 NSAVE  :IN       Integer.
         Number of direction vectors to save and orthogonalize against.
         Must be greater than 1.
 ITOL   :IN       Integer.
         Flag to indicate the type of convergence criterion used.
```

```
        ITOL=0  Means the  iteration stops when the test described
                below on  the  residual RL  is satisfied.  This is
                the  "Natural Stopping Criteria" for this routine.
                Other values  of   ITOL  cause  extra,   otherwise
                unnecessary, computation per iteration and     are
                therefore  much less  efficient.  See  ISSGMR (the
                stop test routine) for more information.
        ITOL=1  Means   the  iteration stops   when the first test
                described below on  the residual RL  is satisfied,
                and there  is either right  or  no preconditioning
                being used.
        ITOL=2  Implies     that   the  user    is   using    left
                preconditioning, and the second stopping criterion
                below is used.
        ITOL=3  Means the  iteration stops   when  the  third test
                described below on Minv*Residual is satisfied, and
                there is either left  or no  preconditioning begin
                used.
        ITOL=11 is    often  useful  for   checking  and comparing
                different routines.  For this case, the  user must
                supply  the  "exact" solution or  a  very accurate
                approximation (one with  an  error much less  than
                TOL) through a common block,
                    COMMON /SSLBLK/ SOLN( )
                If ITOL=11, iteration stops when the 2-norm of the
                difference between the iterative approximation and
                the user-supplied solution  divided by the  2-norm
                of the  user-supplied solution  is  less than TOL.
                Note that this requires  the  user to  set up  the
                "COMMON     /SSLBLK/ SOLN(LENGTH)"  in the calling
                routine.  The routine with this declaration should
                be loaded before the stop test so that the correct
                length is used by  the loader.  This procedure  is
                not standard Fortran and may not work correctly on
                your   system (although  it  has  worked  on every
                system the authors have tried).  If ITOL is not 11
                then this common block is indeed standard Fortran.
TOL     :INOUT    Real.
        Convergence criterion, as described below.  If TOL is set
        to zero on input, then a default value of 500*(the smallest
        positive magnitude, machine epsilon) is used.
ITMAX   :IN       Integer.
        Maximum number of iterations.  This routine uses the default
        of NRMAX = ITMAX/NSAVE to determine when each restart
        should occur.  See the description of NRMAX and MAXL in
        SGMRES for a full and frightfully interesting discussion of
        this topic.
ITER    :OUT      Integer.
        Number of iterations required to reach convergence, or
        ITMAX+1 if convergence criterion could not be achieved in
        ITMAX iterations.
ERR     :OUT      Real.
        Error estimate of error in final approximate solution, as
        defined by ITOL.  Letting norm() denote the Euclidean
        norm, ERR is defined as follows...
        If ITOL=0, then ERR = norm(SB*(B-A*X(L)))/norm(SB*B),
                                for right or no preconditioning, and
                           ERR = norm(SB*(M-inverse)*(B-A*X(L)))/
                                 norm(SB*(M-inverse)*B),
                                for left preconditioning.
```

```
        If ITOL=1, then ERR = norm(SB*(B-A*X(L)))/norm(SB*B),
                              since right or no preconditioning
                              being used.
        If ITOL=2, then ERR = norm(SB*(M-inverse)*(B-A*X(L)))/
                               norm(SB*(M-inverse)*B),
                              since left preconditioning is being
                              used.
        If ITOL=3, then ERR =  Max  |(Minv*(B-A*X(L)))(i)/x(i)|
                              i=1,n
        If ITOL=11, then ERR = norm(SB*(X(L)-SOLN))/norm(SB*SOLN).
IERR   :OUT      Integer.
        Return error flag.
              IERR = 0 => All went well.
              IERR = 1 => Insufficient storage allocated for
                          RGWK or IGWK.
              IERR = 2 => Routine SPIGMR failed to reduce the norm
                          of the current residual on its last call,
                          and so the iteration has stalled.  In
                          this case, X equals the last computed
                          approximation.  The user must either
                          increase MAXL, or choose a different
                          initial guess.
              IERR =-1 => Insufficient length for RGWK array.
                          IGWK(6) contains the required minimum
                          length of the RGWK array.
              IERR =-2 => Inconsistent ITOL and JPRE values.
        For IERR <= 2, RGWK(1) = RHOL, which is the norm on the
        left-hand-side of the relevant stopping test defined
        below associated with the residual for the current
        approximation X(L).
IUNIT  :IN       Integer.
        Unit number on which to write the error at each iteration,
        if this is desired for monitoring convergence.  If unit
        number is 0, no writing will occur.
RWORK  :WORK    Real RWORK(LENW).
        Real array of size LENW.
LENW   :IN       Integer.
        Length of the real workspace, RWORK.
        LENW >= 1 + N*(NSAVE+7) + NSAVE*(NSAVE+3).
        For the recommended values of NSAVE (10), RWORK has size at
        least 131 + 17*N.
IWORK  :INOUT    Integer IWORK(USER DEFINED >= 30).
        Used to hold pointers into the RWORK array.
        Upon return the following locations of IWORK hold information
        which may be of use to the user:
        IWORK(9)  Amount of Integer workspace actually used.
        IWORK(10) Amount of Real workspace actually used.
LENIW  :IN       Integer.
        Length of the integer workspace IWORK.  LENIW >= 30.

*Description:
      SSDGMR solves a linear system A*X = B rewritten in the form:

       (SB*A*(M-inverse)*(SX-inverse))*(SX*M*X) = SB*B,

      with right preconditioning, or

       (SB*(M-inverse)*A*(SX-inverse))*(SX*X) = SB*(M-inverse)*B,

      with left preconditioning, where A is an n-by-n real matrix,
```

X and B are N-vectors, SB and SX are diagonal scaling matrices, and M is the diagonal of A. It uses preconditioned Krylov subpace methods based on the generalized minimum residual method (GMRES). This routine is a driver routine which assumes a SLAP matrix data structure and sets up the necessary information to do diagonal preconditioning and calls the main GMRES routine SGMRES for the solution of the linear system. SGMRES optionally performs either the full orthogonalization version of the GMRES algorithm or an incomplete variant of it. Both versions use restarting of the linear iteration by default, although the user can disable this feature.

The GMRES algorithm generates a sequence of approximations X(L) to the true solution of the above linear system. The convergence criteria for stopping the iteration is based on the size of the scaled norm of the residual R(L) = B - A*X(L). The actual stopping test is either:

```
norm(SB*(B-A*X(L))) .le. TOL*norm(SB*B),
```

for right preconditioning, or

```
norm(SB*(M-inverse)*(B-A*X(L))) .le.
        TOL*norm(SB*(M-inverse)*B),
```

for left preconditioning, where norm() denotes the Euclidean norm, and TOL is a positive scalar less than one input by the user. If TOL equals zero when SSDGMR is called, then a default value of 500*(the smallest positive magnitude, machine epsilon) is used. If the scaling arrays SB and SX are used, then ideally they should be chosen so that the vectors SX*X(or SX*M*X) and SB*B have all their components approximately equal to one in magnitude. If one wants to use the same scaling in X and B, then SB and SX can be the same array in the calling program.

The following is a list of the other routines and their functions used by GMRES:

```
SGMRES  Contains the matrix structure independent driver
        routine for GMRES.
SPIGMR  Contains the main iteration loop for GMRES.
SORTH   Orthogonalizes a new vector against older basis vectors.
SHEQR   Computes a QR decomposition of a Hessenberg matrix.
SHELS   Solves a Hessenberg least-squares system, using QR
        factors.
RLCALC  Computes the scaled residual RL.
XLCALC  Computes the solution XL.
ISSGMR  User-replaceable stopping routine.
```

The Sparse Linear Algebra Package (SLAP) utilizes two matrix data structures: 1) the SLAP Triad format or 2) the SLAP Column format. The user can hand this routine either of the of these data structures and SLAP will figure out which on is being used and act accordingly.

========== S L A P Triad format ==========
This routine requires that the matrix A be stored in the SLAP Triad format. In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies

three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Side Effects:

The SLAP Triad format (IA, JA, A) is modified internally to be the SLAP Column format. See above.

*Cautions:

This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that

```
     this logical unit is attached to a file or terminal before calling
     this routine with a non-zero value for IUNIT.  This routine does
     not check for the validity of a non-zero IUNIT unit number.

***REFERENCES  1. Peter N. Brown and A. C. Hindmarsh, Reduced Storage
                  Matrix Methods in Stiff ODE Systems, Lawrence Liver-
                  more National Laboratory Report UCRL-95088, Rev. 1,
                  Livermore, California, June 1987.
***ROUTINES CALLED  SCHKW, SGMRES, SS2Y, SSDI, SSDS, SSMV
***REVISION HISTORY  (YYMMDD)
   880615  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920407  COMMON BLOCK renamed SSLBLK.  (WRB)
   920511  Added complete declaration section.  (WRB)
   920929  Corrected format of references.  (FNF)
   END PROLOGUE
```

SSDI

```
      SUBROUTINE SSDI (N, B, X, NELT, IA, JA, A, ISYM, RWORK, IWORK)
***BEGIN PROLOGUE  SSDI
***PURPOSE  Diagonal Matrix Vector Multiply.
            Routine to calculate the product  X = DIAG*B, where DIAG
            is a diagonal matrix.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (SSDI-S, DSDI-D)
***KEYWORDS  ITERATIVE PRECONDITION, LINEAR SYSTEM SOLVE, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER  N, NELT, IA(NELT), JA(NELT), ISYM, IWORK(10)
     REAL B(N), X(N), A(NELT), RWORK(USER DEFINED)

     CALL SSDI (N, B, X, NELT, IA, JA, A, ISYM, RWORK, IWORK)

 *Arguments:
 N       :IN       Integer
          Order of the Matrix.
 B       :IN       Real B(N).
          Vector to multiply the diagonal by.
 X       :OUT      Real X(N).
          Result of DIAG*B.
 NELT    :DUMMY    Integer.
 IA      :DUMMY    Integer IA(NELT).
 JA      :DUMMY    Integer JA(NELT).
 A       :DUMMY    Real A(NELT).
 ISYM    :DUMMY    Integer.
          These are for compatibility with SLAP MSOLVE calling sequence.
 RWORK   :IN       Real RWORK(USER DEFINED).
          Work array holding the diagonal of some matrix to scale
          B by.  This array must be set by the user or by a call
          to the SLAP routine SSDS or SSD2S.  The length of RWORK
          must be >= IWORK(4)+N.
 IWORK   :IN       Integer IWORK(10).
          IWORK(4) holds the offset into RWORK for the diagonal matrix
          to scale B by.  This is usually set up by the SLAP pre-
          conditioner setup routines SSDS or SSD2S.

 *Description:
          This routine is supplied with the SLAP package to perform
          the  MSOLVE  operation for iterative drivers that require
          diagonal  Scaling  (e.g., SSDCG, SSDBCG).   It  conforms
          to the SLAP MSOLVE CALLING CONVENTION  and hence does not
          require an interface routine as do some of the other pre-
          conditioners supplied with SLAP.

***SEE ALSO  SSDS, SSD2S
***REFERENCES  (NONE)
```

```
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920511  Added complete declaration section.  (WRB)
   930701  Updated CATEGORY section.  (FNF, WRB)
   END PROLOGUE
```

SSDOMN

```
      SUBROUTINE SSDOMN(N, B, X, NELT, IA, JA, A, ISYM, NSAVE, ITOL, TOL,
     $     ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW )
***BEGIN PROLOGUE  SSDOMN
***PURPOSE  Diagonally Scaled Orthomin Sparse Iterative Ax=b Solver.
            Routine to solve a general linear system  Ax = b  using
            the Orthomin method with diagonal scaling.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SSDOMN-S, DSDOMN-D)
***KEYWORDS  ITERATIVE PRECONDITION, NON-SYMMETRIC LINEAR SYSTEM SOLVE,
             SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, NSAVE, ITOL, ITMAX
     INTEGER ITER, IERR, IUNIT, LENW, IWORK(10), LENIW
     REAL    B(N), X(N), A(NELT), TOL, ERR
     REAL    RWORK(7*N+3*N*NSAVE+NSAVE)

     CALL SSDOMN(N, B, X, NELT, IA, JA, A, ISYM, NSAVE, ITOL, TOL,
     $     ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW )

 *Arguments:
 N      :IN       Integer.
         Order of the Matrix.
 B      :IN       Real B(N).
         Right-hand side vector.
 X      :INOUT    Real X(N).
         On input X is your initial guess for solution vector.
         On output X is the final approximate solution.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :IN       Integer IA(NELT).
 JA     :IN       Integer JA(NELT).
 A      :IN       Real A(NELT).
         These arrays should hold the matrix A in either the SLAP
         Triad format or the SLAP Column format.  See "Description",
         below.  If the SLAP Triad format is chosen, it is changed
         internally to the SLAP Column format.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 NSAVE  :IN       Integer.
         Number of direction vectors to save and orthogonalize against.
 ITOL   :IN       Integer.
         Flag to indicate type of convergence criterion.
         If ITOL=1, iteration stops when the 2-norm of the residual
         divided by the 2-norm of the right-hand side is less than TOL.
```

```
        If ITOL=2, iteration stops when the 2-norm of M-inv times the
        residual divided by the 2-norm of M-inv times the right hand
        side is less than TOL, where M-inv is the inverse of the
        diagonal of A.
        ITOL=11 is often useful for checking and comparing different
        routines.  For this case, the user must supply the "exact"
        solution or a very accurate approximation (one with an error
        much less than TOL) through a common block,
            COMMON /SSLBLK/ SOLN( )
        If ITOL=11, iteration stops when the 2-norm of the difference
        between the iterative approximation and the user-supplied
        solution divided by the 2-norm of the user-supplied solution
        is less than TOL.
TOL    :INOUT    Real.
        Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX  :IN       Integer.
        Maximum number of iterations.
ITER   :OUT      Integer.
        Number of iterations required to reach convergence, or
        ITMAX+1 if convergence criterion could not be achieved in
        ITMAX iterations.
ERR    :OUT      Real.
        Error estimate of error in final approximate solution, as
        defined by ITOL.
IERR   :OUT      Integer.
        Return error flag.
          IERR = 0 => All went well.
          IERR = 1 => Insufficient space allocated for WORK or IWORK.
          IERR = 2 => Method failed to converge in ITMAX steps.
          IERR = 3 => Error in user input.
                      Check input values of N, ITOL.
          IERR = 4 => User error tolerance set too tight.
                      Reset to 500*R1MACH(3).  Iteration proceeded.
          IERR = 5 => Preconditioning matrix, M, is not positive
                      definite.  (r,z) < 0.
          IERR = 6 => Breakdown of method detected.
                      (p,Ap) < epsilon**2.
IUNIT  :IN       Integer.
        Unit number on which to write the error at each iteration,
        if this is desired for monitoring convergence.  If unit
        number is 0, no writing will occur.
RWORK  :WORK     Real RWORK(LENW).
        Real array used for workspace.
LENW   :IN       Integer.
        Length of the real workspace, RWORK.
        LENW >= 7*N+NSAVE*(3*N+1).
IWORK  :WORK     Integer IWORK(LENIW).
        Used to hold pointers into the RWORK array.
LENIW  :IN       Integer.
        Length of the integer workspace, IWORK.  LENIW >= 10.

*Description:
      This routine  is simply a driver  for  the SOMN routine.  It
      calls the SSDS  routine  to set  up the  preconditioning and
      then   calls SOMN with the   appropriate   MATVEC and MSOLVE
      routines.

      The Sparse Linear Algebra Package (SLAP) utilizes two matrix
      data structures: 1) the  SLAP Triad  format or  2)  the SLAP
      Column format.  The user can hand this routine either of the
```

of these data structures and SLAP will figure out which on is being used and act accordingly.

```
=================== S L A P Triad format ===================
```

In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

```
=================== S L A P Column format ==================
```

In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Side Effects:

```
      The SLAP Triad format (IA, JA, A)  is modified internally to
      be the SLAP Column format.  See above.

 *Cautions:
     This routine will attempt to write to the Fortran logical output
     unit IUNIT, if IUNIT .ne. 0.  Thus, the user must make sure that
     this logical unit is attached to a file or terminal before calling
     this routine with a non-zero value for IUNIT.  This routine does
     not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO  SOMN, SSLUOM
***REFERENCES  (NONE)
***ROUTINES CALLED  SCHKW, SOMN, SS2Y, SSDI, SSDS, SSMV
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890921  Removed TeX from comments.  (FNF)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920407  COMMON BLOCK renamed SSLBLK.  (WRB)
   920511  Added complete declaration section.  (WRB)
   921113  Corrected C***CATEGORY line.  (FNF)
   END PROLOGUE
```

SSDS

```
      SUBROUTINE SSDS (N, NELT, IA, JA, A, ISYM, DINV)
***BEGIN PROLOGUE  SSDS
***PURPOSE  Diagonal Scaling Preconditioner SLAP Set Up.
            Routine to compute the inverse of the diagonal of a matrix
            stored in the SLAP Column format.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2E
***TYPE      SINGLE PRECISION (SSDS-S, DSDS-D)
***KEYWORDS  DIAGONAL, SLAP SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM
     REAL    A(NELT), DINV(N)

     CALL SSDS( N, NELT, IA, JA, A, ISYM, DINV )

 *Arguments:
 N       :IN       Integer.
          Order of the Matrix.
 NELT    :IN       Integer.
          Number of elements in arrays IA, JA, and A.
 IA      :INOUT    Integer IA(NELT).
 JA      :INOUT    Integer JA(NELT).
 A       :INOUT    Real A(NELT).
          These arrays should hold the matrix A in the SLAP Column
          format.  See "Description", below.
 ISYM    :IN       Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the upper
          or lower triangle of the matrix is stored.
 DINV    :OUT      Real DINV(N).
          Upon return this array holds 1./DIAG(A).

 *Description
       ================== S L A P Column format ==================
       This routine  requires that  the matrix A  be stored in  the
       SLAP Column format.  In this format the non-zeros are stored
       counting down columns (except for  the diagonal entry, which
       must appear first in each  "column")  and are stored  in the
       real array A.  In other words, for each column in the matrix
       put the diagonal entry in A.  Then put in the other non-zero
       elements going down   the  column (except  the diagonal)  in
       order.  The IA array holds the row  index for each non-zero.
       The JA array holds the offsets into the IA, A arrays for the
       beginning of   each    column.    That  is,    IA(JA(ICOL)),
       A(JA(ICOL)) points to the beginning of the ICOL-th column in
       IA and  A.  IA(JA(ICOL+1)-1),  A(JA(ICOL+1)-1) points to the
       end  of   the ICOL-th  column.  Note   that  we  always have
```

JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

With the SLAP format all of the "inner loops" of this routine should vectorize on machines with hardware support for vector gather/scatter operations. Your compiler may require a compiler directive to convince it that there are no implicit vector dependencies. Compiler directives for the Alliant FX/Fortran and CRI CFT/CFT77 compilers are supplied with the standard SLAP distribution.

*Cautions:

This routine assumes that the diagonal of A is all non-zero and that the operation DINV = 1.0/DIAG(A) will not underflow or overflow. This is done so that the loop vectorizes. Matrices with zero or near zero or very large entries will have numerical difficulties and must be fixed before this routine is called.

***REFERENCES (NONE)
***ROUTINES CALLED (NONE)
***REVISION HISTORY (YYMMDD)
871119 DATE WRITTEN
881213 Previous REVISION DATE
890915 Made changes requested at July 1989 CML Meeting. (MKS)
890922 Numerous changes to prologue to make closer to SLATEC standard. (FNF)
890929 Numerous changes to reduce SP/DP differences. (FNF)
910411 Prologue converted to Version 4.0 format. (BAB)
920511 Added complete declaration section. (WRB)
930701 Updated CATEGORY section. (FNF, WRB)
END PROLOGUE

SSDSCL

```
      SUBROUTINE SSDSCL( N, NELT, IA, JA, A, ISYM, X, B, DINV, JOB,
     +   ITOL)
***BEGIN PROLOGUE  SSDSCL
***PURPOSE  Diagonal Scaling of system Ax = b.
            This routine scales (and unscales) the system  Ax = b
            by symmetric diagonal scaling.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2E
***TYPE      SINGLE PRECISION (SSDSCL-S, DSDSCL-D)
***KEYWORDS  DIAGONAL, SLAP SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

    This routine scales (and unscales) the system Ax = b by symmetric
    diagonal scaling.  The new system is:
         -1/2  -1/2  1/2      -1/2
        D    AD    (D   x) = D    b
    when scaling is selected with the JOB parameter.  When unscaling
    is selected this process is reversed.  The true solution is also
    scaled or unscaled if ITOL is set appropriately, see below.

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, JOB, ITOL
     REAL    A(NELT), X(N), B(N), DINV(N)

     CALL SSDSCL( N, NELT, IA, JA, A, ISYM, X, B, DINV, JOB, ITOL )

 *Arguments:
 N      :IN       Integer
         Order of the Matrix.
 NELT   :IN       Integer.
         Number of elements in arrays IA, JA, and A.
 IA     :IN       Integer IA(NELT).
 JA     :IN       Integer JA(NELT).
 A      :IN       Real A(NELT).
         These arrays should hold the matrix A in the SLAP Column
         format.  See "Description", below.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 X      :INOUT    Real X(N).
         Initial guess that will be later used in the iterative
         solution.
         of the scaled system.
 B      :INOUT    Real B(N).
         Right hand side vector.
 DINV   :INOUT    Real DINV(N).
         Upon return this array holds 1./DIAG(A).
         This is an input if JOB = 0.
```

```
JOB    :IN       Integer.
        Flag indicating whether to scale or not.
        JOB non-zero means do scaling.
        JOB = 0 means do unscaling.
ITOL   :IN       Integer.
        Flag indicating what type of error estimation to do in the
        iterative method.  When ITOL = 11 the exact solution from
        common block SSLBLK will be used.  When the system is scaled
        then the true solution must also be scaled.  If ITOL is not
        11 then this vector is not referenced.

*Common Blocks:
SOLN    :INOUT   Real SOLN(N).  COMMON BLOCK /SSLBLK/
        The true solution, SOLN, is scaled (or unscaled) if ITOL is
        set to 11, see above.

*Description
      ================== S L A P Column format ==================
      This routine  requires that  the matrix A  be stored in  the
      SLAP Column format.  In this format the non-zeros are stored
      counting down columns (except for  the diagonal entry, which
      must appear first in each  "column")  and are stored  in the
      real array A.  In other words, for each column in the matrix
      put the diagonal entry in A.  Then put in the other non-zero
      elements going down   the  column (except  the diagonal)  in
      order.  The IA array holds the row  index for each non-zero.
      The JA array holds the offsets into the IA, A arrays for the
      beginning of   each    column.    That  is,    IA(JA(ICOL)),
      A(JA(ICOL)) points to the beginning of the ICOL-th column in
      IA and  A.  IA(JA(ICOL+1)-1),  A(JA(ICOL+1)-1) points to the
      end  of   the ICOL-th  column.  Note   that  we  always have
      JA(N+1) = NELT+1, where  N  is the number of columns in  the
      matrix and  NELT   is the number of non-zeros in the matrix.

      Here is an example of the  SLAP Column  storage format for a
      5x5 Matrix (in the A and IA arrays '|'  denotes the end of a
      column):

          5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                             1  2  3    4  5    6  7    8    9 10 11
      |11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
      |21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
      | 0  0 33  0 35|  JA:  1  4  6    8  9   12
      | 0  0  0 44  0|
      |51  0 53  0 55|

      With the SLAP  format  all  of  the   "inner  loops" of this
      routine should vectorize  on  machines with hardware support
      for vector   gather/scatter  operations.  Your compiler  may
      require a compiler directive to  convince it that  there are
      no  implicit  vector  dependencies.  Compiler directives for
      the Alliant    FX/Fortran and CRI   CFT/CFT77 compilers  are
      supplied with the standard SLAP distribution.


*Cautions:
      This routine assumes that the diagonal of A is all  non-zero
      and that the operation DINV = 1.0/DIAG(A)  will  not  under-
      flow or overflow. This is done so that the loop  vectorizes.
      Matrices  with zero or near zero or very  large entries will
```

```
       have numerical difficulties  and  must  be fixed before this
       routine is called.

***SEE ALSO  SSDCG
***REFERENCES  (NONE)
***ROUTINES CALLED  (NONE)
***COMMON BLOCKS    SSLBLK
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   910502  Added C***FIRST EXECUTABLE STATEMENT line.  (FNF)
   920407  COMMON BLOCK renamed SSLBLK.  (WRB)
   920511  Added complete declaration section.  (WRB)
   921113  Corrected C***CATEGORY line.  (FNF)
   930701  Updated CATEGORY section.  (FNF, WRB)
   END PROLOGUE
```

SSGS

```
      SUBROUTINE SSGS(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
     $     ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW )
***BEGIN PROLOGUE  SSGS
***PURPOSE  Gauss-Seidel Method Iterative Sparse Ax = b Solver.
            Routine to solve a general linear system  Ax = b  using
            Gauss-Seidel iteration.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SSGS-S, DSGS-D)
***KEYWORDS  ITERATIVE PRECONDITION, LINEAR SYSTEM, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
      INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
      INTEGER ITER, IERR, IUNIT, LENW, IWORK(NL+2*N+1), LENIW
      REAL B(N), X(N), A(NELT), TOL, ERR, RWORK(NL+3*N)

      CALL SSGS(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
     $     ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW )

 *Arguments:
 N       :IN       Integer.
          Order of the Matrix.
 B       :IN       Real B(N).
          Right-hand side vector.
 X       :INOUT    Real X(N).
          On input X is your initial guess for solution vector.
          On output X is the final approximate solution.
 NELT    :IN       Integer.
          Number of Non-Zeros stored in A.
 IA      :INOUT    Integer IA(NELT).
 JA      :INOUT    Integer JA(NELT).
 A       :INOUT    Real A(NELT).
          These arrays should hold the matrix A in either the SLAP
          Triad format or the SLAP Column format.  See "Description",
          below.  If the SLAP Triad format is chosen it is changed
          internally to the SLAP Column format.
 ISYM    :IN       Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the lower
          lower triangle of the matrix is stored.
 ITOL    :IN       Integer.
          Flag to indicate type of convergence criterion.
          If ITOL=1, iteration stops when the 2-norm of the residual
          divided by the 2-norm of the right-hand side is less than TOL.
          If ITOL=2, iteration stops when the 2-norm of M-inv times the
          residual divided by the 2-norm of M-inv times the right hand
          side is less than TOL, where M-inv is the inverse of the
          diagonal of A.
```

```
         ITOL=11 is often useful for checking and comparing different
         routines.  For this case, the user must supply the "exact"
         solution or a very accurate approximation (one with an error
         much less than TOL) through a common block,
             COMMON /SSLBLK/ SOLN( )
         If ITOL=11, iteration stops when the 2-norm of the difference
         between the iterative approximation and the user-supplied
         solution divided by the 2-norm of the user-supplied solution
         is less than TOL.  Note that this requires the user to set up
         the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine.
         The routine with this declaration should be loaded before the
         stop test so that the correct length is used by the loader.
         This procedure is not standard Fortran and may not work
         correctly on your system (although it has worked on every
         system the authors have tried).  If ITOL is not 11 then this
         common block is indeed standard Fortran.
TOL     :INOUT    Real.
         Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX   :IN       Integer.
         Maximum number of iterations.
ITER    :OUT      Integer.
         Number of iterations required to reach convergence, or
         ITMAX+1 if convergence criterion could not be achieved in
         ITMAX iterations.
ERR     :OUT      Real.
         Error estimate of error in final approximate solution, as
         defined by ITOL.
IERR    :OUT      Integer.
         Return error flag.
           IERR = 0 => All went well.
           IERR = 1 => Insufficient space allocated for WORK or IWORK.
           IERR = 2 => Method failed to converge in ITMAX steps.
           IERR = 3 => Error in user input.
                       Check input values of N, ITOL.
           IERR = 4 => User error tolerance set too tight.
                       Reset to 500*R1MACH(3).  Iteration proceeded.
           IERR = 5 => Preconditioning matrix, M, is not positive
                       definite.  (r,z) < 0.
           IERR = 6 => Matrix A is not positive definite.  (p,Ap) < 0.
IUNIT   :IN       Integer.
         Unit number on which to write the error at each iteration,
         if this is desired for monitoring convergence.  If unit
         number is 0, no writing will occur.
RWORK   :WORK     Real RWORK(LENW).
         Real array used for workspace.
LENW    :IN       Integer.
         Length of the real workspace, RWORK.  LENW >= NL+3*N.
         NL is the number of non-zeros in the lower triangle of the
         matrix (including the diagonal).
IWORK   :WORK     Integer IWORK(LENIW).
         Integer array used for workspace.
         Upon return the following locations of IWORK hold information
         which may be of use to the user:
         IWORK(9)  Amount of Integer workspace actually used.
         IWORK(10) Amount of Real workspace actually used.
LENIW   :IN       Integer.
         Length of the integer workspace, IWORK.  LENIW >= NL+N+11.
         NL is the number of non-zeros in the lower triangle of the
         matrix (including the diagonal).
```

*Description

The Sparse Linear Algebra Package (SLAP) utilizes two matrix data structures: 1) the SLAP Triad format or 2) the SLAP Column format. The user can hand this routine either of the of these data structures and SLAP will figure out which on is being used and act accordingly.

```
=================== S L A P Triad format ===================
```

This routine requires that the matrix A be stored in the SLAP Triad format. In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

```
=================== S L A P Column format ==================
```

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
```

```
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Side Effects:
The SLAP Triad format (IA, JA, A) is modified internally to be the SLAP Column format. See above.

*Cautions:
This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO SSJAC, SIR
***REFERENCES (NONE)
***ROUTINES CALLED SCHKW, SIR, SS2LT, SS2Y, SSLI, SSMV
***REVISION HISTORY (YYMMDD)
871119 DATE WRITTEN
881213 Previous REVISION DATE
890915 Made changes requested at July 1989 CML Meeting. (MKS)
890921 Removed TeX from comments. (FNF)
890922 Numerous changes to prologue to make closer to SLATEC standard. (FNF)
890929 Numerous changes to reduce SP/DP differences. (FNF)
910411 Prologue converted to Version 4.0 format. (BAB)
920407 COMMON BLOCK renamed SSLBLK. (WRB)
920511 Added complete declaration section. (WRB)
921019 Corrected NEL to NL. (FNF)
END PROLOGUE

SSICCG

```
      SUBROUTINE SSICCG(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
     +   ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)
***BEGIN PROLOGUE  SSICCG
***PURPOSE  Incomplete Cholesky Conjugate Gradient Sparse Ax=b Solver.
            Routine to solve a symmetric positive definite linear
            system  Ax = b  using the incomplete Cholesky
            Preconditioned Conjugate Gradient method.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2B4
***TYPE      SINGLE PRECISION (SSICCG-S, DSICCG-D)
***KEYWORDS  INCOMPLETE CHOLESKY, ITERATIVE PRECONDITION, SLAP, SPARSE,
             SYMMETRIC LINEAR SYSTEM
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
     INTEGER ITER, IERR, IUNIT, LENW, IWORK(NL+2*N+1), LENIW
     REAL B(N), X(N), A(NELT), TOL, ERR, RWORK(NL+5*N)

     CALL SSICCG(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
    $     ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW )

 *Arguments:
 N      :IN       Integer.
         Order of the Matrix.
 B      :IN       Real B(N).
         Right-hand side vector.
 X      :INOUT    Real X(N).
         On input X is your initial guess for solution vector.
         On output X is the final approximate solution.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :INOUT    Integer IA(NELT).
 JA     :INOUT    Integer JA(NELT).
 A      :INOUT    Real A(NELT).
         These arrays should hold the matrix A in either the SLAP
         Triad format or the SLAP Column format.  See "Description",
         below.  If the SLAP Triad format is chosen it is changed
         internally to the SLAP Column format.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 ITOL   :IN       Integer.
         Flag to indicate type of convergence criterion.
         If ITOL=1, iteration stops when the 2-norm of the residual
         divided by the 2-norm of the right-hand side is less than TOL.
         If ITOL=2, iteration stops when the 2-norm of M-inv times the
         residual divided by the 2-norm of M-inv times the right hand
```

```
        side is less than TOL, where M-inv is the inverse of the
        diagonal of A.
        ITOL=11 is often useful for checking and comparing different
        routines.  For this case, the user must supply the "exact"
        solution or a very accurate approximation (one with an error
        much less than TOL) through a common block,
            COMMON /SSLBLK/ SOLN( )
        If ITOL=11, iteration stops when the 2-norm of the difference
        between the iterative approximation and the user-supplied
        solution divided by the 2-norm of the user-supplied solution
        is less than TOL.  Note that this requires the user to set up
        the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine.
        The routine with this declaration should be loaded before the
        stop test so that the correct length is used by the loader.
        This procedure is not standard Fortran and may not work
        correctly on your system (although it has worked on every
        system the authors have tried).  If ITOL is not 11 then this
        common block is indeed standard Fortran.
TOL    :INOUT    Real.
        Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX  :IN       Integer.
        Maximum number of iterations.
ITER   :OUT      Integer.
        Number of iterations required to reach convergence, or
        ITMAX+1 if convergence criterion could not be achieved in
        ITMAX iterations.
ERR    :OUT      Real.
        Error estimate of error in final approximate solution, as
        defined by ITOL.
IERR   :OUT      Integer.
        Return error flag.
          IERR = 0 => All went well.
          IERR = 1 => Insufficient space allocated for WORK or IWORK.
          IERR = 2 => Method failed to converge in ITMAX steps.
          IERR = 3 => Error in user input.
                      Check input values of N, ITOL.
          IERR = 4 => User error tolerance set too tight.
                      Reset to 500*R1MACH(3).  Iteration proceeded.
          IERR = 5 => Preconditioning matrix, M, is not positive
                      definite.  (r,z) < 0.
          IERR = 6 => Matrix A is not positive definite.  (p,Ap) < 0.
          IERR = 7 => Incomplete factorization broke down and was
                      fudged.  Resulting preconditioning may be less
                      than the best.
IUNIT  :IN       Integer.
        Unit number on which to write the error at each iteration,
        if this is desired for monitoring convergence.  If unit
        number is 0, no writing will occur.
RWORK  :WORK     Real RWORK(LENW).
        Real array used for workspace.
LENW   :IN       Integer.
        Length of the real workspace, RWORK.  LENW >= NL+5*N.
        NL is the number of non-zeros in the lower triangle of the
        matrix (including the diagonal).
IWORK  :WORK     Integer IWORK(LENIW).
        Integer array used for workspace.
        Upon return the following locations of IWORK hold information
        which may be of use to the user:
        IWORK(9)  Amount of Integer workspace actually used.
        IWORK(10) Amount of Real workspace actually used.
```

LENIW :IN Integer.
Length of the integer workspace, IWORK. LENIW >= NL+N+11. NL is the number of non-zeros in the lower triangle of the matrix (including the diagonal).

*Description:

This routine performs preconditioned conjugate gradient method on the symmetric positive definite linear system Ax=b. The preconditioner is the incomplete Cholesky (IC) factorization of the matrix A. See SSICS for details about the incomplete factorization algorithm. One should note here however, that the IC factorization is a slow process and that one should save factorizations for reuse, if possible. The MSOLVE operation (handled in SSLLTI) does vectorize on machines with hardware gather/scatter and is quite fast.

The Sparse Linear Algebra Package (SLAP) utilizes two matrix data structures: 1) the SLAP Triad format or 2) the SLAP Column format. The user can hand this routine either of the of these data structures and SLAP will figure out which on is being used and act accordingly.

========== S L A P Triad format ==========

This routine requires that the matrix A be stored in the SLAP Triad format. In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in

order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Side Effects:
The SLAP Triad format (IA, JA, A) is modified internally to be the SLAP Column format. See above.

*Cautions:
This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO SCG, SSLLTI
***REFERENCES
1. Louis Hageman and David Young, Applied Iterative Methods, Academic Press, New York, 1981.
2. Concus, Golub and O'Leary, A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations, in Sparse Matrix Computations, Bunch and Rose, Eds., Academic Press, New York, 1979.
***ROUTINES CALLED SCG, SCHKW, SS2Y, SSICS, SSLLTI, SSMV, XERMSG
***REVISION HISTORY (YYMMDD)
871119 DATE WRITTEN
881213 Previous REVISION DATE
890915 Made changes requested at July 1989 CML Meeting. (MKS)
890921 Removed TeX from comments. (FNF)
890922 Numerous changes to prologue to make closer to SLATEC standard. (FNF)
890929 Numerous changes to reduce SP/DP differences. (FNF)
900805 Changed XERRWV calls to calls to XERMSG. (RWC)
910411 Prologue converted to Version 4.0 format. (BAB)
920407 COMMON BLOCK renamed SSLBLK. (WRB)
920511 Added complete declaration section. (WRB)
920929 Corrected format of references. (FNF)
921019 Corrected NEL to NL. (FNF)
END PROLOGUE

SSICO

```
      SUBROUTINE SSICO (A, LDA, N, KPVT, RCOND, Z)
***BEGIN PROLOGUE  SSICO
***PURPOSE  Factor a symmetric matrix by elimination with symmetric
            pivoting and estimate the condition number of the matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1A
***TYPE      SINGLE PRECISION (SSICO-S, DSICO-D, CHICO-C, CSICO-C)
***KEYWORDS  CONDITION NUMBER, LINEAR ALGEBRA, LINPACK,
             MATRIX FACTORIZATION, SYMMETRIC
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SSICO factors a real symmetric matrix by elimination with
      symmetric pivoting and estimates the condition of the matrix.

      If  RCOND  is not needed, SSIFA is slightly faster.
      To solve  A*X = B , follow SSICO by SSISL.
      To compute  INVERSE(A)*C , follow SSICO by SSISL.
      To compute  INVERSE(A) , follow SSICO by SSIDI.
      To compute  DETERMINANT(A) , follow SSICO by SSIDI.
      To compute  INERTIA(A), follow SSICO by SSIDI.

      On Entry

         A       REAL(LDA, N)
                 the symmetric matrix to be factored.
                 Only the diagonal and upper triangle are used.

         LDA     INTEGER
                 the leading dimension of the array  A .

         N       INTEGER
                 the order of the matrix  A .

      Output

         A       a block diagonal matrix and the multipliers which
                 were used to obtain it.
                 The factorization can be written  A = U*D*TRANS(U)
                 where  U  is a product of permutation and unit
                 upper triangular matrices , TRANS(U) is the
                 transpose of  U , and  D  is block diagonal
                 with 1 by 1 and 2 by 2 blocks.

         KPVT    INTEGER(N)
                 an integer vector of pivot indices.

         RCOND   REAL
                 an estimate of the reciprocal condition of  A .
                 For the system  A*X = B , relative perturbations
                 in  A  and  B  of size  EPSILON  may cause
                 relative perturbations in  X  of size  EPSILON/RCOND .
                 If  RCOND  is so small that the logical expression
                           1.0 + RCOND .EQ. 1.0
                 is true, then  A  may be singular to working
                 precision.  In particular,  RCOND  is zero  if
```

```
                exact singularity is detected or the estimate
                underflows.

        Z       REAL(N)
                a work vector whose contents are usually unimportant.
                If  A  is close to a singular matrix, then  Z  is
                an approximate null vector in the sense that
                NORM(A*Z) = RCOND*NORM(A)*NORM(Z) .

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SASUM, SAXPY, SDOT, SSCAL, SSIFA
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   891107  Modified routine equivalence list.  (WRB)
   891107  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SSICS

```
      SUBROUTINE SSICS( N, NELT, IA, JA, A, ISYM, NEL, IEL, JEL, EL, D,
     +   R, IWARN)
***BEGIN PROLOGUE  SSICS
***PURPOSE  Incompl. Cholesky Decomposition Preconditioner SLAP Set Up.
            Routine to generate the Incomplete Cholesky decomposition,
            L*D*L-trans, of a symmetric positive definite matrix, A,
            which is stored in SLAP Column format.  The unit lower
            triangular matrix L is stored by rows, and the inverse of
            the diagonal matrix D is stored.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2E
***TYPE      SINGLE PRECISION (SSICS-S, DSICS-D)
***KEYWORDS  INCOMPLETE CHOLESKY FACTORIZATION,
             ITERATIVE PRECONDITION, LINEAR SYSTEM, SLAP SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM
     INTEGER NEL, IEL(NEL), JEL(NEL), IWARN
     REAL    A(NELT), EL(NEL), D(N), R(N)

     CALL SSICS( N, NELT, IA, JA, A, ISYM, NEL, IEL, JEL, EL, D, R,
    $    IWARN )

 *Arguments:
 N       :IN       Integer.
          Order of the Matrix.
 NELT    :IN       Integer.
          Number of elements in arrays IA, JA, and A.
 IA      :INOUT    Integer IA(NELT).
 JA      :INOUT    Integer JA(NELT).
 A       :INOUT    Real A(NELT).
          These arrays should hold the matrix A in the SLAP Column
          format.  See "Description", below.
 ISYM    :IN       Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the lower
          triangle of the matrix is stored.
 NEL     :OUT      Integer.
          Number of non-zeros in the lower triangle of A.   Also
          corresponds to the length of the IEL, JEL, EL arrays.
 IEL     :OUT      Integer IEL(NEL).
 JEL     :OUT      Integer JEL(NEL).
 EL      :OUT      Real     EL(NEL).
          IEL, JEL, EL contain the unit lower triangular factor  of the
          incomplete decomposition   of the A  matrix  stored  in  SLAP
          Row format.   The Diagonal of   ones   *IS*   stored.     See
          "Description", below for more details about the SLAP Row fmt.
 D       :OUT      Real D(N)
```

```
        Upon return this array holds D(I) = 1./DIAG(A).
R      :WORK     Real R(N).
        Temporary real workspace needed for the factorization.
IWARN  :OUT      Integer.
        This is a warning variable and is zero if the IC factoriza-
        tion goes well.  It is set to the row index corresponding to
        the last zero pivot found.  See "Description", below.

*Description
      ================== S L A P Column format ==================
      This routine  requires that  the matrix A  be stored in  the
      SLAP Column format.  In this format the non-zeros are stored
      counting down columns (except for  the diagonal entry, which
      must appear first in each  "column")  and are stored  in the
      real array A.  In other words, for each column in the matrix
      put the diagonal entry in A.  Then put in the other non-zero
      elements going down   the  column (except  the diagonal)  in
      order.  The IA array holds the row  index for each non-zero.
      The JA array holds the offsets into the IA, A arrays for the
      beginning of   each    column.    That  is,    IA(JA(ICOL)),
      A(JA(ICOL)) points to the beginning of the ICOL-th column in
      IA and  A.  IA(JA(ICOL+1)-1),  A(JA(ICOL+1)-1) points to the
      end  of   the ICOL-th  column.  Note   that  we  always have
      JA(N+1) = NELT+1, where  N  is the number of columns in  the
      matrix and  NELT   is the number of non-zeros in the matrix.

      Here is an example of the  SLAP Column  storage format for a
      5x5 Matrix (in the A and IA arrays '|'  denotes the end of a
      column):

          5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                                1  2  3    4  5    6  7    8    9 10 11
      |11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
      |21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
      | 0  0 33  0 35|  JA:  1  4  6    8  9   12
      | 0  0  0 44  0|
      |51  0 53  0 55|

      ==================== S L A P Row format ====================

      This routine requires  that the matrix A  be  stored  in the
      SLAP  Row format.   In this format  the non-zeros are stored
      counting across  rows (except for the diagonal  entry, which
      must appear first in each "row") and  are stored in the real
      array A.  In other words, for each row in the matrix put the
      diagonal entry in  A.   Then   put  in the   other  non-zero
      elements   going  across the  row (except   the diagonal) in
      order.   The  JA array  holds   the column   index for  each
      non-zero.   The IA  array holds the  offsets into  the JA, A
      arrays  for   the   beginning  of   each  row.   That    is,
      JA(IA(IROW)),  A(IA(IROW)) points  to  the beginning  of the
      IROW-th row in JA and A.   JA(IA(IROW+1)-1), A(IA(IROW+1)-1)
      points to the  end of the  IROW-th row.  Note that we always
      have IA(N+1) =  NELT+1, where  N  is  the number of rows  in
      the matrix  and NELT  is the  number   of  non-zeros in  the
      matrix.

      Here is an example of the SLAP Row storage format for a  5x5
      Matrix (in the A and JA arrays '|' denotes the end of a row):
```

```
    5x5 Matrix         SLAP Row format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 12 15 | 22 21 | 33 35 | 44 | 55 51 53
|21 22  0  0  0|  JA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  IA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

With the SLAP format some of the "inner loops" of this routine should vectorize on machines with hardware support for vector gather/scatter operations. Your compiler may require a compiler directive to convince it that there are no implicit vector dependencies. Compiler directives for the Alliant FX/Fortran and CRI CFT/CFT77 compilers are supplied with the standard SLAP distribution.

The IC factorization does not always exist for SPD matrices. In the event that a zero pivot is found it is set to be 1.0 and the factorization proceeds. The integer variable IWARN is set to the last row where the Diagonal was fudged. This eventuality hardly ever occurs in practice.

***SEE ALSO SCG, SSICCG

***REFERENCES 1. Gene Golub and Charles Van Loan, Matrix Computations, Johns Hopkins University Press, Baltimore, Maryland, 1983.

***ROUTINES CALLED XERMSG

***REVISION HISTORY (YYMMDD)

```
871119  DATE WRITTEN
881213  Previous REVISION DATE
890915  Made changes requested at July 1989 CML Meeting.  (MKS)
890922  Numerous changes to prologue to make closer to SLATEC
        standard.  (FNF)
890929  Numerous changes to reduce SP/DP differences.  (FNF)
900805  Changed XERRWV calls to calls to XERMSG.  (RWC)
910411  Prologue converted to Version 4.0 format.  (BAB)
920511  Added complete declaration section.  (WRB)
920929  Corrected format of reference.  (FNF)
930701  Updated CATEGORY section.  (FNF, WRB)
END PROLOGUE
```

SSIDI

```
      SUBROUTINE SSIDI (A, LDA, N, KPVT, DET, INERT, WORK, JOB)
***BEGIN PROLOGUE  SSIDI
***PURPOSE  Compute the determinant, inertia and inverse of a real
            symmetric matrix using the factors from SSIFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1A, D3B1A
***TYPE      SINGLE PRECISION (SSIDI-S, DSIDI-D, CHIDI-C, CSIDI-C)
***KEYWORDS  DETERMINANT, INVERSE, LINEAR ALGEBRA, LINPACK, MATRIX,
             SYMMETRIC
***AUTHOR  Bunch, J., (UCSD)
***DESCRIPTION

      SSIDI computes the determinant, inertia and inverse
      of a real symmetric matrix using the factors from SSIFA.

      On Entry

         A       REAL(LDA,N)
                 the output from SSIFA.

         LDA     INTEGER
                 the leading dimension of the array A.

         N       INTEGER
                 the order of the matrix A.

         KPVT    INTEGER(N)
                 the pivot vector from SSIFA.

         WORK    REAL(N)
                 work vector.  Contents destroyed.

         JOB     INTEGER
                 JOB has the decimal expansion  ABC  where
                    If  C .NE. 0, the inverse is computed,
                    If  B .NE. 0, the determinant is computed,
                    If  A .NE. 0, the inertia is computed.

                 For example, JOB = 111  gives all three.

      On Return

         Variables not requested by JOB are not used.

         A      contains the upper triangle of the inverse of
                the original matrix.  The strict lower triangle
                is never referenced.

         DET    REAL(2)
                determinant of original matrix.
                Determinant = DET(1) * 10.0**DET(2)
                with 1.0 .LE. ABS(DET(1)) .LT. 10.0
                or DET(1) = 0.0.

         INERT  INTEGER(3)
                the inertia of the original matrix.
```

```
                INERT(1)  =  number of positive eigenvalues.
                INERT(2)  =  number of negative eigenvalues.
                INERT(3)  =  number of zero eigenvalues.

     Error Condition

        A division by zero may occur if the inverse is requested
        and  SSICO  has set RCOND .EQ. 0.0
        or  SSIFA  has set  INFO .NE. 0 .

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SCOPY, SDOT, SSWAP
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   891107  Modified routine equivalence list.  (WRB)
   891107  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SSIEV

```
      SUBROUTINE SSIEV (A, LDA, N, E, WORK, JOB, INFO)
***BEGIN PROLOGUE  SSIEV
***PURPOSE  Compute the eigenvalues and, optionally, the eigenvectors
            of a real symmetric matrix.
***LIBRARY   SLATEC
***CATEGORY  D4A1
***TYPE      SINGLE PRECISION (SSIEV-S, CHIEV-C)
***KEYWORDS  COMPLEX HERMITIAN, EIGENVALUES, EIGENVECTORS, MATRIX,
             SYMMETRIC
***AUTHOR  Kahaner, D. K., (NBS)
           Moler, C. B., (U. of New Mexico)
           Stewart, G. W., (U. of Maryland)
***DESCRIPTION

      Abstract
       SSIEV computes the eigenvalues and, optionally, the eigenvectors
       of a real symmetric matrix.

      Call Sequence Parameters-
        (The values of parameters marked with * (star) will be  changed
          by SSIEV.)

        A*      REAL (LDA,N)
                real symmetric input matrix.
                Only the diagonal and upper triangle of A must be input,
                as SSIEV copies the upper triangle to the lower.
                That is, the user must define A(I,J), I=1,..N, and J=I,.
                .,N.
                On return from SSIEV, if the user has set JOB
                = 0        the lower triangle of A has been altered.
                = nonzero  the N eigenvectors of A are stored in its
                first N columns.  See also INFO below.

        LDA     INTEGER
                set by the user to
                the leading dimension of the array A.

        N       INTEGER
                set by the user to
                the order of the matrix A and
                the number of elements in E.

        E*      REAL (N)
                on return from SSIEV, E contains the N
                eigenvalues of A.  See also INFO below.

        WORK*   REAL (2*N)
                temporary storage vector.  Contents changed by SSIEV.

        JOB     INTEGER
                set by user on input
                = 0         only calculate eigenvalues of A.
                = nonzero   calculate eigenvalues and eigenvectors of A.

        INFO*   INTEGER
                on return from SSIEV, the value of INFO is
```

```
                = 0 for normal return.
                = K if the eigenvalue iteration fails to converge.
                    eigenvalues and vectors 1 through K-1 are correct.


     Error Messages-
          No. 1   recoverable  N is greater than LDA
          No. 2   recoverable  N is less than one

***REFERENCES  (NONE)
***ROUTINES CALLED  IMTQL2, TQLRAT, TRED1, TRED2, XERMSG
***REVISION HISTORY  (YYMMDD)
   800808  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   END PROLOGUE
```

SSIFA

```
      SUBROUTINE SSIFA (A, LDA, N, KPVT, INFO)
***BEGIN PROLOGUE  SSIFA
***PURPOSE  Factor a real symmetric matrix by elimination with
            symmetric pivoting.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1A
***TYPE      SINGLE PRECISION (SSIFA-S, DSIFA-D, CHIFA-C, CSIFA-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX FACTORIZATION, SYMMETRIC
***AUTHOR  Bunch, J., (UCSD)
***DESCRIPTION

      SSIFA factors a real symmetric matrix by elimination
      with symmetric pivoting.

      To solve  A*X = B , follow SSIFA by SSISL.
      To compute  INVERSE(A)*C , follow SSIFA by SSISL.
      To compute  DETERMINANT(A) , follow SSIFA by SSIDI.
      To compute  INERTIA(A) , follow SSIFA by SSIDI.
      To compute  INVERSE(A) , follow SSIFA by SSIDI.

      On Entry

         A       REAL(LDA,N)
                 the symmetric matrix to be factored.
                 Only the diagonal and upper triangle are used.

         LDA     INTEGER
                 the leading dimension of the array  A .

         N       INTEGER
                 the order of the matrix  A .

      On Return

         A       a block diagonal matrix and the multipliers which
                 were used to obtain it.
                 The factorization can be written  A = U*D*TRANS(U)
                 where  U  is a product of permutation and unit
                 upper triangular matrices , TRANS(U) is the
                 transpose of  U , and  D  is block diagonal
                 with 1 by 1 and 2 by 2 blocks.

         KPVT    INTEGER(N)
                 an integer vector of pivot indices.

         INFO    INTEGER
                 = 0  normal value.
                 = K  if the K-th pivot block is singular.  This is
                      not an error condition for this subroutine,
                      but it does indicate that SSISL or SSIDI may
                      divide by zero if called.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  ISAMAX, SAXPY, SSWAP
***REVISION HISTORY  (YYMMDD)
```

```
780814  DATE WRITTEN
890531  Changed all specific intrinsics to generic.  (WRB)
890831  Modified array declarations.  (WRB)
891107  Modified routine equivalence list.  (WRB)
891107  REVISION DATE from Version 3.2
891214  Prologue converted to Version 4.0 format.  (BAB)
900326  Removed duplicate information from DESCRIPTION section.
        (WRB)
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

SSILUR

```
      SUBROUTINE SSILUR(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
     +   ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)
***BEGIN PROLOGUE  SSILUR
***PURPOSE  Incomplete LU Iterative Refinement Sparse Ax = b Solver.
            Routine to solve a general linear system  Ax = b  using
            the incomplete LU decomposition with iterative refinement.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SSILUR-S, DSILUR-D)
***KEYWORDS  ITERATIVE PRECONDITION, LINEAR SYSTEM, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
     INTEGER ITER, IERR, IUNIT, LENW, IWORK(NL+NU+4*N+2), LENIW
     REAL B(N), X(N), A(NELT), TOL, ERR, RWORK(NL+NU+4*N)

     CALL SSILUR(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
    $     ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)

 *Arguments:
 N       :IN       Integer
          Order of the Matrix.
 B       :IN       Real B(N).
          Right-hand side vector.
 X       :INOUT    Real X(N).
          On input X is your initial guess for solution vector.
          On output X is the final approximate solution.
 NELT    :IN       Integer.
          Number of Non-Zeros stored in A.
 IA      :INOUT    Integer IA(NELT).
 JA      :INOUT    Integer JA(NELT).
 A       :INOUT    Real A(NELT).
          These arrays should hold the matrix A in either the SLAP
          Triad format or the SLAP Column format.  See "Description",
          below.  If the SLAP Triad format is chosen it is changed
          internally to the SLAP Column format.
 ISYM    :IN       Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the upper
          or lower triangle of the matrix is stored.
 ITOL    :IN       Integer.
          Flag to indicate type of convergence criterion.
          If ITOL=1, iteration stops when the 2-norm of the residual
          divided by the 2-norm of the right-hand side is less than TOL.
          If ITOL=2, iteration stops when the 2-norm of M-inv times the
          residual divided by the 2-norm of M-inv times the right hand
          side is less than TOL, where M-inv is the inverse of the
          diagonal of A.
```

```
         ITOL=11 is often useful for checking and comparing different
         routines.  For this case, the user must supply the "exact"
         solution or a very accurate approximation (one with an error
         much less than TOL) through a common block,
             COMMON /SSLBLK/ SOLN( )
         If ITOL=11, iteration stops when the 2-norm of the difference
         between the iterative approximation and the user-supplied
         solution divided by the 2-norm of the user-supplied solution
         is less than TOL.  Note that this requires the user to set up
         the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine.
         The routine with this declaration should be loaded before the
         stop test so that the correct length is used by the loader.
         This procedure is not standard Fortran and may not work
         correctly on your system (although it has worked on every
         system the authors have tried).  If ITOL is not 11 then this
         common block is indeed standard Fortran.
TOL     :INOUT    Real.
         Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX   :IN       Integer.
         Maximum number of iterations.
ITER    :OUT      Integer.
         Number of iterations required to reach convergence, or
         ITMAX+1 if convergence criterion could not be achieved in
         ITMAX iterations.
ERR     :OUT      Real.
         Error estimate of error in final approximate solution, as
         defined by ITOL.
IERR    :OUT      Integer.
         Return error flag.
           IERR = 0 => All went well.
           IERR = 1 => Insufficient space allocated for WORK or IWORK.
           IERR = 2 => Method failed to converge in ITMAX steps.
           IERR = 3 => Error in user input.
                       Check input values of N, ITOL.
           IERR = 4 => User error tolerance set too tight.
                       Reset to 500*R1MACH(3).  Iteration proceeded.
           IERR = 5 => Preconditioning matrix, M, is not positive
                       definite.  (r,z) < 0.
           IERR = 6 => Matrix A is not positive definite.  (p,Ap) < 0.
           IERR = 7 => Incomplete factorization broke down and was
                       fudged.  Resulting preconditioning may be less
                       than the best.
IUNIT   :IN       Integer.
         Unit number on which to write the error at each iteration,
         if this is desired for monitoring convergence.  If unit
         number is 0, no writing will occur.
RWORK   :WORK     Real RWORK(LENW).
         Real array used for workspace.
LENW    :IN       Integer.
         Length of the real workspace, RWORK.  LENW >= NL+NU+4*N.
         NL is the number of non-zeros in the lower triangle of the
         matrix (including the diagonal).
         NU is the number of non-zeros in the upper triangle of the
         matrix (including the diagonal).
IWORK   :WORK     Integer IWORK(LENIW).
         Integer array used for workspace.
         Upon return the following locations of IWORK hold information
         which may be of use to the user:
         IWORK(9)  Amount of Integer workspace actually used.
         IWORK(10) Amount of Real workspace actually used.
```

LENIW :IN Integer.
Length of integer workspace, IWORK. LENIW >= NL+NU+4*N+10.
NL is the number of non-zeros in the lower triangle of the matrix (including the diagonal).
NU is the number of non-zeros in the upper triangle of the matrix (including the diagonal).

*Description

The Sparse Linear Algebra Package (SLAP) utilizes two matrix data structures: 1) the SLAP Triad format or 2) the SLAP Column format. The user can hand this routine either of the of these data structures and SLAP will figure out which on is being used and act accordingly.

========== S L A P Triad format ==========

This routine requires that the matrix A be stored in the SLAP Triad format. In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Side Effects:
The SLAP Triad format (IA, JA, A) is modified internally to be the SLAP Column format. See above.

*Cautions:
This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO SSJAC, SSGS, SIR
***REFERENCES (NONE)
***ROUTINES CALLED SCHKW, SIR, SS2Y, SSILUS, SSLUI, SSMV
***REVISION HISTORY (YYMMDD)
871119 DATE WRITTEN
881213 Previous REVISION DATE
890915 Made changes requested at July 1989 CML Meeting. (MKS)
890921 Removed TeX from comments. (FNF)
890922 Numerous changes to prologue to make closer to SLATEC standard. (FNF)
890929 Numerous changes to reduce SP/DP differences. (FNF)
910411 Prologue converted to Version 4.0 format. (BAB)
920407 COMMON BLOCK renamed SSLBLK. (WRB)
920511 Added complete declaration section. (WRB)
921019 Corrected NEL to NL. (FNF)
END PROLOGUE

SSILUS

```
      SUBROUTINE SSILUS( N, NELT, IA, JA, A, ISYM, NL, IL, JL, L, DINV,
     +   NU, IU, JU, U, NROW, NCOL)
***BEGIN PROLOGUE  SSILUS
***PURPOSE  Incomplete LU Decomposition Preconditioner SLAP Set Up.
            Routine to generate the incomplete LDU decomposition of a
            matrix.  The unit lower triangular factor L is stored by
            rows and the unit upper triangular factor U is stored by
            columns.  The inverse of the diagonal matrix D is stored.
            No fill in is allowed.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2E
***TYPE      SINGLE PRECISION (SSILUS-S, DSILUS-D)
***KEYWORDS  INCOMPLETE LU FACTORIZATION, ITERATIVE PRECONDITION,
             NON-SYMMETRIC LINEAR SYSTEM, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM
     INTEGER NL, IL(NL), JL(NL), NU, IU(NU), JU(NU)
     INTEGER NROW(N), NCOL(N)
     REAL    A(NELT), L(NL), DINV(N), U(NU)

     CALL SSILUS( N, NELT, IA, JA, A, ISYM, NL, IL, JL, L,
    $    DINV, NU, IU, JU, U, NROW, NCOL )

 *Arguments:
 N       :IN       Integer
          Order of the Matrix.
 NELT    :IN       Integer.
          Number of elements in arrays IA, JA, and A.
 IA      :IN       Integer IA(NELT).
 JA      :IN       Integer JA(NELT).
 A       :IN       Real A(NELT).
          These arrays should hold the matrix A in the SLAP Column
          format.  See "Description", below.
 ISYM    :IN       Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the lower
          triangle of the matrix is stored.
 NL      :OUT      Integer.
          Number of non-zeros in the L array.
 IL      :OUT      Integer IL(NL).
 JL      :OUT      Integer JL(NL).
 L       :OUT      Real     L(NL).
          IL, JL, L  contain the unit lower triangular factor of  the
          incomplete decomposition  of some  matrix stored  in   SLAP
          Row format.     The   Diagonal  of ones  *IS*  stored.  See
          "DESCRIPTION", below for more details about the SLAP format.
 NU      :OUT      Integer.
```

```
        Number of non-zeros in the U array.
IU     :OUT      Integer IU(NU).
JU     :OUT      Integer JU(NU).
U      :OUT      Real     U(NU).
        IU, JU, U contain   the unit upper triangular factor of the
        incomplete  decomposition    of some matrix  stored in SLAP
        Column  format.   The Diagonal of ones   *IS*  stored.  See
        "Description", below  for  more  details  about  the   SLAP
        format.
NROW   :WORK     Integer NROW(N).
        NROW(I) is the number of non-zero elements in the I-th row
        of L.
NCOL   :WORK     Integer NCOL(N).
        NCOL(I) is the number of non-zero elements in the I-th
        column of U.

*Description
      IL, JL, L should contain the unit  lower triangular factor of
      the incomplete decomposition of the A matrix  stored in SLAP
      Row format.  IU, JU, U should contain  the unit upper factor
      of the  incomplete decomposition of  the A matrix  stored in
      SLAP Column format This ILU factorization can be computed by
      the SSILUS routine. The diagonals (which are all one's) are
      stored.

      ================== S L A P Column format ==================

      This routine  requires that  the matrix A  be stored in  the
      SLAP Column format.  In this format the non-zeros are stored
      counting down columns (except for  the diagonal entry, which
      must appear first in each  "column")  and are stored  in the
      real array A.  In other words, for each column in the matrix
      put the diagonal entry in A.  Then put in the other non-zero
      elements going down   the  column (except  the diagonal)  in
      order.  The IA array holds the row  index for each non-zero.
      The JA array holds the offsets into the IA, A arrays for the
      beginning of   each    column.    That  is,    IA(JA(ICOL)),
      A(JA(ICOL)) points to the beginning of the ICOL-th column in
      IA and  A.  IA(JA(ICOL+1)-1),  A(JA(ICOL+1)-1) points to the
      end  of   the ICOL-th  column.  Note   that  we  always have
      JA(N+1) = NELT+1, where  N  is the number of columns in  the
      matrix and  NELT   is the number of non-zeros in the matrix.

      Here is an example of the  SLAP Column  storage format for a
      5x5 Matrix (in the A and IA arrays '|'  denotes the end of a
      column):

          5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                             1  2  3    4  5    6  7    8    9 10 11
      |11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
      |21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
      | 0  0 33  0 35|  JA:  1  4  6    8  9   12
      | 0  0  0 44  0|
      |51  0 53  0 55|

      ==================== S L A P Row format ====================

      This routine requires  that the matrix A  be  stored  in the
      SLAP  Row format.   In this format  the non-zeros are stored
      counting across  rows (except for the diagonal  entry, which
```

must appear first in each "row") and are stored in the real array A. In other words, for each row in the matrix put the diagonal entry in A. Then put in the other non-zero elements going across the row (except the diagonal) in order. The JA array holds the column index for each non-zero. The IA array holds the offsets into the JA, A arrays for the beginning of each row. That is, JA(IA(IROW)), A(IA(IROW)) points to the beginning of the IROW-th row in JA and A. JA(IA(IROW+1)-1), A(IA(IROW+1)-1) points to the end of the IROW-th row. Note that we always have IA(N+1) = NELT+1, where N is the number of rows in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Row storage format for a 5x5 Matrix (in the A and JA arrays '|' denotes the end of a row):

```
    5x5 Matrix         SLAP Row format for 5x5 matrix on left.
                        1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 12 15 | 22 21 | 33 35 | 44 | 55 51 53
|21 22  0  0  0|  JA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  IA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

***SEE ALSO SILUR

***REFERENCES 1. Gene Golub and Charles Van Loan, Matrix Computations, Johns Hopkins University Press, Baltimore, Maryland, 1983.

***ROUTINES CALLED (NONE)

***REVISION HISTORY (YYMMDD)

871119 DATE WRITTEN
881213 Previous REVISION DATE
890915 Made changes requested at July 1989 CML Meeting. (MKS)
890922 Numerous changes to prologue to make closer to SLATEC standard. (FNF)
890929 Numerous changes to reduce SP/DP differences. (FNF)
910411 Prologue converted to Version 4.0 format. (BAB)
920511 Added complete declaration section. (WRB)
920929 Corrected format of reference. (FNF)
930701 Updated CATEGORY section. (FNF, WRB)
END PROLOGUE

SSISL

```
      SUBROUTINE SSISL (A, LDA, N, KPVT, B)
***BEGIN PROLOGUE  SSISL
***PURPOSE  Solve a real symmetric system using the factors obtained
            from SSIFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1A
***TYPE      SINGLE PRECISION (SSISL-S, DSISL-D, CHISL-C, CSISL-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX, SOLVE, SYMMETRIC
***AUTHOR  Bunch, J., (UCSD)
***DESCRIPTION

      SSISL solves the real symmetric system
      A * X = B
      using the factors computed by SSIFA.

      On Entry

         A       REAL(LDA,N)
                 the output from SSIFA.

         LDA     INTEGER
                 the leading dimension of the array  A .

         N       INTEGER
                 the order of the matrix  A .

         KPVT    INTEGER(N)
                 the pivot vector from SSIFA.

         B       REAL(N)
                 the right hand side vector.

      On Return

         B       the solution vector  X .

      Error Condition

         A division by zero may occur if  SSICO  has set RCOND .EQ. 0.0
         or  SSIFA  has set INFO .NE. 0  .

      To compute  INVERSE(A) * C  where  C  is a matrix
      with  P  columns
            CALL SSIFA(A,LDA,N,KPVT,INFO)
            IF (INFO .NE. 0) GO TO ...
            DO 10 J = 1, P
               CALL SSISL(A,LDA,N,KPVT,C(1,J))
         10 CONTINUE

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SDOT
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
```

```
891107  Modified routine equivalence list.  (WRB)
891107  REVISION DATE from Version 3.2
891214  Prologue converted to Version 4.0 format.  (BAB)
900326  Removed duplicate information from DESCRIPTION section.
        (WRB)
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

SSJAC

```
      SUBROUTINE SSJAC(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
     +   ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)
***BEGIN PROLOGUE  SSJAC
***PURPOSE  Jacobi's Method Iterative Sparse Ax = b Solver.
            Routine to solve a general linear system  Ax = b  using
            Jacobi iteration.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SSJAC-S, DSJAC-D)
***KEYWORDS  ITERATIVE PRECONDITION, LINEAR SYSTEM, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
      INTEGER  N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
      INTEGER  ITER, IERR, IUNIT, LENW, IWORK(LENIW), LENIW
      REAL     B(N), X(N), A(NELT), TOL, ERR, RWORK(LENW)

      CALL SSJAC(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
     $     ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW )

 *Arguments:
 N       :IN       Integer.
          Order of the Matrix.
 B       :IN       Real B(N).
          Right-hand side vector.
 X       :INOUT    Real X(N).
          On input X is your initial guess for solution vector.
          On output X is the final approximate solution.
 NELT    :IN       Integer.
          Number of Non-Zeros stored in A.
 IA      :INOUT    Integer IA(NELT).
 JA      :INOUT    Integer JA(NELT).
 A       :INOUT    Real A(NELT).
          These arrays should hold the matrix A in either the SLAP
          Triad format or the SLAP Column format.  See "Description",
          below.  If the SLAP Triad format is chosen it is changed
          internally to the SLAP Column format.
 ISYM    :IN       Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the upper
          or lower triangle of the matrix is stored.
 ITOL    :IN       Integer.
          Flag to indicate type of convergence criterion.
          If ITOL=1, iteration stops when the 2-norm of the residual
          divided by the 2-norm of the right-hand side is less than TOL.
          If ITOL=2, iteration stops when the 2-norm of M-inv times the
          residual divided by the 2-norm of M-inv times the right hand
          side is less than TOL, where M-inv is the inverse of the
          diagonal of A.
```

```
         ITOL=11 is often useful for checking and comparing different
         routines.  For this case, the user must supply the "exact"
         solution or a very accurate approximation (one with an error
         much less than TOL) through a common block,
             COMMON /SSLBLK/ SOLN( )
         If ITOL=11, iteration stops when the 2-norm of the difference
         between the iterative approximation and the user-supplied
         solution divided by the 2-norm of the user-supplied solution
         is less than TOL.  Note that this requires the user to set up
         the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine.
         The routine with this declaration should be loaded before the
         stop test so that the correct length is used by the loader.
         This procedure is not standard Fortran and may not work
         correctly on your system (although it has worked on every
         system the authors have tried).  If ITOL is not 11 then this
         common block is indeed standard Fortran.
TOL     :INOUT    Real.
         Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX   :IN       Integer.
         Maximum number of iterations.
ITER    :OUT      Integer.
         Number of iterations required to reach convergence, or
         ITMAX+1 if convergence criterion could not be achieved in
         ITMAX iterations.
ERR     :OUT      Real.
         Error estimate of error in final approximate solution, as
         defined by ITOL.
IERR    :OUT      Integer.
         Return error flag.
           IERR = 0 => All went well.
           IERR = 1 => Insufficient space allocated for WORK or IWORK.
           IERR = 2 => Method failed to converge in ITMAX steps.
           IERR = 3 => Error in user input.
                       Check input values of N, ITOL.
           IERR = 4 => User error tolerance set too tight.
                       Reset to 500*R1MACH(3).  Iteration proceeded.
           IERR = 5 => Preconditioning matrix, M, is not positive
                       definite.  (r,z) < 0.
           IERR = 6 => Matrix A is not positive definite.  (p,Ap) < 0.
IUNIT   :IN       Integer.
         Unit number on which to write the error at each iteration,
         if this is desired for monitoring convergence.  If unit
         number is 0, no writing will occur.
RWORK   :WORK     Real RWORK(LENW).
         Real array used for workspace.
LENW    :IN       Integer.
         Length of the real workspace, RWORK.  LENW >= 4*N.
IWORK   :WORK     Integer IWORK(LENIW).
         Used to hold pointers into the real workspace, RWORK.
         Upon return the following locations of IWORK hold information
         which may be of use to the user:
         IWORK(9)  Amount of Integer workspace actually used.
         IWORK(10) Amount of Real workspace actually used.
LENIW   :IN       Integer.
         Length of the integer workspace, IWORK.  LENIW >= 10.

*Description:
      Jacobi's method solves the linear system Ax=b with the
      basic iterative method (where A = L + D + U):
```

$$X^{n+1} = D^{-1} (B - LX^{n} - UX^{n})$$

$$= X^{n} + D^{-1} (B - AX^{n})$$

The Sparse Linear Algebra Package (SLAP) utilizes two matrix data structures: 1) the SLAP Triad format or 2) the SLAP Column format. The user can hand this routine either of the of these data structures and SLAP will figure out which one is being used and act accordingly.

========== S L A P Triad format ==========

This routine requires that the matrix A be stored in the SLAP Triad format. In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a

column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Side Effects:
The SLAP Triad format (IA, JA, A) is modified internally to be the SLAP Column format. See above.

*Cautions:
This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO SSGS, SIR
***REFERENCES (NONE)
***ROUTINES CALLED SCHKW, SIR, SS2Y, SSDI, SSDS, SSMV
***REVISION HISTORY (YYMMDD)

```
871119  DATE WRITTEN
881213  Previous REVISION DATE
890915  Made changes requested at July 1989 CML Meeting.  (MKS)
890921  Removed TeX from comments.  (FNF)
890922  Numerous changes to prologue to make closer to SLATEC
        standard.  (FNF)
890929  Numerous changes to reduce SP/DP differences.  (FNF)
910411  Prologue converted to Version 4.0 format.  (BAB)
910506  Corrected error in C***ROUTINES CALLED list.  (FNF)
920407  COMMON BLOCK renamed SSLBLK.  (WRB)
920511  Added complete declaration section.  (WRB)
END PROLOGUE
```

SSLI

```
      SUBROUTINE SSLI (N, B, X, NELT, IA, JA, A, ISYM, RWORK, IWORK)
***BEGIN PROLOGUE  SSLI
***PURPOSE  SLAP MSOLVE for Lower Triangle Matrix.
            This routine acts as an interface between the SLAP generic
            MSOLVE calling convention and the routine that actually
                      -1
            computes L  B = X.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A3
***TYPE      SINGLE PRECISION (SSLI-S, DSLI-D)
***KEYWORDS  ITERATIVE PRECONDITION, LINEAR SYSTEM SOLVE, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION
       It is assumed that RWORK and IWORK have initialized with
       the information required for SSLI2:
          IWORK(1) = NEL
          IWORK(2) = Starting location of IEL in IWORK.
          IWORK(3) = Starting location of JEL in IWORK.
          IWORK(4) = Starting location of EL in RWORK.
       See the DESCRIPTION of SSLI2 for details.
***REFERENCES  (NONE)
***ROUTINES CALLED  SSLI2
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920511  Added complete declaration section.  (WRB)
   921113  Corrected C***CATEGORY line.  (FNF)
   930701  Updated CATEGORY section.  (FNF, WRB)
   END PROLOGUE
```

SSLI2

```
      SUBROUTINE SSLI2 (N, B, X, NEL, IEL, JEL, EL)
***BEGIN PROLOGUE  SSLI2
***PURPOSE  SLAP Lower Triangle Matrix Backsolve.
            Routine to solve a system of the form  Lx = b , where L
            is a lower triangular matrix.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A3
***TYPE      SINGLE PRECISION (SSLI2-S, DSLI2-D)
***KEYWORDS  ITERATIVE PRECONDITION, LINEAR SYSTEM SOLVE, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NEL, IEL(NEL), JEL(NEL)
     REAL    B(N), X(N), EL(NEL)

     CALL SSLI2( N, B, X, NEL, IEL, JEL, EL )

 *Arguments:
 N      :IN       Integer
         Order of the Matrix.
 B      :IN       Real B(N).
         Right hand side vector.
 X      :OUT      Real X(N).
         Solution to Lx = b.
 NEL    :IN       Integer.
         Number of non-zeros in the EL array.
 IEL    :IN       Integer IEL(NEL).
 JEL    :IN       Integer JEL(NEL).
 EL     :IN       Real     EL(NEL).
         IEL, JEL, EL contain the unit lower triangular factor   of
         the incomplete decomposition   of the A  matrix  stored in
         SLAP Row format.  The diagonal of  ones *IS* stored.  This
         structure can be  set up by the  SS2LT  routine.  See  the
         "Description", below, for more details about the  SLAP Row
         format.

 *Description:
       This routine is supplied with the SLAP package  as a routine
       to perform the MSOLVE operation in the SIR iteration routine
       for the driver routine SSGS.  It must be called via the SLAP
       MSOLVE calling sequence convention interface routine SSLI.
         **** THIS ROUTINE ITSELF DOES NOT CONFORM TO THE ****
               **** SLAP MSOLVE CALLING CONVENTION ****

       ==================== S L A P Row format ====================

       This routine requires  that the matrix A  be  stored  in the
       SLAP  Row format.   In this format  the non-zeros are stored
       counting across  rows (except for the diagonal  entry, which
       must appear first in each "row") and  are stored in the real
```

array A. In other words, for each row in the matrix put the diagonal entry in A. Then put in the other non-zero elements going across the row (except the diagonal) in order. The JA array holds the column index for each non-zero. The IA array holds the offsets into the JA, A arrays for the beginning of each row. That is, JA(IA(IROW)), A(IA(IROW)) points to the beginning of the IROW-th row in JA and A. JA(IA(IROW+1)-1), A(IA(IROW+1)-1) points to the end of the IROW-th row. Note that we always have IA(N+1) = NELT+1, where N is the number of rows in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Row storage format for a 5x5 Matrix (in the A and JA arrays '|' denotes the end of a row):

```
    5x5 Matrix         SLAP Row format for 5x5 matrix on left.
                        1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 12 15 | 22 21 | 33 35 | 44 | 55 51 53
|21 22  0  0  0|  JA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  IA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

With the SLAP Row format the "inner loop" of this routine should vectorize on machines with hardware support for vector gather/scatter operations. Your compiler may require a compiler directive to convince it that there are no implicit vector dependencies. Compiler directives for the Alliant FX/Fortran and CRI CFT/CFT77 compilers are supplied with the standard SLAP distribution.

***SEE ALSO SSLI
***REFERENCES (NONE)
***ROUTINES CALLED (NONE)
***REVISION HISTORY (YYMMDD)
871119 DATE WRITTEN
881213 Previous REVISION DATE
890915 Made changes requested at July 1989 CML Meeting. (MKS)
890922 Numerous changes to prologue to make closer to SLATEC standard. (FNF)
890929 Numerous changes to reduce SP/DP differences. (FNF)
910411 Prologue converted to Version 4.0 format. (BAB)
920511 Added complete declaration section. (WRB)
921113 Corrected C***CATEGORY line. (FNF)
930701 Updated CATEGORY section. (FNF, WRB)
END PROLOGUE

SSLLTI

```
      SUBROUTINE SSLLTI (N, B, X, NELT, IA, JA, A, ISYM, RWORK, IWORK)
***BEGIN PROLOGUE  SSLLTI
***PURPOSE  SLAP MSOLVE for LDL' (IC) Factorization.
            This routine acts as an interface between the SLAP generic
            MSOLVE calling convention and the routine that actually
                                 -1
            computes (LDL')  B = X.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2E
***TYPE      SINGLE PRECISION (SSLLTI-S, DSLLTI-D)
***KEYWORDS  ITERATIVE PRECONDITION, LINEAR SYSTEM SOLVE, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION
       It is assumed that RWORK and IWORK have initialized with
       the information required for SLLTI2:
          IWORK(1) = NEL
          IWORK(2) = Starting location of IEL in IWORK.
          IWORK(3) = Starting location of JEL in IWORK.
          IWORK(4) = Starting location of EL in RWORK.
          IWORK(5) = Starting location of DINV in RWORK.
       See the DESCRIPTION of SLLTI2 for details.
***REFERENCES  (NONE)
***ROUTINES CALLED  SLLTI2
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   910502  Corrected conversion error.  (FNF)
   920511  Added complete declaration section.  (WRB)
   921113  Corrected C***CATEGORY line.  (FNF)
   930701  Updated CATEGORY section.  (FNF, WRB)
   END PROLOGUE
```

SSLUBC

```
      SUBROUTINE SSLUBC(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
     +   ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)
***BEGIN PROLOGUE  SSLUBC
***PURPOSE  Incomplete LU BiConjugate Gradient Sparse Ax=b Solver.
            Routine to solve a linear system  Ax = b  using the
            BiConjugate Gradient method with Incomplete LU
            decomposition preconditioning.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SSLUBC-S, DSLUBC-D)
***KEYWORDS  ITERATIVE INCOMPLETE LU PRECONDITION,
             NON-SYMMETRIC LINEAR SYSTEM, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
     INTEGER ITER, IERR, IUNIT, LENW, IWORK(NL+NU+4*N+2), LENIW
     REAL B(N), X(N), A(NELT), TOL, ERR, RWORK(NL+NU+8*N)

     CALL SSLUBC(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
    $     ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)

 *Arguments:
 N      :IN       Integer.
         Order of the Matrix.
 B      :IN       Real B(N).
         Right-hand side vector.
 X      :INOUT    Real X(N).
         On input X is your initial guess for solution vector.
         On output X is the final approximate solution.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :INOUT    Integer IA(NELT).
 JA     :INOUT    Integer JA(NELT).
 A      :INOUT    Real A(NELT).
         These arrays should hold the matrix A in either the SLAP
         Triad format or the SLAP Column format.  See "Description",
         below.  If the SLAP Triad format is chosen it is changed
         internally to the SLAP Column format.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 ITOL   :IN       Integer.
         Flag to indicate type of convergence criterion.
         If ITOL=1, iteration stops when the 2-norm of the residual
         divided by the 2-norm of the right-hand side is less than TOL.
         If ITOL=2, iteration stops when the 2-norm of M-inv times the
         residual divided by the 2-norm of M-inv times the right hand
```

```
         side is less than TOL, where M-inv is the inverse of the
         diagonal of A.
         ITOL=11 is often useful for checking and comparing different
         routines.  For this case, the user must supply the "exact"
         solution or a very accurate approximation (one with an error
         much less than TOL) through a common block,
             COMMON /SSLBLK/ SOLN( )
         If ITOL=11, iteration stops when the 2-norm of the difference
         between the iterative approximation and the user-supplied
         solution divided by the 2-norm of the user-supplied solution
         is less than TOL.  Note that this requires the user to set up
         the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine.
         The routine with this declaration should be loaded before the
         stop test so that the correct length is used by the loader.
         This procedure is not standard Fortran and may not work
         correctly on your system (although it has worked on every
         system the authors have tried).  If ITOL is not 11 then this
         common block is indeed standard Fortran.
TOL     :INOUT    Real.
         Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX   :IN       Integer.
         Maximum number of iterations.
ITER    :OUT      Integer.
         Number of iterations required to reach convergence, or
         ITMAX+1 if convergence criterion could not be achieved in
         ITMAX iterations.
ERR     :OUT      Real.
         Error estimate of error in final approximate solution, as
         defined by ITOL.
IERR    :OUT      Integer.
         Return error flag.
           IERR = 0 => All went well.
           IERR = 1 => Insufficient space allocated for WORK or IWORK.
           IERR = 2 => Method failed to converge in ITMAX steps.
           IERR = 3 => Error in user input.
                       Check input values of N, ITOL.
           IERR = 4 => User error tolerance set too tight.
                       Reset to 500*R1MACH(3).  Iteration proceeded.
           IERR = 5 => Preconditioning matrix, M, is not positive
                       definite.  (r,z) < 0.
           IERR = 6 => Matrix A is not positive definite.  (p,Ap) < 0.
           IERR = 7 => Incomplete factorization broke down and was
                       fudged.  Resulting preconditioning may be less
                       than the best.
IUNIT   :IN       Integer.
         Unit number on which to write the error at each iteration,
         if this is desired for monitoring convergence.  If unit
         number is 0, no writing will occur.
RWORK   :WORK     Real RWORK(LENW).
         Real array used for workspace.
LENW    :IN       Integer.
         Length of the real workspace, RWORK.  LENW >= NL+NU+8*N.
         NL is the number of non-zeros in the lower triangle of the
         matrix (including the diagonal).
         NU is the number of non-zeros in the upper triangle of the
         matrix (including the diagonal).
IWORK   :WORK     Integer IWORK(LENIW).
         Integer array used for workspace.
         Upon return the following locations of IWORK hold information
         which may be of use to the user:
```

```
        IWORK(9)  Amount of Integer workspace actually used.
        IWORK(10) Amount of Real workspace actually used.
LENIW  :IN       Integer.
        Length of the integer workspace, IWORK.
        LENIW >= NL+NU+4*N+12.
        NL is the number of non-zeros in the lower triangle of the
        matrix (including the diagonal).
        NU is the number of non-zeros in the upper triangle of the
        matrix (including the diagonal).

*Description:
      This routine is simply a  driver for the SBCGN  routine.  It
      calls the SSILUS routine to set  up the  preconditioning and
      then  calls SBCGN with  the appropriate   MATVEC, MTTVEC and
      MSOLVE, MTSOLV routines.

      The Sparse Linear Algebra Package (SLAP) utilizes two matrix
      data structures: 1) the  SLAP Triad  format or  2)  the SLAP
      Column format.  The user can hand this routine either of the
      of these data structures and SLAP  will figure out  which on
      is being used and act accordingly.

      ================== S L A P Triad format ==================

      This routine requires that the  matrix A be   stored in  the
      SLAP  Triad format.  In  this format only the non-zeros  are
      stored.  They may appear in  *ANY* order.  The user supplies
      three arrays of  length NELT, where  NELT is  the number  of
      non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)).  For
      each non-zero the user puts the row and column index of that
      matrix element  in the IA and  JA arrays.  The  value of the
      non-zero   matrix  element is  placed  in  the corresponding
      location of the A array.   This is  an  extremely  easy data
      structure to generate.  On  the  other hand it   is  not too
      efficient on vector computers for  the iterative solution of
      linear systems.  Hence,   SLAP changes   this  input    data
      structure to the SLAP Column format  for  the iteration (but
      does not change it back).

      Here is an example of the  SLAP Triad   storage format for a
      5x5 Matrix.  Recall that the entries may appear in any order.

          5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                                1  2  3  4  5  6  7  8  9 10 11
      |11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
      |21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
      | 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
      | 0  0  0 44  0|
      |51  0 53  0 55|

      ================== S L A P Column format ==================

      This routine  requires that  the matrix A  be stored in  the
      SLAP Column format.  In this format the non-zeros are stored
      counting down columns (except for  the diagonal entry, which
      must appear first in each  "column")  and are stored  in the
      real array A.  In other words, for each column in the matrix
      put the diagonal entry in A.  Then put in the other non-zero
      elements going down   the  column (except  the diagonal)  in
      order.  The IA array holds the row  index for each non-zero.
```

The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Side Effects:
The SLAP Triad format (IA, JA, A) is modified internally to be the SLAP Column format. See above.

*Cautions:
This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

```
***SEE ALSO  SBCG, SSDBCG
***REFERENCES  (NONE)
***ROUTINES CALLED  SBCG, SCHKW, SS2Y, SSILUS, SSLUI, SSLUTI, SSMTV,
                    SSMV
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890921  Removed TeX from comments.  (FNF)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920407  COMMON BLOCK renamed SSLBLK.  (WRB)
   920511  Added complete declaration section.  (WRB)
   921113  Corrected C***CATEGORY line.  (FNF)
   END PROLOGUE
```

SSLUCN

```
      SUBROUTINE SSLUCN(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
     +   ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)
***BEGIN PROLOGUE  SSLUCN
***PURPOSE  Incomplete LU CG Sparse Ax=b Solver for Normal Equations.
            Routine to solve a general linear system  Ax = b  using the
            incomplete LU decomposition with the Conjugate Gradient
            method applied to the normal equations, viz.,  AA'y = b,
            x = A'y.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SSLUCN-S, DSLUCN-D)
***KEYWORDS  ITERATIVE INCOMPLETE LU PRECONDITION,
             NON-SYMMETRIC LINEAR SYSTEM, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
     INTEGER ITER, IERR, IUNIT, LENW, IWORK(NL+NU+4*N+2), LENIW
     REAL B(N), X(N), A(NELT), TOL, ERR, RWORK(NL+NU+8*N)

     CALL SSLUCN(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
    $     ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW )

 *Arguments:
 N       :IN       Integer
          Order of the Matrix.
 B       :IN       Real B(N).
          Right-hand side vector.
 X       :INOUT    Real X(N).
          On input X is your initial guess for solution vector.
          On output X is the final approximate solution.
 NELT    :IN       Integer.
          Number of Non-Zeros stored in A.
 IA      :INOUT    Integer IA(NELT).
 JA      :INOUT    Integer JA(NELT).
 A       :INOUT    Real A(NELT).
          These arrays should hold the matrix A in either the SLAP
          Triad format or the SLAP Column format.  See "Description",
          below.  If the SLAP Triad format is chosen it is changed
          internally to the SLAP Column format.
 ISYM    :IN       Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the upper
          or lower triangle of the matrix is stored.
 ITOL    :IN       Integer.
          Flag to indicate type of convergence criterion.
          If ITOL=1, iteration stops when the 2-norm of the residual
          divided by the 2-norm of the right-hand side is less than TOL.
          If ITOL=2, iteration stops when the 2-norm of M-inv times the
```

```
         residual divided by the 2-norm of M-inv times the right hand
         side is less than TOL, where M-inv is the inverse of the
         diagonal of A.
         ITOL=11 is often useful for checking and comparing different
         routines.  For this case, the user must supply the "exact"
         solution or a very accurate approximation (one with an error
         much less than TOL) through a common block,
             COMMON /SSLBLK/ SOLN( )
         If ITOL=11, iteration stops when the 2-norm of the difference
         between the iterative approximation and the user-supplied
         solution divided by the 2-norm of the user-supplied solution
         is less than TOL.  Note that this requires the user to set up
         the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine.
         The routine with this declaration should be loaded before the
         stop test so that the correct length is used by the loader.
         This procedure is not standard Fortran and may not work
         correctly on your system (although it has worked on every
         system the authors have tried).  If ITOL is not 11 then this
         common block is indeed standard Fortran.
TOL     :INOUT    Real.
         Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX   :IN       Integer.
         Maximum number of iterations.
ITER    :OUT      Integer.
         Number of iterations required to reach convergence, or
         ITMAX+1 if convergence criterion could not be achieved in
         ITMAX iterations.
ERR     :OUT      Real.
         Error estimate of error in final approximate solution, as
         defined by ITOL.
IERR    :OUT      Integer.
         Return error flag.
           IERR = 0 => All went well.
           IERR = 1 => Insufficient space allocated for WORK or IWORK.
           IERR = 2 => Method failed to converge in ITMAX steps.
           IERR = 3 => Error in user input.
                       Check input values of N, ITOL.
           IERR = 4 => User error tolerance set too tight.
                       Reset to 500*R1MACH(3).  Iteration proceeded.
           IERR = 5 => Preconditioning matrix, M, is not positive
                       definite.  (r,z) < 0.
           IERR = 6 => Matrix A is not positive definite.  (p,Ap) < 0.
           IERR = 7 => Incomplete factorization broke down and was
                       fudged.  Resulting preconditioning may be less
                       than the best.
IUNIT   :IN       Integer.
         Unit number on which to write the error at each iteration,
         if this is desired for monitoring convergence.  If unit
         number is 0, no writing will occur.
RWORK   :WORK     Real RWORK(LENW).
         Real array used for workspace.
LENW    :IN       Integer.
         Length of the real workspace, RWORK.  LENW >= NL+NU+8*N.
         NL is the number of non-zeros in the lower triangle of the
         matrix (including the diagonal).
         NU is the number of non-zeros in the upper triangle of the
         matrix (including the diagonal).
IWORK   :WORK     Integer IWORK(LENIW).
         Integer array used for workspace.
         Upon return the following locations of IWORK hold information
```

which may be of use to the user:
IWORK(9) Amount of Integer workspace actually used.
IWORK(10) Amount of Real workspace actually used.

LENIW :IN Integer.
Length of the integer workspace, IWORK.
LENIW >= NL+NU+4*N+12.
NL is the number of non-zeros in the lower triangle of the matrix (including the diagonal).
NU is the number of non-zeros in the upper triangle of the matrix (including the diagonal).

*Description:

This routine is simply a driver for the SCGN routine. It calls the SSILUS routine to set up the preconditioning and then calls SCGN with the appropriate MATVEC and MSOLVE routines.

The Sparse Linear Algebra Package (SLAP) utilizes two matrix data structures: 1) the SLAP Triad format or 2) the SLAP Column format. The user can hand this routine either of the of these data structures and SLAP will figure out which on is being used and act accordingly.

========== S L A P Triad format ==========

This routine requires that the matrix A be stored in the SLAP Triad format. In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero.

The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Side Effects:
The SLAP Triad format (IA, JA, A) is modified internally to be the SLAP Column format. See above.

*Cautions:
This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

```
***SEE ALSO  SCGN, SDCGN, SSILUS
***REFERENCES  (NONE)
***ROUTINES CALLED  SCGN, SCHKW, SS2Y, SSILUS, SSMMTI, SSMTV, SSMV
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890921  Removed TeX from comments.  (FNF)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920407  COMMON BLOCK renamed SSLBLK.  (WRB)
   920511  Added complete declaration section.  (WRB)
   921113  Corrected C***CATEGORY line.  (FNF)
   END PROLOGUE
```

SSLUCS

```
      SUBROUTINE SSLUCS(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
     +   ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)
***BEGIN PROLOGUE  SSLUCS
***PURPOSE  Incomplete LU BiConjugate Gradient Squared Ax=b Solver.
            Routine to solve a linear system  Ax = b  using the
            BiConjugate Gradient Squared method with Incomplete LU
            decomposition preconditioning.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SSLUCS-S, DSLUCS-D)
***KEYWORDS  ITERATIVE INCOMPLETE LU PRECONDITION,
             NON-SYMMETRIC LINEAR SYSTEM, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
     INTEGER ITER, IERR, IUNIT, LENW, IWORK(NL+NU+4*N+2), LENIW
     REAL B(N), X(N), A(NELT), TOL, ERR, RWORK(NL+NU+8*N)

     CALL SSLUCS(N, B, X, NELT, IA, JA, A, ISYM, ITOL, TOL,
    $     ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)

 *Arguments:
 N      :IN       Integer.
         Order of the Matrix.
 B      :IN       Real B(N).
         Right-hand side vector.
 X      :INOUT    Real X(N).
         On input X is your initial guess for solution vector.
         On output X is the final approximate solution.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :INOUT    Integer IA(NELT).
 JA     :INOUT    Integer JA(NELT).
 A      :INOUT    Real A(NELT).
         These arrays should hold the matrix A in either the SLAP
         Triad format or the SLAP Column format.  See "Description",
         below.  If the SLAP Triad format is chosen it is changed
         internally to the SLAP Column format.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 ITOL   :IN       Integer.
         Flag to indicate type of convergence criterion.
         If ITOL=1, iteration stops when the 2-norm of the residual
         divided by the 2-norm of the right-hand side is less than TOL.
         This routine must calculate the residual from R = A*X - B.
         This is unnatural and hence expensive for this type of iter-
```

```
        ative method.  ITOL=2 is *STRONGLY* recommended.
        If ITOL=2, iteration stops when the 2-norm of M-inv times the
        residual divided by the 2-norm of M-inv times the right hand
        side is less than TOL, where M-inv time a vector is the pre-
        conditioning step.  This is the *NATURAL* stopping for this
        iterative method and is *STRONGLY* recommended.
TOL    :INOUT    Real.
        Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX  :IN       Integer.
        Maximum number of iterations.
ITER   :OUT      Integer.
        Number of iterations required to reach convergence, or
        ITMAX+1 if convergence criterion could not be achieved in
        ITMAX iterations.
ERR    :OUT      Real.
        Error estimate of error in final approximate solution, as
        defined by ITOL.
IERR   :OUT      Integer.
        Return error flag.
          IERR = 0 => All went well.
          IERR = 1 => Insufficient space allocated for WORK or IWORK.
          IERR = 2 => Method failed to converge in ITMAX steps.
          IERR = 3 => Error in user input.
                      Check input values of N, ITOL.
          IERR = 4 => User error tolerance set too tight.
                      Reset to 500*R1MACH(3).  Iteration proceeded.
          IERR = 5 => Breakdown of the method detected.
                      (r0,r) approximately 0.
          IERR = 6 => Stagnation of the method detected.
                      (r0,v) approximately 0.
          IERR = 7 => Incomplete factorization broke down and was
                      fudged.  Resulting preconditioning may be less
                      than the best.
IUNIT  :IN       Integer.
        Unit number on which to write the error at each iteration,
        if this is desired for monitoring convergence.  If unit
        number is 0, no writing will occur.
RWORK  :WORK     Real RWORK(LENW).
        Real array used for workspace.  NL is the number of non-
        zeros in the lower triangle of the matrix (including the
        diagonal).  NU is the number of non-zeros in the upper
        triangle of the matrix (including the diagonal).
LENW   :IN       Integer.
        Length of the real workspace, RWORK.  LENW >= NL+NU+8*N.
IWORK  :WORK     Integer IWORK(LENIW).
        Integer array used for workspace.  NL is the number of non-
        zeros in the lower triangle of the matrix (including the
        diagonal).  NU is the number of non-zeros in the upper
        triangle of the matrix (including the diagonal).
        Upon return the following locations of IWORK hold information
        which may be of use to the user:
        IWORK(9)  Amount of Integer workspace actually used.
        IWORK(10) Amount of Real workspace actually used.
LENIW  :IN       Integer.
        Length of the integer workspace, IWORK.
        LENIW >= NL+NU+4*N+12.

*Description:
      This routine is simply a  driver for the SCGSN  routine.  It
      calls the SSILUS routine to set  up the  preconditioning and
```

then calls SCGSN with the appropriate MATVEC, MTTVEC and MSOLVE, MTSOLV routines.

The Sparse Linear Algebra Package (SLAP) utilizes two matrix data structures: 1) the SLAP Triad format or 2) the SLAP Column format. The user can hand this routine either of the of these data structures and SLAP will figure out which on is being used and act accordingly.

========== S L A P Triad format ==========

This routine requires that the matrix A be stored in the SLAP Triad format. In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
```

```
                                 1  2  3    4  5    6  7    8    9 10 11
       |11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
       |21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
       | 0  0 33  0 35|  JA:  1  4  6    8  9   12
       | 0  0  0 44  0|
       |51  0 53  0 55|
```

*Side Effects:
The SLAP Triad format (IA, JA, A) is modified internally to be the SLAP Column format. See above.

*Cautions:
This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO SCGS, SSDCGS

***REFERENCES

1. P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric linear systems, Delft University of Technology Report 84-16, Department of Mathematics and Informatics, Delft, The Netherlands.
2. E. F. Kaasschieter, The solution of non-symmetric linear systems by biconjugate gradients or conjugate gradients squared, Delft University of Technology Report 86-21, Department of Mathematics and Informatics, Delft, The Netherlands.

***ROUTINES CALLED SCGS, SCHKW, SS2Y, SSILUS, SSLUI, SSMV

***REVISION HISTORY (YYMMDD)

```
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890921  Removed TeX from comments.  (FNF)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920511  Added complete declaration section.  (WRB)
   920929  Corrected format of references.  (FNF)
   921113  Corrected C***CATEGORY line.  (FNF)
   END PROLOGUE
```

SSLUGM

```
      SUBROUTINE SSLUGM(N, B, X, NELT, IA, JA, A, ISYM, NSAVE, ITOL,
     +   TOL, ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)
***BEGIN PROLOGUE  SSLUGM
***PURPOSE  Incomplete LU GMRES Iterative Sparse Ax=b Solver.
            This routine uses the generalized minimum residual
            (GMRES) method with incomplete LU factorization for
            preconditioning to solve possibly non-symmetric linear
            systems of the form: Ax = b.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SSLUGM-S, DSLUGM-D)
***KEYWORDS  GENERALIZED MINIMUM RESIDUAL, ITERATIVE PRECONDITION,
             NON-SYMMETRIC LINEAR SYSTEM, SLAP, SPARSE
***AUTHOR  Brown, Peter, (LLNL), pnbrown@llnl.gov
           Hindmarsh, Alan, (LLNL), alanh@llnl.gov
           Seager, Mark K., (LLNL), seager@llnl.gov
             Lawrence Livermore National Laboratory
             PO Box 808, L-60
             Livermore, CA 94550 (510) 423-3141
***DESCRIPTION

 *Usage:
      INTEGER   N, NELT, IA(NELT), JA(NELT), ISYM, NSAVE, ITOL
      INTEGER   ITMAX, ITER, IERR, IUNIT, LENW, IWORK(LENIW), LENIW
      REAL      B(N), X(N), A(NELT), TOL, ERR, RWORK(LENW)

      CALL SSLUGM(N, B, X, NELT, IA, JA, A, ISYM, NSAVE,
     $     ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT,
     $     RWORK, LENW, IWORK, LENIW)

 *Arguments:
 N      :IN       Integer.
         Order of the Matrix.
 B      :IN       Real B(N).
         Right-hand side vector.
 X      :INOUT    Real X(N).
         On input X is your initial guess for solution vector.
         On output X is the final approximate solution.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :IN       Integer IA(NELT).
 JA     :IN       Integer JA(NELT).
 A      :IN       Real A(NELT).
         These arrays should hold the matrix A in either the SLAP
         Triad format or the SLAP Column format.  See "Description",
         below.  If the SLAP Triad format is chosen it is changed
         internally to the SLAP Column format.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 NSAVE  :IN       Integer.
         Number of direction vectors to save and orthogonalize against.
         Must be greater than 1.
 ITOL   :IN       Integer.
```

Flag to indicate the type of convergence criterion used.

ITOL=0 Means the iteration stops when the test described below on the residual RL is satisfied. This is the "Natural Stopping Criteria" for this routine. Other values of ITOL cause extra, otherwise unnecessary, computation per iteration and are therefore much less efficient. See ISSGMR (the stop test routine) for more information.

ITOL=1 Means the iteration stops when the first test described below on the residual RL is satisfied, and there is either right or no preconditioning being used.

ITOL=2 Implies that the user is using left preconditioning, and the second stopping criterion below is used.

ITOL=3 Means the iteration stops when the third test described below on Minv*Residual is satisfied, and there is either left or no preconditioning begin used.

ITOL=11 is often useful for checking and comparing different routines. For this case, the user must supply the "exact" solution or a very accurate approximation (one with an error much less than TOL) through a common block,

```
    COMMON /SSLBLK/ SOLN( )
```

If ITOL=11, iteration stops when the 2-norm of the difference between the iterative approximation and the user-supplied solution divided by the 2-norm of the user-supplied solution is less than TOL. Note that this requires the user to set up the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine. The routine with this declaration should be loaded before the stop test so that the correct length is used by the loader. This procedure is not standard Fortran and may not work correctly on your system (although it has worked on every system the authors have tried). If ITOL is not 11 then this common block is indeed standard Fortran.

TOL :INOUT Real.

Convergence criterion, as described below. If TOL is set to zero on input, then a default value of 500*(the smallest positive magnitude, machine epsilon) is used.

ITMAX :IN Integer.

Maximum number of iterations. This routine uses the default of NRMAX = ITMAX/NSAVE to determine the when each restart should occur. See the description of NRMAX and MAXL in SGMRES for a full and frightfully interesting discussion of this topic.

ITER :OUT Integer.

Number of iterations required to reach convergence, or ITMAX+1 if convergence criterion could not be achieved in ITMAX iterations.

ERR :OUT Real.

Error estimate of error in final approximate solution, as defined by ITOL. Letting norm() denote the Euclidean norm, ERR is defined as follows...

If ITOL=0, then ERR = norm(SB*(B-A*X(L)))/norm(SB*B), for right or no preconditioning, and

ERR = norm(SB*(M-inverse)*(B-A*X(L)))/ norm(SB*(M-inverse)*B),

```
                                for left preconditioning.
        If ITOL=1, then ERR = norm(SB*(B-A*X(L)))/norm(SB*B),
                                since right or no preconditioning
                                being used.
        If ITOL=2, then ERR = norm(SB*(M-inverse)*(B-A*X(L)))/
                                 norm(SB*(M-inverse)*B),
                                since left preconditioning is being
                                used.
        If ITOL=3, then ERR =  Max  |(Minv*(B-A*X(L)))(i)/x(i)|
                                i=1,n
        If ITOL=11, then ERR = norm(SB*(X(L)-SOLN))/norm(SB*SOLN).
IERR   :OUT      Integer.
        Return error flag.
              IERR = 0 => All went well.
              IERR = 1 => Insufficient storage allocated for
                          RGWK or IGWK.
              IERR = 2 => Routine SPIGMR failed to reduce the norm
                          of the current residual on its last call,
                          and so the iteration has stalled.  In
                          this case, X equals the last computed
                          approximation.  The user must either
                          increase MAXL, or choose a different
                          initial guess.
              IERR =-1 => Insufficient length for RGWK array.
                          IGWK(6) contains the required minimum
                          length of the RGWK array.
              IERR =-2 => Inconsistent ITOL and JPRE values.
        For IERR <= 2, RGWK(1) = RHOL, which is the norm on the
        left-hand-side of the relevant stopping test defined
        below associated with the residual for the current
        approximation X(L).
IUNIT  :IN       Integer.
        Unit number on which to write the error at each iteration,
        if this is desired for monitoring convergence.  If unit
        number is 0, no writing will occur.
RWORK  :WORK    Real RWORK(LENW).
        Real array of size LENW.
LENW   :IN       Integer.
        Length of the real workspace, RWORK.
        LENW >= 1 + N*(NSAVE+7) +  NSAVE*(NSAVE+3)+NL+NU.
        Here NL is the number of non-zeros in the lower triangle of
        the matrix (including the diagonal) and NU is the number of
        non-zeros in the upper triangle of the matrix (including the
        diagonal).
        For the recommended values,  RWORK  has size at least
        131 + 17*N + NL + NU.
IWORK  :INOUT    Integer IWORK(LENIW).
        Used to hold pointers into the RWORK array.
        Upon return the following locations of IWORK hold information
        which may be of use to the user:
        IWORK(9)  Amount of Integer workspace actually used.
        IWORK(10) Amount of Real workspace actually used.
LENIW  :IN       Integer.
        Length of the integer workspace, IWORK.
        LENIW >= NL+NU+4*N+32.

*Description:
      SSLUGM solves a linear system A*X = B rewritten in the form:

       (SB*A*(M-inverse)*(SX-inverse))*(SX*M*X) = SB*B,
```

with right preconditioning, or

```
(SB*(M-inverse)*A*(SX-inverse))*(SX*X) = SB*(M-inverse)*B,
```

with left preconditioning, where A is an n-by-n real matrix, X and B are N-vectors, SB and SX are diagonal scaling matrices, and M is the Incomplete LU factorization of A. It uses preconditioned Krylov subpace methods based on the generalized minimum residual method (GMRES). This routine is a driver routine which assumes a SLAP matrix data structure and sets up the necessary information to do diagonal preconditioning and calls the main GMRES routine SGMRES for the solution of the linear system. SGMRES optionally performs either the full orthogonalization version of the GMRES algorithm or an incomplete variant of it. Both versions use restarting of the linear iteration by default, although the user can disable this feature.

The GMRES algorithm generates a sequence of approximations X(L) to the true solution of the above linear system. The convergence criteria for stopping the iteration is based on the size of the scaled norm of the residual R(L) = B - A*X(L). The actual stopping test is either:

```
        norm(SB*(B-A*X(L))) .le. TOL*norm(SB*B),
```

for right preconditioning, or

```
        norm(SB*(M-inverse)*(B-A*X(L))) .le.
                TOL*norm(SB*(M-inverse)*B),
```

for left preconditioning, where norm() denotes the Euclidean norm, and TOL is a positive scalar less than one input by the user. If TOL equals zero when SSLUGM is called, then a default value of 500*(the smallest positive magnitude, machine epsilon) is used. If the scaling arrays SB and SX are used, then ideally they should be chosen so that the vectors SX*X(or SX*M*X) and SB*B have all their components approximately equal to one in magnitude. If one wants to use the same scaling in X and B, then SB and SX can be the same array in the calling program.

The following is a list of the other routines and their functions used by GMRES:

SGMRES	Contains the matrix structure independent driver routine for GMRES.
SPIGMR	Contains the main iteration loop for GMRES.
SORTH	Orthogonalizes a new vector against older basis vectors.
SHEQR	Computes a QR decomposition of a Hessenberg matrix.
SHELS	Solves a Hessenberg least-squares system, using QR factors.
RLCALC	Computes the scaled residual RL.
XLCALC	Computes the solution XL.
ISSGMR	User-replaceable stopping routine.

The Sparse Linear Algebra Package (SLAP) utilizes two matrix data structures: 1) the SLAP Triad format or 2) the SLAP Column format. The user can hand this routine either of the of these data structures and SLAP will figure out which on

is being used and act accordingly.

```
=================== S L A P Triad format ===================
```

This routine requires that the matrix A be stored in the SLAP Triad format. In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

```
=================== S L A P Column format ==================
```

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Side Effects:

```
      The SLAP Triad format (IA, JA, A) is modified internally to be
      the SLAP Column format.  See above.

 *Cautions:
     This routine will attempt to write to the Fortran logical output
     unit IUNIT, if IUNIT .ne. 0.  Thus, the user must make sure that
     this logical unit is attached to a file or terminal before calling
     this routine with a non-zero value for IUNIT.  This routine does
     not check for the validity of a non-zero IUNIT unit number.

***REFERENCES  1. Peter N. Brown and A. C. Hindmarsh, Reduced Storage
                  Matrix Methods in Stiff ODE Systems, Lawrence Liver-
                  more National Laboratory Report UCRL-95088, Rev. 1,
                  Livermore, California, June 1987.
***ROUTINES CALLED  SCHKW, SGMRES, SS2Y, SSILUS, SSLUI, SSMV
***REVISION HISTORY  (YYMMDD)
   880615  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920407  COMMON BLOCK renamed SSLBLK.  (WRB)
   920511  Added complete declaration section.  (WRB)
   920929  Corrected format of references.  (FNF)
   921019  Corrected NEL to NL.  (FNF)
   END PROLOGUE
```

SSLUI

```
      SUBROUTINE SSLUI (N, B, X, NELT, IA, JA, A, ISYM, RWORK, IWORK)
***BEGIN PROLOGUE  SSLUI
***PURPOSE  SLAP MSOLVE for LDU Factorization.
            This routine acts as an interface between the SLAP generic
            MSOLVE calling convention and the routine that actually
                                 -1
            computes  (LDU)  B = X.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2E
***TYPE      SINGLE PRECISION (SSLUI-S, DSLUI-D)
***KEYWORDS  ITERATIVE PRECONDITION, NON-SYMMETRIC LINEAR SYSTEM SOLVE,
             SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION
       It is assumed that RWORK and IWORK have initialized with
       the information required for SSLUI2:
          IWORK(1) = Starting location of IL in IWORK.
          IWORK(2) = Starting location of JL in IWORK.
          IWORK(3) = Starting location of IU in IWORK.
          IWORK(4) = Starting location of JU in IWORK.
          IWORK(5) = Starting location of L in RWORK.
          IWORK(6) = Starting location of DINV in RWORK.
          IWORK(7) = Starting location of U in RWORK.
       See the DESCRIPTION of SSLUI2 for details.
***REFERENCES  (NONE)
***ROUTINES CALLED  SSLUI2
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920511  Added complete declaration section.  (WRB)
   921113  Corrected C***CATEGORY line.  (FNF)
   930701  Updated CATEGORY section.  (FNF, WRB)
   END PROLOGUE
```

SSLUI2

```
      SUBROUTINE SSLUI2 (N, B, X, IL, JL, L, DINV, IU, JU, U)
***BEGIN PROLOGUE  SSLUI2
***PURPOSE  SLAP Backsolve for LDU Factorization.
            Routine to solve a system of the form  L*D*U X = B,
            where L is a unit lower triangular matrix, D is a diagonal
            matrix, and U is a unit upper triangular matrix.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2E
***TYPE      SINGLE PRECISION (SSLUI2-S, DSLUI2-D)
***KEYWORDS  ITERATIVE PRECONDITION, NON-SYMMETRIC LINEAR SYSTEM SOLVE,
             SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, IL(NL), JL(NL), IU(NU), JU(NU)
     REAL    B(N), X(N), L(NL), DINV(N), U(NU)

     CALL SSLUI2( N, B, X, IL, JL, L, DINV, IU, JU, U )

 *Arguments:
 N       :IN       Integer
          Order of the Matrix.
 B       :IN       Real B(N).
          Right hand side.
 X       :OUT      Real X(N).
          Solution of L*D*U x = b.
 IL      :IN       Integer IL(NL).
 JL      :IN       Integer JL(NL).
 L       :IN       Real     L(NL).
          IL, JL, L contain the unit  lower triangular factor of the
          incomplete decomposition of some matrix stored in SLAP Row
          format.  The diagonal of ones *IS* stored.  This structure
          can   be   set  up  by   the  SSILUS  routine.   See   the
          "Description", below  for more   details about   the  SLAP
          format.  (NL is the number of non-zeros in the L array.)
 DINV    :IN       Real DINV(N).
          Inverse of the diagonal matrix D.
 IU      :IN       Integer IU(NU).
 JU      :IN       Integer JU(NU).
 U       :IN       Real     U(NU).
          IU, JU, U contain the unit upper triangular factor  of the
          incomplete decomposition  of  some  matrix stored in  SLAP
          Column format.   The diagonal of ones  *IS* stored.   This
          structure can be set up  by the SSILUS routine.  See   the
          "Description", below   for  more   details about  the SLAP
          format.  (NU is the number of non-zeros in the U array.)

 *Description:
       This routine is supplied with  the SLAP package as a routine
       to  perform  the  MSOLVE operation  in   the  SIR and   SBCG
```

iteration routines for the drivers SSILUR and SSLUBC. It must be called via the SLAP MSOLVE calling sequence convention interface routine SSLUI.

```
  **** THIS ROUTINE ITSELF DOES NOT CONFORM TO THE ****
        **** SLAP MSOLVE CALLING CONVENTION ****
```

IL, JL, L should contain the unit lower triangular factor of the incomplete decomposition of the A matrix stored in SLAP Row format. IU, JU, U should contain the unit upper factor of the incomplete decomposition of the A matrix stored in SLAP Column format This ILU factorization can be computed by the SSILUS routine. The diagonals (which are all one's) are stored.

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Row format ==========

This routine requires that the matrix A be stored in the SLAP Row format. In this format the non-zeros are stored counting across rows (except for the diagonal entry, which must appear first in each "row") and are stored in the real array A. In other words, for each row in the matrix put the diagonal entry in A. Then put in the other non-zero elements going across the row (except the diagonal) in order. The JA array holds the column index for each non-zero. The IA array holds the offsets into the JA, A arrays for the beginning of each row. That is, JA(IA(IROW)), A(IA(IROW)) points to the beginning of the IROW-th row in JA and A. JA(IA(IROW+1)-1), A(IA(IROW+1)-1) points to the end of the IROW-th row. Note that we always have IA(N+1) = NELT+1, where N is the number of rows in

the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Row storage format for a 5x5 Matrix (in the A and JA arrays '|' denotes the end of a row):

```
    5x5 Matrix         SLAP Row format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 12 15 | 22 21 | 33 35 | 44 | 55 51 53
|21 22  0  0  0|  JA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  IA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

With the SLAP format the "inner loops" of this routine should vectorize on machines with hardware support for vector gather/scatter operations. Your compiler may require a compiler directive to convince it that there are no implicit vector dependencies. Compiler directives for the Alliant FX/Fortran and CRI CFT/CFT77 compilers are supplied with the standard SLAP distribution.

```
***SEE ALSO  SSILUS
***REFERENCES  (NONE)
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920511  Added complete declaration section.  (WRB)
   921113  Corrected C***CATEGORY line.  (FNF)
   930701  Updated CATEGORY section.  (FNF, WRB)
   END PROLOGUE
```

SSLUI4

```
      SUBROUTINE SSLUI4 (N, B, X, IL, JL, L, DINV, IU, JU, U)
***BEGIN PROLOGUE  SSLUI4
***PURPOSE  SLAP Backsolve for LDU Factorization.
            Routine to solve a system of the form  (L*D*U)' X = B,
            where L is a unit lower triangular matrix, D is a diagonal
            matrix, and U is a unit upper triangular matrix and '
            denotes transpose.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2E
***TYPE      SINGLE PRECISION (SSLUI4-S, DSLUI4-D)
***KEYWORDS  ITERATIVE PRECONDITION, NON-SYMMETRIC LINEAR SYSTEM SOLVE,
             SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, IL(NL), JL(NL), IU(NU), JU(NU)
     REAL    B(N), X(N), L(NL), DINV(N), U(NU)

     CALL SSLUI4( N, B, X, IL, JL, L, DINV, IU, JU, U )

 *Arguments:
 N       :IN       Integer
          Order of the Matrix.
 B       :IN       Real B(N).
          Right hand side.
 X       :OUT      Real X(N).
          Solution of (L*D*U)trans x = b.
 IL      :IN       Integer IL(NL).
 JL      :IN       Integer JL(NL).
 L       :IN       Real     L(NL).
          IL, JL, L contain the unit lower triangular  factor of the
          incomplete decomposition of some matrix stored in SLAP Row
          format.  The diagonal of ones *IS* stored.  This structure
          can    be  set  up  by   the  SSILUS  routine.   See   the
          "Description",  below for  more  details about  the   SLAP
          format.  (NL is the number of non-zeros in the L array.)
 DINV    :IN       Real DINV(N).
          Inverse of the diagonal matrix D.
 IU      :IN       Integer IU(NU).
 JU      :IN       Integer JU(NU).
 U       :IN       Real     U(NU).
          IU, JU, U contain the  unit upper triangular factor of the
          incomplete  decomposition of some  matrix stored  in  SLAP
          Column  format.   The diagonal of  ones *IS* stored.  This
          structure can be set up by the  SSILUS routine.  See   the
          "Description",  below for  more  details  about  the  SLAP
          format.  (NU is the number of non-zeros in the U array.)

 *Description:
       This routine is supplied with the SLAP package as  a routine
```

to perform the MTSOLV operation in the SBCG iteration routine for the driver SSLUBC. It must be called via the SLAP MTSOLV calling sequence convention interface routine SSLUTI.

```
  **** THIS ROUTINE ITSELF DOES NOT CONFORM TO THE ****
        **** SLAP MSOLVE CALLING CONVENTION ****
```

IL, JL, L should contain the unit lower triangular factor of the incomplete decomposition of the A matrix stored in SLAP Row format. IU, JU, U should contain the unit upper factor of the incomplete decomposition of the A matrix stored in SLAP Column format This ILU factorization can be computed by the SSILUS routine. The diagonals (which are all one's) are stored.

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Row format ==========

This routine requires that the matrix A be stored in the SLAP Row format. In this format the non-zeros are stored counting across rows (except for the diagonal entry, which must appear first in each "row") and are stored in the real array A. In other words, for each row in the matrix put the diagonal entry in A. Then put in the other non-zero elements going across the row (except the diagonal) in order. The JA array holds the column index for each non-zero. The IA array holds the offsets into the JA, A arrays for the beginning of each row. That is, JA(IA(IROW)), A(IA(IROW)) points to the beginning of the IROW-th row in JA and A. JA(IA(IROW+1)-1), A(IA(IROW+1)-1) points to the end of the IROW-th row. Note that we always

have IA(N+1) = NELT+1, where N is the number of rows in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Row storage format for a 5x5 Matrix (in the A and JA arrays '|' denotes the end of a row):

```
    5x5 Matrix         SLAP Row format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 12 15 | 22 21 | 33 35 | 44 | 55 51 53
|21 22  0  0  0|  JA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  IA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

With the SLAP format the "inner loops" of this routine should vectorize on machines with hardware support for vector gather/scatter operations. Your compiler may require a compiler directive to convince it that there are no implicit vector dependencies. Compiler directives for the Alliant FX/Fortran and CRI CFT/CFT77 compilers are supplied with the standard SLAP distribution.

```
***SEE ALSO  SSILUS
***REFERENCES  (NONE)
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920511  Added complete declaration section.  (WRB)
   921113  Corrected C***CATEGORY line.  (FNF)
   930701  Updated CATEGORY section.  (FNF, WRB)
   END PROLOGUE
```

SSLUOM

```
      SUBROUTINE SSLUOM(N, B, X, NELT, IA, JA, A, ISYM, NSAVE, ITOL, TOL,
     +   TOL, ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW)
***BEGIN PROLOGUE  SSLUOM
***PURPOSE  Incomplete LU Orthomin Sparse Iterative Ax=b Solver.
            Routine to solve a general linear system  Ax = b  using
            the Orthomin method with Incomplete LU decomposition.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2A4, D2B4
***TYPE      SINGLE PRECISION (SSLUOM-S, DSLUOM-D)
***KEYWORDS  ITERATIVE INCOMPLETE LU PRECONDITION,
             NON-SYMMETRIC LINEAR SYSTEM, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
      INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, NSAVE, ITOL, ITMAX
      INTEGER ITER, IERR, IUNIT, LENW, IWORK(NL+NU+4*N+2), LENIW
      REAL B(N), X(N), A(NELT), TOL, ERR
      REAL RWORK(NL+NU+7*N+3*N*NSAVE+NSAVE)

      CALL SSLUOM(N, B, X, NELT, IA, JA, A, ISYM, NSAVE, ITOL, TOL,
     $     ITMAX, ITER, ERR, IERR, IUNIT, RWORK, LENW, IWORK, LENIW )

 *Arguments:
 N      :IN       Integer.
         Order of the matrix.
 B      :IN       Real B(N).
         Right-hand side vector.
 X      :INOUT    Real X(N).
         On input X is your initial guess for solution vector.
         On output X is the final approximate solution.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :INOUT    Integer IA(NELT).
 JA     :INOUT    Integer JA(NELT).
 A      :INOUT    Real A(NELT).
         These arrays should hold the matrix A in either the SLAP
         Triad format or the SLAP Column format.  See "Description",
         below.  If the SLAP Triad format is chosen, it is changed
         internally to the SLAP Column format.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.
 NSAVE  :IN       Integer.
         Number of direction vectors to save and orthogonalize against.
 ITOL   :IN       Integer.
         Flag to indicate type of convergence criterion.
         If ITOL=1, iteration stops when the 2-norm of the residual
         divided by the 2-norm of the right-hand side is less than TOL.
```

```
        If ITOL=2, iteration stops when the 2-norm of M-inv times the
        residual divided by the 2-norm of M-inv times the right hand
        side is less than TOL, where M-inv is the inverse of the
        diagonal of A.
        ITOL=11 is often useful for checking and comparing different
        routines.  For this case, the user must supply the "exact"
        solution or a very accurate approximation (one with an error
        much less than TOL) through a common block,
            COMMON /SSLBLK/ SOLN( )
        If ITOL=11, iteration stops when the 2-norm of the difference
        between the iterative approximation and the user-supplied
        solution divided by the 2-norm of the user-supplied solution
        is less than TOL.  Note that this requires the user to set up
        the "COMMON /SSLBLK/ SOLN(LENGTH)" in the calling routine.
        The routine with this declaration should be loaded before the
        stop test so that the correct length is used by the loader.
        This procedure is not standard Fortran and may not work
        correctly on your system (although it has worked on every
        system the authors have tried).  If ITOL is not 11 then this
        common block is indeed standard Fortran.
TOL    :INOUT    Real.
        Convergence criterion, as described above.  (Reset if IERR=4.)
ITMAX  :IN       Integer.
        Maximum number of iterations.
ITER   :OUT      Integer.
        Number of iterations required to reach convergence, or
        ITMAX+1 if convergence criterion could not be achieved in
        ITMAX iterations.
ERR    :OUT      Real.
        Error estimate of error in final approximate solution, as
        defined by ITOL.
IERR   :OUT      Integer.
        Return error flag.
          IERR = 0 => All went well.
          IERR = 1 => Insufficient space allocated for WORK or IWORK.
          IERR = 2 => Method failed to converge in ITMAX steps.
          IERR = 3 => Error in user input.
                      Check input values of N, ITOL.
          IERR = 4 => User error tolerance set too tight.
                      Reset to 500*R1MACH(3).  Iteration proceeded.
          IERR = 5 => Preconditioning matrix, M, is not positive
                      definite.  (r,z) < 0.
          IERR = 6 => Breakdown of the method detected.
                      (p,Ap) < epsilon**2.
          IERR = 7 => Incomplete factorization broke down and was
                      fudged.  Resulting preconditioning may be less
                      than the best.
IUNIT  :IN       Integer.
        Unit number on which to write the error at each iteration,
        if this is desired for monitoring convergence.  If unit
        number is 0, no writing will occur.
RWORK  :WORK     Real RWORK(LENW).
        Real array used for workspace.  NL is the number of non-
        zeros in the lower triangle of the matrix (including the
        diagonal).  NU is the number of non-zeros in the upper
        triangle of the matrix (including the diagonal).
LENW   :IN       Integer.
        Length of the real workspace, RWORK.
        LENW >= NL+NU+4*N+NSAVE*(3*N+1)
IWORK  :WORK     Integer IWORK(LENIW)
```

Integer array used for workspace. NL is the number of non-zeros in the lower triangle of the matrix (including the diagonal). NU is the number of non-zeros in the upper triangle of the matrix (including the diagonal).
Upon return the following locations of IWORK hold information which may be of use to the user:
IWORK(9) Amount of Integer workspace actually used.
IWORK(10) Amount of Real workspace actually used.

LENIW :IN Integer.
Length of the integer workspace, IWORK.
LENIW >= NL+NU+4*N+12.

*Description:

This routine is simply a driver for the SOMN routine. It calls the SSILUS routine to set up the preconditioning and then calls SOMN with the appropriate MATVEC and MSOLVE routines.

The Sparse Linear Algebra Package (SLAP) utilizes two matrix data structures: 1) the SLAP Triad format or 2) the SLAP Column format. The user can hand this routine either of the of these data structures and SLAP will figure out which on is being used and act accordingly.

========== S L A P Triad format ==========

This routine requires that the matrix A be stored in the SLAP Triad format. In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero

elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Side Effects:
The SLAP Triad format (IA, JA, A) is modified internally to be the SLAP Column format. See above.

*Cautions:
This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

***SEE ALSO SOMN, SSDOMN
***REFERENCES (NONE)
***ROUTINES CALLED SCHKW, SOMN, SS2Y, SSILUS, SSLUI, SSMV
***REVISION HISTORY (YYMMDD)
871119 DATE WRITTEN
881213 Previous REVISION DATE
890915 Made changes requested at July 1989 CML Meeting. (MKS)
890921 Removed TeX from comments. (FNF)
890922 Numerous changes to prologue to make closer to SLATEC standard. (FNF)
890929 Numerous changes to reduce SP/DP differences. (FNF)
910411 Prologue converted to Version 4.0 format. (BAB)
920407 COMMON BLOCK renamed SSLBLK. (WRB)
920511 Added complete declaration section. (WRB)
921019 Corrected NEL to NL. (FNF)
921113 Corrected C***CATEGORY line. (FNF)
END PROLOGUE

SSLUTI

```
      SUBROUTINE SSLUTI (N, B, X, NELT, IA, JA, A, ISYM, RWORK, IWORK)
***BEGIN PROLOGUE  SSLUTI
***PURPOSE  SLAP MTSOLV for LDU Factorization.
            This routine acts as an interface between the SLAP generic
            MTSOLV calling convention and the routine that actually
                                 -T
            computes  (LDU)  B = X.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2E
***TYPE      SINGLE PRECISION (SSLUTI-S, DSLUTI-D)
***KEYWORDS  ITERATIVE PRECONDITION, LINEAR SYSTEM SOLVE, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION
       It is assumed that RWORK and IWORK have initialized with
       the information required for SSLUI4:
          IWORK(1) = Starting location of IL in IWORK.
          IWORK(2) = Starting location of JL in IWORK.
          IWORK(3) = Starting location of IU in IWORK.
          IWORK(4) = Starting location of JU in IWORK.
          IWORK(5) = Starting location of L in RWORK.
          IWORK(6) = Starting location of DINV in RWORK.
          IWORK(7) = Starting location of U in RWORK.
       See the DESCRIPTION of SSLUI4 for details.
***REFERENCES  (NONE)
***ROUTINES CALLED  SSLUI4
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920511  Added complete declaration section.  (WRB)
   921113  Corrected C***CATEGORY line.  (FNF)
   930701  Updated CATEGORY section.  (FNF, WRB)
   END PROLOGUE
```

SSMMI2

```
      SUBROUTINE SSMMI2 (N, B, X, IL, JL, L, DINV, IU, JU, U)
***BEGIN PROLOGUE  SSMMI2
***PURPOSE  SLAP Backsolve for LDU Factorization of Normal Equations.
            To solve a system of the form  (L*D*U)*(L*D*U)' X = B,
            where L is a unit lower triangular matrix, D is a diagonal
            matrix, and U is a unit upper triangular matrix and '
            denotes transpose.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2E
***TYPE      SINGLE PRECISION (SSMMI2-S, DSMMI2-D)
***KEYWORDS  ITERATIVE PRECONDITION, LINEAR SYSTEM, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
      INTEGER N, IL(NL), JL(NL), IU(NU), JU(NU)
      REAL    B(N), X(N), L(NL), DINV(N), U(NU)

      CALL SSMMI2( N, B, X, IL, JL, L, DINV, IU, JU, U )

 *Arguments:
 N       :IN       Integer
          Order of the Matrix.
 B       :IN       Real B(N).
          Right hand side.
 X       :OUT      Real X(N).
          Solution of (L*D*U)(L*D*U)trans x = b.
 IL      :IN       Integer IL(NL).
 JL      :IN       Integer JL(NL).
 L       :IN       Real     L(NL).
          IL, JL, L contain the unit lower  triangular factor of the
          incomplete decomposition of some matrix stored in SLAP Row
          format.  The diagonal of ones *IS* stored.  This structure
          can  be  set  up  by   the  SSILUS   routine.    See   the
          "Description", below for  more   details   about  the SLAP
          format.  (NL is the number of non-zeros in the L array.)
 DINV    :IN       Real DINV(N).
          Inverse of the diagonal matrix D.
 IU      :IN       Integer IU(NU).
 JU      :IN       Integer JU(NU).
 U       :IN       Real     U(NU).
          IU, JU, U contain the unit upper  triangular factor of the
          incomplete decomposition  of   some matrix stored in  SLAP
          Column  format.  The diagonal  of  ones *IS* stored.  This
          structure can be set up  by the SSILUS routine.  See   the
          "Description",  below  for  more  details  about  the SLAP
          format.  (NU is the number of non-zeros in the U array.)

 *Description:
       This routine is supplied with the SLAP package as  a routine
       to  perform  the  MSOLVE  operation  in  the SBCGN iteration
```

routine for the driver SSLUCN. It must be called via the SLAP MSOLVE calling sequence convention interface routine SSMMTI.

```
  **** THIS ROUTINE ITSELF DOES NOT CONFORM TO THE ****
        **** SLAP MSOLVE CALLING CONVENTION ****
```

IL, JL, L should contain the unit lower triangular factor of the incomplete decomposition of the A matrix stored in SLAP Row format. IU, JU, U should contain the unit upper factor of the incomplete decomposition of the A matrix stored in SLAP Column format This ILU factorization can be computed by the SSILUS routine. The diagonals (which are all one's) are stored.

========== S L A P Column format ==========

This routine requires that the matrix A be stored in the SLAP Column format. In this format the non-zeros are stored counting down columns (except for the diagonal entry, which must appear first in each "column") and are stored in the real array A. In other words, for each column in the matrix put the diagonal entry in A. Then put in the other non-zero elements going down the column (except the diagonal) in order. The IA array holds the row index for each non-zero. The JA array holds the offsets into the IA, A arrays for the beginning of each column. That is, IA(JA(ICOL)), A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

========== S L A P Row format ==========

This routine requires that the matrix A be stored in the SLAP Row format. In this format the non-zeros are stored counting across rows (except for the diagonal entry, which must appear first in each "row") and are stored in the real array A. In other words, for each row in the matrix put the diagonal entry in A. Then put in the other non-zero elements going across the row (except the diagonal) in order. The JA array holds the column index for each non-zero. The IA array holds the offsets into the JA, A arrays for the beginning of each row. That is, JA(IA(IROW)), A(IA(IROW)) points to the beginning of the IROW-th row in JA and A. JA(IA(IROW+1)-1), A(IA(IROW+1)-1) points to the end of the IROW-th row. Note that we always have IA(N+1) = NELT+1, where N is the number of rows in

the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Row storage format for a 5x5 Matrix (in the A and JA arrays '|' denotes the end of a row):

```
    5x5 Matrix         SLAP Row format for 5x5 matrix on left.
                        1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 12 15 | 22 21 | 33 35 | 44 | 55 51 53
|21 22  0  0  0|  JA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  IA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

With the SLAP format the "inner loops" of this routine should vectorize on machines with hardware support for vector gather/scatter operations. Your compiler may require a compiler directive to convince it that there are no implicit vector dependencies. Compiler directives for the Alliant FX/Fortran and CRI CFT/CFT77 compilers are supplied with the standard SLAP distribution.

```
***SEE ALSO  SSILUS
***REFERENCES  (NONE)
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920511  Added complete declaration section.  (WRB)
   921113  Corrected C***CATEGORY line.  (FNF)
   930701  Updated CATEGORY section.  (FNF, WRB)
   END PROLOGUE
```

SSMMTI

```
      SUBROUTINE SSMMTI (N, B, X, NELT, IA, JA, A, ISYM, RWORK, IWORK)
***BEGIN PROLOGUE  SSMMTI
***PURPOSE  SLAP MSOLVE for LDU Factorization of Normal Equations.
            This routine acts as an interface between the SLAP generic
            MMTSLV calling convention and the routine that actually
                                                 -1
            computes  [(LDU)*(LDU)']  B = X.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D2E
***TYPE      SINGLE PRECISION (SSMMTI-S, DSMMTI-D)
***KEYWORDS  ITERATIVE PRECONDITION, LINEAR SYSTEM SOLVE, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION
       It is assumed that RWORK and IWORK have initialized with
       the information required for SSMMI2:
          IWORK(1) = Starting location of IL in IWORK.
          IWORK(2) = Starting location of JL in IWORK.
          IWORK(3) = Starting location of IU in IWORK.
          IWORK(4) = Starting location of JU in IWORK.
          IWORK(5) = Starting location of L in RWORK.
          IWORK(6) = Starting location of DINV in RWORK.
          IWORK(7) = Starting location of U in RWORK.
       See the DESCRIPTION of SSMMI2 for details.
***REFERENCES  (NONE)
***ROUTINES CALLED  SSMMI2
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920511  Added complete declaration section.  (WRB)
   921113  Corrected C***CATEGORY line.  (FNF)
   930701  Updated CATEGORY section.  (FNF, WRB)
   END PROLOGUE
```

SSMTV

```
      SUBROUTINE SSMTV (N, X, Y, NELT, IA, JA, A, ISYM)
***BEGIN PROLOGUE  SSMTV
***PURPOSE  SLAP Column Format Sparse Matrix Transpose Vector Product.
            Routine to calculate the sparse matrix vector product:
            Y = A'*X, where ' denotes transpose.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (SSMTV-S, DSMTV-D)
***KEYWORDS  MATRIX TRANSPOSE VECTOR MULTIPLY, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER  N, NELT, IA(NELT), JA(NELT), ISYM
     REAL     X(N), Y(N), A(NELT)

     CALL SSMTV(N, X, Y, NELT, IA, JA, A, ISYM )

 *Arguments:
 N      :IN       Integer.
         Order of the Matrix.
 X      :IN       Real X(N).
         The vector that should be multiplied by the transpose of
         the matrix.
 Y      :OUT      Real Y(N).
         The product of the transpose of the matrix and the vector.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :IN       Integer IA(NELT).
 JA     :IN       Integer JA(NELT).
 A      :IN       Real A(NELT).
         These arrays should hold the matrix A in the SLAP Column
         format.  See "Description", below.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.

 *Description
       ================== S L A P Column format ==================
       This routine  requires that  the matrix A  be stored in  the
       SLAP Column format.  In this format the non-zeros are stored
       counting down columns (except for  the diagonal entry, which
       must appear first in each  "column")  and are stored  in the
       real array A.  In other words, for each column in the matrix
       put the diagonal entry in A.  Then put in the other non-zero
       elements going down   the  column (except  the diagonal)  in
       order.  The IA array holds the row  index for each non-zero.
       The JA array holds the offsets into the IA, A arrays for the
       beginning of   each    column.    That  is,    IA(JA(ICOL)),
```

A(JA(ICOL)) points to the beginning of the ICOL-th column in IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

With the SLAP format the "inner loops" of this routine should vectorize on machines with hardware support for vector gather/scatter operations. Your compiler may require a compiler directive to convince it that there are no implicit vector dependencies. Compiler directives for the Alliant FX/Fortran and CRI CFT/CFT77 compilers are supplied with the standard SLAP distribution.

*Cautions:

This routine assumes that the matrix A is stored in SLAP Column format. It does not check for this (for speed) and evil, ugly, ornery and nasty things will happen if the matrix data structure is, in fact, not SLAP Column. Beware of the wrong data structure!!!

***SEE ALSO SSMV
***REFERENCES (NONE)
***ROUTINES CALLED (NONE)
***REVISION HISTORY (YYMMDD)

```
871119  DATE WRITTEN
881213  Previous REVISION DATE
890915  Made changes requested at July 1989 CML Meeting.  (MKS)
890922  Numerous changes to prologue to make closer to SLATEC
        standard.  (FNF)
890929  Numerous changes to reduce SP/DP differences.  (FNF)
910411  Prologue converted to Version 4.0 format.  (BAB)
920511  Added complete declaration section.  (WRB)
930701  Updated CATEGORY section.  (FNF, WRB)
END PROLOGUE
```

SSMV

```
      SUBROUTINE SSMV (N, X, Y, NELT, IA, JA, A, ISYM)
***BEGIN PROLOGUE  SSMV
***PURPOSE  SLAP Column Format Sparse Matrix Vector Product.
            Routine to calculate the sparse matrix vector product:
            Y = A*X.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (SSMV-S, DSMV-D)
***KEYWORDS  MATRIX VECTOR MULTIPLY, SLAP, SPARSE
***AUTHOR  Greenbaum, Anne, (Courant Institute)
           Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER  N, NELT, IA(NELT), JA(NELT), ISYM
     REAL     X(N), Y(N), A(NELT)

     CALL SSMV(N, X, Y, NELT, IA, JA, A, ISYM )

 *Arguments:
 N      :IN       Integer.
         Order of the Matrix.
 X      :IN       Real X(N).
         The vector that should be multiplied by the matrix.
 Y      :OUT      Real Y(N).
         The product of the matrix and the vector.
 NELT   :IN       Integer.
         Number of Non-Zeros stored in A.
 IA     :IN       Integer IA(NELT).
 JA     :IN       Integer JA(NELT).
 A      :IN       Real A(NELT).
         These arrays should hold the matrix A in the SLAP Column
         format.  See "Description", below.
 ISYM   :IN       Integer.
         Flag to indicate symmetric storage format.
         If ISYM=0, all non-zero entries of the matrix are stored.
         If ISYM=1, the matrix is symmetric, and only the upper
         or lower triangle of the matrix is stored.

 *Description
       ================== S L A P Column format ==================
       This routine  requires that  the matrix A  be stored in  the
       SLAP Column format.  In this format the non-zeros are stored
       counting down columns (except for  the diagonal entry, which
       must appear first in each  "column")  and are stored  in the
       real array A.  In other words, for each column in the matrix
       put the diagonal entry in A.  Then put in the other non-zero
       elements going down   the  column (except  the diagonal)  in
       order.  The IA array holds the row  index for each non-zero.
       The JA array holds the offsets into the IA, A arrays for the
       beginning of   each    column.    That  is,    IA(JA(ICOL)),
       A(JA(ICOL)) points to the beginning of the ICOL-th column in
```

IA and A. IA(JA(ICOL+1)-1), A(JA(ICOL+1)-1) points to the end of the ICOL-th column. Note that we always have JA(N+1) = NELT+1, where N is the number of columns in the matrix and NELT is the number of non-zeros in the matrix.

Here is an example of the SLAP Column storage format for a 5x5 Matrix (in the A and IA arrays '|' denotes the end of a column):

```
    5x5 Matrix      SLAP Column format for 5x5 matrix on left.
                       1  2  3    4  5    6  7    8    9 10 11
|11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
|21 22  0  0  0|  IA:  1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
| 0  0 33  0 35|  JA:  1  4  6    8  9   12
| 0  0  0 44  0|
|51  0 53  0 55|
```

With the SLAP format the "inner loops" of this routine should vectorize on machines with hardware support for vector gather/scatter operations. Your compiler may require a compiler directive to convince it that there are no implicit vector dependencies. Compiler directives for the Alliant FX/Fortran and CRI CFT/CFT77 compilers are supplied with the standard SLAP distribution.

*Cautions:
This routine assumes that the matrix A is stored in SLAP Column format. It does not check for this (for speed) and evil, ugly, ornery and nasty things will happen if the matrix data structure is, in fact, not SLAP Column. Beware of the wrong data structure!!!

***SEE ALSO SSMTV
***REFERENCES (NONE)
***ROUTINES CALLED (NONE)
***REVISION HISTORY (YYMMDD)
871119 DATE WRITTEN
881213 Previous REVISION DATE
890915 Made changes requested at July 1989 CML Meeting. (MKS)
890922 Numerous changes to prologue to make closer to SLATEC standard. (FNF)
890929 Numerous changes to reduce SP/DP differences. (FNF)
910411 Prologue converted to Version 4.0 format. (BAB)
920511 Added complete declaration section. (WRB)
930701 Updated CATEGORY section. (FNF, WRB)
END PROLOGUE

SSORT

```
      SUBROUTINE SSORT (X, Y, N, KFLAG)
***BEGIN PROLOGUE  SSORT
***PURPOSE  Sort an array and optionally make the same interchanges in
            an auxiliary array.  The array may be sorted in increasing
            or decreasing order.  A slightly modified QUICKSORT
            algorithm is used.
***LIBRARY   SLATEC
***CATEGORY  N6A2B
***TYPE      SINGLE PRECISION (SSORT-S, DSORT-D, ISORT-I)
***KEYWORDS  SINGLETON QUICKSORT, SORT, SORTING
***AUTHOR  Jones, R. E., (SNLA)
           Wisniewski, J. A., (SNLA)
***DESCRIPTION

   SSORT sorts array X and optionally makes the same interchanges in
   array Y.  The array X may be sorted in increasing order or
   decreasing order.  A slightly modified quicksort algorithm is used.

   Description of Parameters
      X - array of values to be sorted   (usually abscissas)
      Y - array to be (optionally) carried along
      N - number of values in array X to be sorted
      KFLAG - control parameter
            =  2  means sort X in increasing order and carry Y along.
            =  1  means sort X in increasing order (ignoring Y)
            = -1  means sort X in decreasing order (ignoring Y)
            = -2  means sort X in decreasing order and carry Y along.

***REFERENCES  R. C. Singleton, Algorithm 347, An efficient algorithm
                 for sorting with minimal storage, Communications of
                 the ACM, 12, 3 (1969), pp. 185-187.
***ROUTINES CALLED  XERMSG
***REVISION HISTORY  (YYMMDD)
   761101  DATE WRITTEN
   761118  Modified to use the Singleton quicksort algorithm.  (JAW)
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   891009  Removed unreferenced statement labels.  (WRB)
   891024  Changed category.  (WRB)
   891024  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   901012  Declared all variables; changed X,Y to SX,SY. (M. McClain)
   920501  Reformatted the REFERENCES section.  (DWL, WRB)
   920519  Clarified error messages.  (DWL)
   920801  Declarations section rebuilt and code restructured to use
           IF-THEN-ELSE-ENDIF.  (RWC, WRB)
   END PROLOGUE
```

SSPCO

```
      SUBROUTINE SSPCO (AP, N, KPVT, RCOND, Z)
***BEGIN PROLOGUE  SSPCO
***PURPOSE  Factor a real symmetric matrix stored in packed form
            by elimination with symmetric pivoting and estimate the
            condition number of the matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1A
***TYPE      SINGLE PRECISION (SSPCO-S, DSPCO-D, CHPCO-C, CSPCO-C)
***KEYWORDS  CONDITION NUMBER, LINEAR ALGEBRA, LINPACK,
             MATRIX FACTORIZATION, PACKED, SYMMETRIC
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      SSPCO factors a real symmetric matrix stored in packed
      form by elimination with symmetric pivoting and estimates
      the condition of the matrix.

      If  RCOND  is not needed, SSPFA is slightly faster.
      To solve  A*X = B , follow SSPCO by SSPSL.
      To compute  INVERSE(A)*C , follow SSPCO by SSPSL.
      To compute  INVERSE(A) , follow SSPCO by SSPDI.
      To compute  DETERMINANT(A) , follow SSPCO by SSPDI.
      To compute  INERTIA(A), follow SSPCO by SSPDI.

      On Entry

         AP      REAL (N*(N+1)/2)
                 the packed form of a symmetric matrix  A .  The
                 columns of the upper triangle are stored sequentially
                 in a one-dimensional array of length  N*(N+1)/2 .
                 See comments below for details.

         N       INTEGER
                 the order of the matrix  A .

      Output

         AP      a block diagonal matrix and the multipliers which
                 were used to obtain it stored in packed form.
                 The factorization can be written  A = U*D*TRANS(U)
                 where  U  is a product of permutation and unit
                 upper triangular matrices , TRANS(U) is the
                 transpose of  U , and  D  is block diagonal
                 with 1 by 1 and 2 by 2 blocks.

         KPVT    INTEGER(N)
                 an integer vector of pivot indices.

         RCOND   REAL
                 an estimate of the reciprocal condition of  A .
                 For the system  A*X = B , relative perturbations
                 in  A  and  B  of size  EPSILON  may cause
                 relative perturbations in  X  of size  EPSILON/RCOND .
                 If  RCOND  is so small that the logical expression
                           1.0 + RCOND .EQ. 1.0
                 is true, then  A  may be singular to working
```

```
                precision.  In particular,  RCOND  is zero  if
                exact singularity is detected or the estimate
                underflows.

        Z       REAL(N)
                a work vector whose contents are usually unimportant.
                If  A  is close to a singular matrix, then  Z  is
                an approximate null vector in the sense that
                NORM(A*Z) = RCOND*NORM(A)*NORM(Z) .

     Packed Storage

          The following program segment will pack the upper
          triangle of a symmetric matrix.

                K = 0
                DO 20 J = 1, N
                   DO 10 I = 1, J
                      K = K + 1
                      AP(K) = A(I,J)
             10    CONTINUE
             20 CONTINUE

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SASUM, SAXPY, SDOT, SSCAL, SSPFA
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   891107  Modified routine equivalence list.  (WRB)
   891107  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SSPDI

```
      SUBROUTINE SSPDI (AP, N, KPVT, DET, INERT, WORK, JOB)
***BEGIN PROLOGUE  SSPDI
***PURPOSE  Compute the determinant, inertia, inverse of a real
            symmetric matrix stored in packed form using the factors
            from SSPFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1A, D3B1A
***TYPE      SINGLE PRECISION (SSPDI-S, DSPDI-D, CHPDI-C, CSPDI-C)
***KEYWORDS  DETERMINANT, INVERSE, LINEAR ALGEBRA, LINPACK, MATRIX,
             PACKED, SYMMETRIC
***AUTHOR  Bunch, J., (UCSD)
***DESCRIPTION

      SSPDI computes the determinant, inertia and inverse
      of a real symmetric matrix using the factors from SSPFA,
      where the matrix is stored in packed form.

      On Entry

         AP      REAL (N*(N+1)/2)
                 the output from SSPFA.

         N       INTEGER
                 the order of the matrix A.

         KPVT    INTEGER(N)
                 the pivot vector from SSPFA.

         WORK    REAL(N)
                 work vector.  Contents ignored.

         JOB     INTEGER
                 JOB has the decimal expansion  ABC  where
                    If  C .NE. 0, the inverse is computed,
                    If  B .NE. 0, the determinant is computed,
                    If  A .NE. 0, the inertia is computed.

                 For example, JOB = 111  gives all three.

      On Return

         Variables not requested by JOB are not used.

         AP     contains the upper triangle of the inverse of
                the original matrix, stored in packed form.
                The columns of the upper triangle are stored
                sequentially in a one-dimensional array.

         DET    REAL(2)
                determinant of original matrix.
                Determinant = DET(1) * 10.0**DET(2)
                with 1.0 .LE. ABS(DET(1)) .LT. 10.0
                or DET(1) = 0.0.

         INERT  INTEGER(3)
                the inertia of the original matrix.
```

```
                INERT(1)  =  number of positive eigenvalues.
                INERT(2)  =  number of negative eigenvalues.
                INERT(3)  =  number of zero eigenvalues.

     Error Condition

        A division by zero will occur if the inverse is requested
        and  SSPCO  has set RCOND .EQ. 0.0
        or  SSPFA  has set  INFO .NE. 0 .

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SCOPY, SDOT, SSWAP
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   891107  Modified routine equivalence list.  (WRB)
   891107  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SSPEV

```
      SUBROUTINE SSPEV (A, N, E, V, LDV, WORK, JOB, INFO)
***BEGIN PROLOGUE  SSPEV
***PURPOSE  Compute the eigenvalues and, optionally, the eigenvectors
            of a real symmetric matrix stored in packed form.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4A1
***TYPE      SINGLE PRECISION (SSPEV-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK, PACKED, SYMMETRIC
***AUTHOR  Kahaner, D. K., (NBS)
           Moler, C. B., (U. of New Mexico)
           Stewart, G. W., (U. of Maryland)
***DESCRIPTION

      Abstract
       SSPEV computes the eigenvalues and, optionally, the eigenvectors
       of a real symmetric matrix stored in packed form.

      Call Sequence Parameters-
        (The values of parameters marked with * (star) will be  changed
          by SSPEV.)

         A*      REAL(N*(N+1)/2)
                 real symmetric packed input matrix.  Contains upper
                 triangle and diagonal of A, by column (elements
                 11, 12, 22, 13, 23, 33, ...).

         N       INTEGER
                 set by the user to
                 the order of the matrix A.

         E*      REAL(N)
                 on return from SSPEV, E contains the eigenvalues of A.
                 See also INFO below.

         V*      REAL(LDV,N)
                 on return from SSPEV, if the user has set JOB
                 = 0        V is not referenced.
                 = nonzero  the N eigenvectors of A are stored in the
                 first N columns of V.  See also INFO below.

         LDV     INTEGER
                 set by the user to
                 the leading dimension of the array V if JOB is also
                 set nonzero.  In that case, N must be .LE. LDV.
                 If JOB is set to zero, LDV is not referenced.

         WORK*   REAL(2N)
                 temporary storage vector.  Contents changed by SSPEV.

         JOB     INTEGER
                 set by the user to
                 = 0        eigenvalues only to be calculated by SSPEV.
                            Neither V nor LDV are referenced.
                 = nonzero  eigenvalues and vectors to be calculated.
                            In this case, A & V must be distinct arrays.
                            Also, if LDA .GT. LDV, SSPEV changes all the
```

```
                              elements of A thru column N.  If LDA < LDV,
                              SSPEV changes all the elements of V through
                              column N.  If LDA=LDV, only A(I,J) and V(I,
                              J) for I,J = 1,...,N are changed by SSPEV.

        INFO*   INTEGER
                on return from SSPEV, the value of INFO is
                = 0 for normal return.
                = K if the eigenvalue iteration fails to converge.
                    Eigenvalues and vectors 1 through K-1 are correct.


      Error Messages-
           No. 1   recoverable  N is greater than LDV and JOB is nonzero
           No. 2   recoverable  N is less than one

***REFERENCES  (NONE)
***ROUTINES CALLED  IMTQL2, TQLRAT, TRBAK3, TRED3, XERMSG
***REVISION HISTORY  (YYMMDD)
   800808  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   END PROLOGUE
```

SSPFA

```
      SUBROUTINE SSPFA (AP, N, KPVT, INFO)
***BEGIN PROLOGUE  SSPFA
***PURPOSE  Factor a real symmetric matrix stored in packed form by
            elimination with symmetric pivoting.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1A
***TYPE      SINGLE PRECISION (SSPFA-S, DSPFA-D, CHPFA-C, CSPFA-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX FACTORIZATION, PACKED,
             SYMMETRIC
***AUTHOR  Bunch, J., (UCSD)
***DESCRIPTION

      SSPFA factors a real symmetric matrix stored in
      packed form by elimination with symmetric pivoting.

      To solve  A*X = B , follow SSPFA by SSPSL.
      To compute  INVERSE(A)*C , follow SSPFA by SSPSL.
      To compute  DETERMINANT(A) , follow SSPFA by SSPDI.
      To compute  INERTIA(A) , follow SSPFA by SSPDI.
      To compute  INVERSE(A) , follow SSPFA by SSPDI.

      On Entry

         AP      REAL (N*(N+1)/2)
                 the packed form of a symmetric matrix  A .  The
                 columns of the upper triangle are stored sequentially
                 in a one-dimensional array of length  N*(N+1)/2 .
                 See comments below for details.

         N       INTEGER
                 the order of the matrix  A .

      Output

         AP      a block diagonal matrix and the multipliers which
                 were used to obtain it stored in packed form.
                 The factorization can be written  A = U*D*TRANS(U)
                 where  U  is a product of permutation and unit
                 upper triangular matrices , TRANS(U) is the
                 transpose of  U , and  D  is block diagonal
                 with 1 by 1 and 2 by 2 blocks.

         KPVT    INTEGER(N)
                 an integer vector of pivot indices.

         INFO    INTEGER
                 = 0  normal value.
                 = K  if the K-th pivot block is singular.  This is
                      not an error condition for this subroutine,
                      but it does indicate that SSPSL or SSPDI may
                      divide by zero if called.

      Packed Storage

           The following program segment will pack the upper
           triangle of a symmetric matrix.
```

```
      K = 0
      DO 20 J = 1, N
         DO 10 I = 1, J
            K = K + 1
            AP(K)  = A(I,J)
   10    CONTINUE
   20 CONTINUE

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  ISAMAX, SAXPY, SSWAP
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   891107  Modified routine equivalence list.  (WRB)
   891107  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SSPMV

```
      SUBROUTINE SSPMV (UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)
***BEGIN PROLOGUE  SSPMV
***PURPOSE  Perform the matrix-vector operation.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (SSPMV-S, DSPMV-D, CSPMV-C)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  SSPMV  performs the matrix-vector operation

     y := alpha*A*x + beta*y,

  where alpha and beta are scalars, 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  - REAL             .
           On entry, ALPHA specifies the scalar alpha.
           Unchanged on exit.

  AP     - REAL              array of DIMENSION at least
           ( ( n*( n + 1))/2).
           Before entry with UPLO = 'U' or 'u', the array AP must
           contain the upper 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( 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 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 ) respectively, and so on.
            Unchanged on exit.

  X       - REAL             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.

  BETA    - REAL            .
            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       - REAL             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 elements of
            Y. INCY must not be zero.
            Unchanged on exit.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

SSPR

```
      SUBROUTINE SSPR (UPLO, N, ALPHA, X, INCX, AP)
***BEGIN PROLOGUE  SSPR
***PURPOSE  Performs the symmetric rank 1 operation.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (SSPR-S)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  SSPR    performs the symmetric rank 1 operation

     A := alpha*x*x' + A,

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

  X      - REAL              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.
```

```
  AP     - REAL             array of DIMENSION at least
           ( ( n*( n + 1 ) )/2 ).
           Before entry with  UPLO = 'U' or 'u', the array AP must
           contain the upper 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( 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 ) respectively, and so on. On exit, the array
           AP is overwritten by the lower triangular part of the
           updated matrix.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

SSPR2

```
      SUBROUTINE SSPR2 (UPLO, N, ALPHA, X, INCX, Y, INCY, AP)
***BEGIN PROLOGUE  SSPR2
***PURPOSE  Perform the symmetric rank 2 operation.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (SSPR2-S, DSPR2-D, CSPR2-C)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  SSPR2  performs the symmetric rank 2 operation

     A := alpha*x*y' + alpha*y*x' + A,

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

  X      - REAL              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      - REAL             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.

  AP     - REAL             array of DIMENSION at least
           ( ( n*( n + 1 ) )/2 ).
           Before entry with  UPLO = 'U' or 'u', the array AP must
           contain the upper 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( 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 ) respectively, and so on. On exit, the array
           AP is overwritten by the lower triangular part of the
           updated matrix.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

SSPSL

```
      SUBROUTINE SSPSL (AP, N, KPVT, B)
***BEGIN PROLOGUE  SSPSL
***PURPOSE  Solve a real symmetric system using the factors obtained
            from SSPFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1A
***TYPE      SINGLE PRECISION (SSPSL-S, DSPSL-D, CHPSL-C, CSPSL-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX, PACKED, SOLVE, SYMMETRIC
***AUTHOR  Bunch, J., (UCSD)
***DESCRIPTION

      SSISL solves the real symmetric system
      A * X = B
      using the factors computed by SSPFA.

      On Entry

         AP      REAL(N*(N+1)/2)
                 the output from SSPFA.

         N       INTEGER
                 the order of the matrix  A .

         KPVT    INTEGER(N)
                 the pivot vector from SSPFA.

         B       REAL(N)
                 the right hand side vector.

      On Return

         B       the solution vector  X .

      Error Condition

         A division by zero may occur if  SSPCO  has set RCOND .EQ. 0.0
         or  SSPFA  has set INFO .NE. 0  .

      To compute  INVERSE(A) * C  where  C  is a matrix
      with  P  columns
            CALL SSPFA(AP,N,KPVT,INFO)
            IF (INFO .NE. 0) GO TO ...
            DO 10 J = 1, P
               CALL SSPSL(AP,N,KPVT,C(1,J))
         10 CONTINUE

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SDOT
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   891107  Modified routine equivalence list.  (WRB)
   891107  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
```

```
900326  Removed duplicate information from DESCRIPTION section.
        (WRB)
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

SSVDC

```
      SUBROUTINE SSVDC (X, LDX, N, P, S, E, U, LDU, V, LDV, WORK, JOB,
     +   INFO)
***BEGIN PROLOGUE  SSVDC
***PURPOSE  Perform the singular value decomposition of a rectangular
            matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D6
***TYPE      SINGLE PRECISION (SSVDC-S, DSVDC-D, CSVDC-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX,
             SINGULAR VALUE DECOMPOSITION
***AUTHOR  Stewart, G. W., (U. of Maryland)
***DESCRIPTION

      SSVDC is a subroutine to reduce a real NxP matrix X by orthogonal
      transformations U and V to diagonal form.  The elements S(I) are
      the singular values of X.  The columns of U are the corresponding
      left singular vectors, and the columns of V the right singular
      vectors.

      On Entry

          X         REAL(LDX,P), where LDX .GE. N.
                    X contains the matrix whose singular value
                    decomposition is to be computed.  X is
                    destroyed by SSVDC.

          LDX       INTEGER
                    LDX is the leading dimension of the array X.

          N         INTEGER
                    N is the number of rows of the matrix X.

          P         INTEGER
                    P is the number of columns of the matrix X.

          LDU       INTEGER
                    LDU is the leading dimension of the array U.
                    (See below).

          LDV       INTEGER
                    LDV is the leading dimension of the array V.
                    (See below).

          WORK      REAL(N)
                    work is a scratch array.

          JOB       INTEGER
                    JOB controls the computation of the singular
                    vectors.  It has the decimal expansion AB
                    with the following meaning

                         A .EQ. 0  Do not compute the left singular
                                   vectors.
                         A .EQ. 1  Return the N left singular vectors
                                   in U.
                         A .GE. 2  Return the first MIN(N,P) singular
```

```
                                vectors in U.
                     B .EQ. 0  Do not compute the right singular
                                vectors.
                     B .EQ. 1  Return the right singular vectors
                                in V.

     On Return

         S         REAL(MM), where MM=MIN(N+1,P).
                   The first MIN(N,P) entries of S contain the
                   singular values of X arranged in descending
                   order of magnitude.

         E         REAL(P).
                   E ordinarily contains zeros.  However, see the
                   discussion of INFO for exceptions.

         U         REAL(LDU,K), where LDU .GE. N.  If JOBA .EQ. 1, then
                                   K .EQ. N.  If JOBA .GE. 2 , then
                                   K .EQ. MIN(N,P).
                   U contains the matrix of right singular vectors.
                   U is not referenced if JOBA .EQ. 0.  If N .LE. P
                   or if JOBA .EQ. 2, then U may be identified with X
                   in the subroutine call.

         V         REAL(LDV,P), where LDV .GE. P.
                   V contains the matrix of right singular vectors.
                   V is not referenced if JOB .EQ. 0.  If P .LE. N,
                   then V may be identified with X in the
                   subroutine call.

         INFO      INTEGER.
                   the singular values (and their corresponding
                   singular vectors) S(INFO+1),S(INFO+2),...,S(M)
                   are correct (here M=MIN(N,P)).  Thus if
                   INFO .EQ. 0, all the singular values and their
                   vectors are correct.  In any event, the matrix
                   B = TRANS(U)*X*V is the bidiagonal matrix
                   with the elements of S on its diagonal and the
                   elements of E on its super-diagonal (TRANS(U)
                   is the transpose of U).  Thus the singular
                   values of X and B are the same.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SDOT, SNRM2, SROT, SROTG, SSCAL, SSWAP
***REVISION HISTORY  (YYMMDD)
   790319  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890531  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SSWAP

```
      SUBROUTINE SSWAP (N, SX, INCX, SY, INCY)
***BEGIN PROLOGUE  SSWAP
***PURPOSE  Interchange two vectors.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A5
***TYPE      SINGLE PRECISION (SSWAP-S, DSWAP-D, CSWAP-C, ISWAP-I)
***KEYWORDS  BLAS, INTERCHANGE, LINEAR ALGEBRA, VECTOR
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
           Kincaid, D. R., (U. of Texas)
           Krogh, F. T., (JPL)
***DESCRIPTION

                B L A S  Subprogram
    Description of Parameters

     --Input--
        N  number of elements in input vector(s)
       SX  single precision vector with N elements
     INCX  storage spacing between elements of SX
       SY  single precision vector with N elements
     INCY  storage spacing between elements of SY

     --Output--
       SX  input vector SY (unchanged if N .LE. 0)
       SY  input vector SX (unchanged if N .LE. 0)

     Interchange single precision SX and single precision SY.
     For I = 0 to N-1, interchange  SX(LX+I*INCX) and SY(LY+I*INCY),
     where LX = 1 if INCX .GE. 0, else LX = 1+(1-N)*INCX, and LY is
     defined in a similar way using INCY.

***REFERENCES  C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T.
                 Krogh, Basic linear algebra subprograms for Fortran
                 usage, Algorithm No. 539, Transactions on Mathematical
                 Software 5, 3 (September 1979), pp. 308-323.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   791001  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920310  Corrected definition of LX in DESCRIPTION.  (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

SSYMM

```
      SUBROUTINE SSYMM (SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA,
     $   C, LDC)
***BEGIN PROLOGUE  SSYMM
***PURPOSE  Multiply a real general matrix by a real symmetric matrix.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B6
***TYPE      SINGLE PRECISION (SSYMM-S, DSYMM-D, CSYMM-C)
***KEYWORDS  LEVEL 3 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J., (ANL)
           Duff, I., (AERE)
           Du Croz, J., (NAG)
           Hammarling, S. (NAG)
***DESCRIPTION

  SSYMM  performs one of the matrix-matrix operations

     C := alpha*A*B + beta*C,

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

A - REAL array of DIMENSION (LDA, ka), where ka is 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 leading m by m upper triangular part of the array A must contain the upper triangular part of the symmetric 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 leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric 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 - REAL 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 - REAL .
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 - REAL 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.

***REFERENCES  Dongarra, J., Du Croz, J., Duff, I., and Hammarling, S.
                 A set of level 3 basic linear algebra subprograms.
                 ACM TOMS, Vol. 16, No. 1, pp. 1-17, March 1990.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   890208  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

SSYMV

```
      SUBROUTINE SSYMV (UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
***BEGIN PROLOGUE  SSYMV
***PURPOSE  Multiply a real vector by a real symmetric matrix.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (SSYMV-S, DSYMV-D, CSYMV-C)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  SSYMV  performs the matrix-vector  operation

     y := alpha*A*x + beta*y,

  where alpha and beta are scalars, 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  - REAL             .
           On entry, ALPHA specifies the scalar alpha.
           Unchanged on exit.

  A      - REAL             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 contain 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 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. LDA must be at least
           max( 1, n ).
           Unchanged on exit.

  X      - REAL             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.

  BETA   - REAL            .
           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      - REAL             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 elements of
           Y. INCY must not be zero.
           Unchanged on exit.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

SSYR

```
      SUBROUTINE SSYR (UPLO, N, ALPHA, X, INCX, A, LDA)
***BEGIN PROLOGUE  SSYR
***PURPOSE  Perform symmetric rank 1 update of a real symmetric matrix.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (SSYR-S)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  SSYR   performs the symmetric rank 1 operation

     A := alpha*x*x' + A,

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

  X      - REAL              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.
```

```
  A      - REAL             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 contain 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 triangular 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.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

SSYR2

```
     SUBROUTINE SSYR2 (UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)
***BEGIN PROLOGUE  SSYR2
***PURPOSE  Perform symmetric rank 2 update of a real symmetric matrix.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (SSYR2-S, DSYR2-D, CSYR2-C)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  SSYR2  performs the symmetric rank 2 operation

     A := alpha*x*y' + alpha*y*x' + A,

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

  X      - REAL              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      - REAL             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      - REAL             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 contain 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 triangular 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.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

SSYR2K

```
      SUBROUTINE SSYR2K (UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA,
     $   C, LDC)
***BEGIN PROLOGUE  SSYR2K
***PURPOSE  Perform symmetric rank 2k update of a real symmetric matrix
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B6
***TYPE      SINGLE PRECISION (SSYR2-S, DSYR2-D, CSYR2-C, SSYR2K-S)
***KEYWORDS  LEVEL 3 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J., (ANL)
           Duff, I., (AERE)
           Du Croz, J., (NAG)
           Hammarling, S. (NAG)
***DESCRIPTION

  SSYR2K  performs one of the symmetric rank 2k operations

     C := alpha*A*B' + alpha*B*A' + beta*C,

  or

     C := alpha*A'*B + alpha*B'*A + beta*C,

  where  alpha and beta  are scalars, C is an  n by n  symmetric 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 performed 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 - REAL .
On entry, ALPHA specifies the scalar alpha.
Unchanged on exit.

A - REAL array of DIMENSION (LDA, ka), where ka is 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.

B - REAL array of DIMENSION (LDB, kb), where kb is 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 - REAL .
On entry, BETA specifies the scalar beta.
Unchanged on exit.

C - REAL 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.

***REFERENCES  Dongarra, J., Du Croz, J., Duff, I., and Hammarling, S.
                 A set of level 3 basic linear algebra subprograms.
                 ACM TOMS, Vol. 16, No. 1, pp. 1-17, March 1990.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   890208  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

SSYRK

```
      SUBROUTINE SSYRK (UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)
***BEGIN PROLOGUE  SSYRK
***PURPOSE  Perform symmetric rank k update of a real symmetric matrix.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B6
***TYPE      SINGLE PRECISION (SSYRK-S, DSYRK-D, CSYRK-C)
***KEYWORDS  LEVEL 3 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J., (ANL)
           Duff, I., (AERE)
           Du Croz, J., (NAG)
           Hammarling, S. (NAG)
***DESCRIPTION

  SSYRK  performs one of the symmetric rank k operations

     C := alpha*A*A' + beta*C,

  or

     C := alpha*A'*A + beta*C,

  where  alpha and beta  are scalars, C is an  n by n  symmetric 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 performed 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   - REAL            .
            On entry, ALPHA specifies the scalar alpha.
            Unchanged on exit.

  A       - REAL             array of DIMENSION ( LDA, ka ), where ka is
            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    - REAL            .
            On entry, BETA specifies the scalar beta.
            Unchanged on exit.

  C       - REAL             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.

***REFERENCES  Dongarra, J., Du Croz, J., Duff, I., and Hammarling, S.
                 A set of level 3 basic linear algebra subprograms.
                 ACM TOMS, Vol. 16, No. 1, pp. 1-17, March 1990.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   890208  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
```

```
        lines were modified.  (BKS)
END PROLOGUE
```

STBMV

```
      SUBROUTINE STBMV (UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
***BEGIN PROLOGUE  STBMV
***PURPOSE  Multiply a real vector by a real triangular band matrix.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (STBMV-S, DTBMV-D, CTBMV-C)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  STBMV  performs one of the matrix-vector operations

     x := A*x,   or   x := A'*x,

  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 performed 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 triangular.

              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.

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 - REAL 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       - REAL             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.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

STBSV

```
      SUBROUTINE STBSV (UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
***BEGIN PROLOGUE  STBSV
***PURPOSE  Solve a real triangular banded system of linear equations.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (STBSV-S, DTBSV-D, CTBSV-C)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  STBSV  solves one of the systems of equations

     A*x = b,   or   A'*x = b,

  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 triangular.

              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.

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 - REAL 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       - REAL             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.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

STEPS

```
      SUBROUTINE STEPS (F, NEQN, Y, X, H, EPS, WT, START, HOLD, K,
     +   KOLD, CRASH, PHI, P, YP, PSI, ALPHA, BETA, SIG, V, W, G,
     +   PHASE1, NS, NORND, KSTEPS, TWOU, FOURU, XOLD, KPREV, IVC, IV,
     +   KGI, GI, RPAR, IPAR)
***BEGIN PROLOGUE  STEPS
***PURPOSE  Integrate a system of first order ordinary differential
            equations one step.
***LIBRARY   SLATEC (DEPAC)
***CATEGORY  I1A1B
***TYPE      SINGLE PRECISION (STEPS-S, DSTEPS-D)
***KEYWORDS  ADAMS METHOD, DEPAC, INITIAL VALUE PROBLEMS, ODE,
             ORDINARY DIFFERENTIAL EQUATIONS, PREDICTOR-CORRECTOR
***AUTHOR  Shampine, L. F., (SNLA)
           Gordon, M. K., (SNLA)
             MODIFIED BY H.A. WATTS
***DESCRIPTION

   Written by L. F. Shampine and M. K. Gordon

   Abstract

   Subroutine  STEPS  is normally used indirectly through subroutine
   DEABM .  Because  DEABM  suffices for most problems and is much
   easier to use, using it should be considered before using  STEPS
   alone.

   Subroutine STEPS integrates a system of  NEQN  first order ordinary
   differential equations one step, normally from X to X+H, using a
   modified divided difference form of the Adams Pece formulas.  Local
   extrapolation is used to improve absolute stability and accuracy.
   The code adjusts its order and step size to control the local error
   per unit step in a generalized sense.  Special devices are included
   to control roundoff error and to detect when the user is requesting
   too much accuracy.

   This code is completely explained and documented in the text,
   Computer Solution of Ordinary Differential Equations, The Initial
   Value Problem  by L. F. Shampine and M. K. Gordon.
   Further details on use of this code are available in "Solving
   Ordinary Differential Equations with ODE, STEP, and INTRP",
   by L. F. Shampine and M. K. Gordon, SLA-73-1060.


   The parameters represent --
      F -- subroutine to evaluate derivatives
      NEQN -- number of equations to be integrated
      Y(*) -- solution vector at X
      X -- independent variable
      H -- appropriate step size for next step.  Normally determined by
           code
      EPS -- local error tolerance
      WT(*) -- vector of weights for error criterion
      START -- logical variable set .TRUE. for first step,  .FALSE.
           otherwise
      HOLD -- step size used for last successful step
      K -- appropriate order for next step (determined by code)
```

```
   KOLD -- order used for last successful step
   CRASH -- logical variable set .TRUE. when no step can be taken,
        .FALSE. otherwise.
   YP(*) -- derivative of solution vector at  X  after successful
        step
   KSTEPS -- counter on attempted steps
   TWOU -- 2.*U where U is machine unit roundoff quantity
   FOURU -- 4.*U where U is machine unit roundoff quantity
   RPAR,IPAR -- parameter arrays which you may choose to use
         for communication between your program and subroutine F.
         They are not altered or used by STEPS.
The variables X,XOLD,KOLD,KGI and IVC and the arrays Y,PHI,ALPHA,G,
W,P,IV and GI are required for the interpolation subroutine SINTRP.
The remaining variables and arrays are included in the call list
only to eliminate local retention of variables between calls.

Input to STEPS

   First call --

The user must provide storage in his calling program for all arrays
in the call list, namely

  DIMENSION Y(NEQN),WT(NEQN),PHI(NEQN,16),P(NEQN),YP(NEQN),PSI(12),
 1  ALPHA(12),BETA(12),SIG(13),V(12),W(12),G(13),GI(11),IV(10),
 2  RPAR(*),IPAR(*)

 **Note**

The user must also declare  START ,  CRASH ,  PHASE1  and  NORND
logical variables and  F  an EXTERNAL subroutine, supply the
subroutine  F(X,Y,YP)  to evaluate
   DY(I)/DX = YP(I) = F(X,Y(1),Y(2),...,Y(NEQN))
and initialize only the following parameters.
   NEQN -- number of equations to be integrated
   Y(*) -- vector of initial values of dependent variables
   X -- initial value of the independent variable
   H -- nominal step size indicating direction of integration
        and maximum size of step.  Must be variable
   EPS -- local error tolerance per step.  Must be variable
   WT(*) -- vector of non-zero weights for error criterion
   START -- .TRUE.
   YP(*) -- vector of initial derivative values
   KSTEPS -- set KSTEPS to zero
   TWOU -- 2.*U where U is machine unit roundoff quantity
   FOURU -- 4.*U where U is machine unit roundoff quantity
Define U to be the machine unit roundoff quantity by calling
the function routine  R1MACH,  U = R1MACH(4), or by
computing U so that U is the smallest positive number such
that 1.0+U .GT. 1.0.

STEPS  requires that the L2 norm of the vector with components
LOCAL ERROR(L)/WT(L)  be less than  EPS  for a successful step.  The
array  WT  allows the user to specify an error test appropriate
for his problem.  For example,
   WT(L) = 1.0  specifies absolute error,
         = ABS(Y(L))  error relative to the most recent value of the
              L-th component of the solution,
         = ABS(YP(L))  error relative to the most recent value of
              the L-th component of the derivative,
```

```
            = MAX(WT(L),ABS(Y(L)))  error relative to the largest
                 magnitude of L-th component obtained so far,
            = ABS(Y(L))*RELERR/EPS + ABSERR/EPS  specifies a mixed
                 relative-absolute test where  RELERR  is relative
                 error,  ABSERR  is absolute error and  EPS =
                 MAX(RELERR,ABSERR) .

      Subsequent calls --

   Subroutine  STEPS  is designed so that all information needed to
   continue the integration, including the step size  H  and the order
   K , is returned with each step.  With the exception of the step
   size, the error tolerance, and the weights, none of the parameters
   should be altered.  The array  WT  must be updated after each step
   to maintain relative error tests like those above.  Normally the
   integration is continued just beyond the desired endpoint and the
   solution interpolated there with subroutine  SINTRP .  If it is
   impossible to integrate beyond the endpoint, the step size may be
   reduced to hit the endpoint since the code will not take a step
   larger than the  H  input.  Changing the direction of integration,
   i.e., the sign of  H , requires the user set  START = .TRUE. before
   calling  STEPS  again.  This is the only situation in which  START
   should be altered.

   Output from STEPS

      Successful Step --

   The subroutine returns after each successful step with  START  and
   CRASH  set .FALSE. .  X  represents the independent variable
   advanced one step of length  HOLD  from its value on input and  Y
   the solution vector at the new value of  X .  All other parameters
   represent information corresponding to the new  X  needed to
   continue the integration.

      Unsuccessful Step --

   When the error tolerance is too small for the machine precision,
   the subroutine returns without taking a step and  CRASH = .TRUE. .
   An appropriate step size and error tolerance for continuing are
   estimated and all other information is restored as upon input
   before returning.  To continue with the larger tolerance, the user
   just calls the code again.  A restart is neither required nor
   desirable.

***REFERENCES  L. F. Shampine and M. K. Gordon, Solving ordinary
                 differential equations with ODE, STEP, and INTRP,
                 Report SLA-73-1060, Sandia Laboratories, 1973.
***ROUTINES CALLED  HSTART, R1MACH
***REVISION HISTORY  (YYMMDD)
   740101  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

STIN

```
      SUBROUTINE STIN (N, NELT, IA, JA, A, ISYM, SOLN, RHS, IUNIT, JOB)
***BEGIN PROLOGUE  STIN
***PURPOSE  Read in SLAP Triad Format Linear System.
            Routine to read in a SLAP Triad format matrix and right
            hand side and solution to the system, if known.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  N1
***TYPE      SINGLE PRECISION (STIN-S, DTIN-D)
***KEYWORDS  DIAGNOSTICS, LINEAR SYSTEM, SLAP SPARSE
***AUTHOR  Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, IUNIT, JOB
     REAL    A(NELT), SOLN(N), RHS(N)

     CALL STIN( N, NELT, IA, JA, A, ISYM, SOLN, RHS, IUNIT, JOB )

 *Arguments:
 N       :OUT      Integer
          Order of the Matrix.
 NELT    :INOUT    Integer.
          On input NELT is the maximum number of non-zeros that
          can be stored in the IA, JA, A arrays.
          On output NELT is the number of non-zeros stored in A.
 IA      :OUT      Integer IA(NELT).
 JA      :OUT      Integer JA(NELT).
 A       :OUT      Real A(NELT).
          On output these arrays hold the matrix A in the SLAP
          Triad format.  See "Description", below.
 ISYM    :OUT      Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the lower
          triangle of the matrix is stored.
 SOLN    :OUT      Real SOLN(N).
          The solution to the linear system, if present.  This array
          is accessed if and only if JOB to read it in, see below.
          If the user requests that SOLN be read in, but it is not in
          the file, then it is simply zeroed out.
 RHS     :OUT      Real RHS(N).
          The right hand side vector.  This array is accessed if and
          only if JOB is set to read it in, see below.
          If the user requests that RHS be read in, but it is not in
          the file, then it is simply zeroed out.
 IUNIT   :IN       Integer.
          Fortran logical I/O device unit number to write the matrix
          to.  This unit must be connected in a system dependent fashion
          to a file or the console or you will get a nasty message
          from the Fortran I/O libraries.
 JOB     :INOUT    Integer.
          Flag indicating what I/O operations to perform.
```

On input JOB indicates what Input operations to try to perform.
JOB = 0 => Read only the matrix.
JOB = 1 => Read matrix and RHS (if present).
JOB = 2 => Read matrix and SOLN (if present).
JOB = 3 => Read matrix, RHS and SOLN (if present).
On output JOB indicates what operations were actually performed.
JOB = 0 => Read in only the matrix.
JOB = 1 => Read in the matrix and RHS.
JOB = 2 => Read in the matrix and SOLN.
JOB = 3 => Read in the matrix, RHS and SOLN.

*Description:

The format for the input is as follows. On the first line are counters and flags: N, NELT, ISYM, IRHS, ISOLN. N, NELT and ISYM are described above. IRHS is a flag indicating if the RHS was written out (1 is yes, 0 is no). ISOLN is a flag indicating if the SOLN was written out (1 is yes, 0 is no). The format for the fist line is: 5i10. Then comes the NELT Triad's IA(I), JA(I) and A(I), I = 1, NELT. The format for these lines is : 1X,I5,1X,I5,1X,E16.7. Then comes RHS(I), I = 1, N, if IRHS = 1. Then comes SOLN(I), I = 1, N, if ISOLN = 1. The format for these lines is: 1X,E16.7.

========== S L A P Triad format ==========
This routine requires that the matrix A be stored in the SLAP Triad format. In this format only the non-zeros are stored. They may appear in *ANY* order. The user supplies three arrays of length NELT, where NELT is the number of non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)). For each non-zero the user puts the row and column index of that matrix element in the IA and JA arrays. The value of the non-zero matrix element is placed in the corresponding location of the A array. This is an extremely easy data structure to generate. On the other hand it is not too efficient on vector computers for the iterative solution of linear systems. Hence, SLAP changes this input data structure to the SLAP Column format for the iteration (but does not change it back).

Here is an example of the SLAP Triad storage format for a 5x5 Matrix. Recall that the entries may appear in any order.

```
    5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                       1  2  3  4  5  6  7  8  9 10 11
|11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
|21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
| 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
| 0  0  0 44  0|
|51  0 53  0 55|
```

*Cautions:

This routine will attempt to write to the Fortran logical output unit IUNIT, if IUNIT .ne. 0. Thus, the user must make sure that this logical unit is attached to a file or terminal before calling this routine with a non-zero value for IUNIT. This routine does not check for the validity of a non-zero IUNIT unit number.

***REFERENCES (NONE)

***ROUTINES CALLED (NONE)

```
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920511  Added complete declaration section.  (WRB)
   930701  Updated CATEGORY section.  (FNF, WRB)
   END PROLOGUE
```

STOUT

```
      SUBROUTINE STOUT (N, NELT, IA, JA, A, ISYM, SOLN, RHS, IUNIT, JOB)
***BEGIN PROLOGUE  STOUT
***PURPOSE  Write out SLAP Triad Format Linear System.
            Routine to write out a SLAP Triad format matrix and right
            hand side and solution to the system, if known.
***LIBRARY   SLATEC (SLAP)
***CATEGORY  N1
***TYPE      SINGLE PRECISION (STOUT-S, DTOUT-D)
***KEYWORDS  DIAGNOSTICS, LINEAR SYSTEM, SLAP SPARSE
***AUTHOR  Seager, Mark K., (LLNL)
             Lawrence Livermore National Laboratory
             PO BOX 808, L-60
             Livermore, CA 94550 (510) 423-3141
             seager@llnl.gov
***DESCRIPTION

 *Usage:
     INTEGER N, NELT, IA(NELT), JA(NELT), ISYM, IUNIT, JOB
     REAL    A(NELT), SOLN(N), RHS(N)

     CALL STOUT( N, NELT, IA, JA, A, ISYM, SOLN, RHS, IUNIT, JOB )

 *Arguments:
 N       :IN       Integer
          Order of the Matrix.
 NELT    :IN       Integer.
          Number of non-zeros stored in A.
 IA      :IN       Integer IA(NELT).
 JA      :IN       Integer JA(NELT).
 A       :IN       Real A(NELT).
          These arrays should hold the matrix A in the SLAP
          Triad format.  See "Description", below.
 ISYM    :IN       Integer.
          Flag to indicate symmetric storage format.
          If ISYM=0, all non-zero entries of the matrix are stored.
          If ISYM=1, the matrix is symmetric, and only the lower
          triangle of the matrix is stored.
 SOLN    :IN       Real SOLN(N).
          The solution to the linear system, if known.  This array
          is accessed if and only if JOB is set to print it out,
          see below.
 RHS     :IN       Real RHS(N).
          The right hand side vector.  This array is accessed if and
          only if JOB is set to print it out, see below.
 IUNIT   :IN       Integer.
          Fortran logical I/O device unit number to write the matrix
          to.  This unit must be connected in a system dependent fashion
          to a file or the console or you will get a nasty message
          from the Fortran I/O libraries.
 JOB     :IN       Integer.
          Flag indicating what I/O operations to perform.
          JOB = 0 => Print only the matrix.
              = 1 => Print matrix and RHS.
              = 2 => Print matrix and SOLN.
              = 3 => Print matrix, RHS and SOLN.
```

```
*Description:
      The format for the output is as follows.  On  the first line
      are counters and flags: N, NELT, ISYM, IRHS, ISOLN.  N, NELT
      and ISYM are described above.  IRHS is  a flag indicating if
      the RHS was  written out (1 is  yes, 0 is  no).  ISOLN  is a
      flag indicating if the SOLN was written out  (1 is yes, 0 is
      no).  The format for the fist line is: 5i10.  Then comes the
      NELT Triad's IA(I), JA(I) and A(I), I = 1, NELT.  The format
      for  these lines is   :  1X,I5,1X,I5,1X,E16.7.   Then  comes
      RHS(I), I = 1, N, if IRHS = 1.  Then  comes SOLN(I), I  = 1,
      N, if ISOLN = 1.  The format for these lines is: 1X,E16.7.

      ==================== S L A P Triad format ====================
      This routine requires that the  matrix A be   stored in  the
      SLAP  Triad format.  In  this format only the non-zeros  are
      stored.  They may appear in  *ANY* order.  The user supplies
      three arrays of  length NELT, where  NELT is  the number  of
      non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)).  For
      each non-zero the user puts the row and column index of that
      matrix element  in the IA and  JA arrays.  The  value of the
      non-zero  matrix  element is  placed   in  the corresponding
      location of the A array.   This is  an  extremely  easy data
      structure to generate.  On  the  other hand it   is  not too
      efficient on vector computers for  the iterative solution of
      linear systems.  Hence,   SLAP changes   this  input    data
      structure to the SLAP Column format  for  the iteration (but
      does not change it back).

      Here is an example of the  SLAP Triad   storage format for a
      5x5 Matrix.  Recall that the entries may appear in any order.

          5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
                                1  2  3  4  5  6  7  8  9 10 11
      |11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
      |21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
      | 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
      | 0  0  0 44  0|
      |51  0 53  0 55|

*Cautions:
    This routine will attempt to write to the Fortran logical output
    unit IUNIT, if IUNIT .ne. 0.  Thus, the user must make sure that
    this logical unit is attached to a file or terminal before calling
    this routine with a non-zero value for IUNIT.  This routine does
    not check for the validity of a non-zero IUNIT unit number.
***REFERENCES  (NONE)
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   871119  DATE WRITTEN
   881213  Previous REVISION DATE
   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
   890922  Numerous changes to prologue to make closer to SLATEC
           standard.  (FNF)
   890929  Numerous changes to reduce SP/DP differences.  (FNF)
   910411  Prologue converted to Version 4.0 format.  (BAB)
   920511  Added complete declaration section.  (WRB)
   930701  Updated CATEGORY section.  (FNF, WRB)
   END PROLOGUE
```

STPMV

```
      SUBROUTINE STPMV (UPLO, TRANS, DIAG, N, AP, X, INCX)
***BEGIN PROLOGUE  STPMV
***PURPOSE  Perform one of the matrix-vector operations.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (STPMV-S, DTPMV-D, CTPMV-C)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  STPMV  performs one of the matrix-vector operations

     x := A*x,   or   x := A'*x,

  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 performed 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 triangular.

              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     - REAL             array of DIMENSION at least
           ( ( n*( n + 1))/2).
           Before entry with  UPLO = 'U' or 'u', the array AP must
           contain the upper triangular 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 ) contains 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      - REAL             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.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

STPSV

```
      SUBROUTINE STPSV (UPLO, TRANS, DIAG, N, AP, X, INCX)
***BEGIN PROLOGUE  STPSV
***PURPOSE  Solve one of the systems of equations.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (STPSV-S, DTPSV-D, CTPSV-C)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  STPSV  solves one of the systems of equations

     A*x = b,   or   A'*x = b,

  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 triangular.

              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      - REAL             array of DIMENSION at least
            ( ( n*( n + 1))/2).
            Before entry with  UPLO = 'U' or 'u', the array AP must
            contain the upper triangular 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 ) contains 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       - REAL             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.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

STRCO

```
      SUBROUTINE STRCO (T, LDT, N, RCOND, Z, JOB)
***BEGIN PROLOGUE  STRCO
***PURPOSE  Estimate the condition number of a triangular matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A3
***TYPE      SINGLE PRECISION (STRCO-S, DTRCO-D, CTRCO-C)
***KEYWORDS  CONDITION NUMBER, LINEAR ALGEBRA, LINPACK,
             TRIANGULAR MATRIX
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      STRCO estimates the condition of a real triangular matrix.

      On Entry

         T       REAL(LDT,N)
                 T contains the triangular matrix.  The zero
                 elements of the matrix are not referenced, and
                 the corresponding elements of the array can be
                 used to store other information.

         LDT     INTEGER
                 LDT is the leading dimension of the array T.

         N       INTEGER
                 N is the order of the system.

         JOB     INTEGER
                 = 0         T  is lower triangular.
                 = nonzero   T  is upper triangular.

      On Return

         RCOND   REAL
                 an estimate of the reciprocal condition of  T .
                 For the system  T*X = B , relative perturbations
                 in  T  and  B  of size  EPSILON  may cause
                 relative perturbations in  X  of size  EPSILON/RCOND .
                 If  RCOND  is so small that the logical expression
                           1.0 + RCOND .EQ. 1.0
                 is true, then  T  may be singular to working
                 precision.  In particular,  RCOND  is zero  if
                 exact singularity is detected or the estimate
                 underflows.

         Z       REAL(N)
                 a work vector whose contents are usually unimportant.
                 If  T  is close to a singular matrix, then  Z  is
                 an approximate null vector in the sense that
                 NORM(A*Z) = RCOND*NORM(A)*NORM(Z) .

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SASUM, SAXPY, SSCAL
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
```

```
890531  Changed all specific intrinsics to generic.  (WRB)
890831  Modified array declarations.  (WRB)
890831  REVISION DATE from Version 3.2
891214  Prologue converted to Version 4.0 format.  (BAB)
900326  Removed duplicate information from DESCRIPTION section.
        (WRB)
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

STRDI

```
      SUBROUTINE STRDI (T, LDT, N, DET, JOB, INFO)
***BEGIN PROLOGUE  STRDI
***PURPOSE  Compute the determinant and inverse of a triangular matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A3, D3A3
***TYPE      SINGLE PRECISION (STRDI-S, DTRDI-D, CTRDI-C)
***KEYWORDS  DETERMINANT, INVERSE, LINEAR ALGEBRA, LINPACK, MATRIX,
             TRIANGULAR
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      STRDI computes the determinant and inverse of a real
      triangular matrix.

      On Entry

         T       REAL(LDT,N)
                 T contains the triangular matrix.  The zero
                 elements of the matrix are not referenced, and
                 the corresponding elements of the array can be
                 used to store other information.

         LDT     INTEGER
                 LDT is the leading dimension of the array T.

         N       INTEGER
                 N is the order of the system.

         JOB     INTEGER
                 = 010       no det, inverse of lower triangular.
                 = 011       no det, inverse of upper triangular.
                 = 100       det, no inverse.
                 = 110       det, inverse of lower triangular.
                 = 111       det, inverse of upper triangular.

      On Return

         T       inverse of original matrix if requested.
                 Otherwise unchanged.

         DET     REAL(2)
                 determinant of original matrix if requested.
                 Otherwise not referenced.
                 Determinant = DET(1) * 10.0**DET(2)
                 with  1.0 .LE. ABS(DET(1)) .LT. 10.0
                 or  DET(1) .EQ. 0.0 .

         INFO    INTEGER
                 INFO contains zero if the system is nonsingular
                 and the inverse is requested.
                 Otherwise INFO contains the index of
                 a zero diagonal element of T.

***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SSCAL
```

```
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

STRMM

```
      SUBROUTINE STRMM (SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA,
     $   B, LDB)
***BEGIN PROLOGUE  STRMM
***PURPOSE  Multiply a real general matrix by a real triangular matrix.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B6
***TYPE      SINGLE PRECISION (STRMM-S, DTRMM-D, CTRMM-C)
***KEYWORDS  LEVEL 3 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J., (ANL)
           Duff, I., (AERE)
           Du Croz, J., (NAG)
           Hammarling, S. (NAG)
***DESCRIPTION

  STRMM  performs 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'.

  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 multiplication 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 triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

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 - REAL .
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.

A - REAL array of DIMENSION (LDA, k), where k is m 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 referenced.
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 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. 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 - REAL 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.

```
***REFERENCES  Dongarra, J., Du Croz, J., Duff, I., and Hammarling, S.
                 A set of level 3 basic linear algebra subprograms.
                 ACM TOMS, Vol. 16, No. 1, pp. 1-17, March 1990.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   890208  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

STRMV

```
      SUBROUTINE STRMV (UPLO, TRANS, DIAG, N, A, LDA, X, INCX)
***BEGIN PROLOGUE  STRMV
***PURPOSE  Multiply a real vector by a real triangular matrix.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (STRMV-S, DTRMV-D, CTRMV-C)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  STRMV  performs one of the matrix-vector operations

     x := A*x,   or   x := A'*x,

  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 performed 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 triangular.

              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.

  A      - REAL             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 contain 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 triangular 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      - REAL             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.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

STRSL

```
      SUBROUTINE STRSL (T, LDT, N, B, JOB, INFO)
***BEGIN PROLOGUE  STRSL
***PURPOSE  Solve a system of the form  T*X=B or TRANS(T)*X=B, where
            T is a triangular matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A3
***TYPE      SINGLE PRECISION (STRSL-S, DTRSL-D, CTRSL-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, TRIANGULAR LINEAR SYSTEM,
             TRIANGULAR MATRIX
***AUTHOR  Stewart, G. W., (U. of Maryland)
***DESCRIPTION

     STRSL solves systems of the form

                   T * X = B
     or
                   TRANS(T) * X = B

     where T is a triangular matrix of order N.  Here TRANS(T)
     denotes the transpose of the matrix T.

     On Entry

         T         REAL(LDT,N)
                   T contains the matrix of the system.  The zero
                   elements of the matrix are not referenced, and
                   the corresponding elements of the array can be
                   used to store other information.

         LDT       INTEGER
                   LDT is the leading dimension of the array T.

         N         INTEGER
                   N is the order of the system.

         B         REAL(N).
                   B contains the right hand side of the system.

         JOB       INTEGER
                   JOB specifies what kind of system is to be solved.
                   If JOB is

                        00   solve T*X=B, T lower triangular,
                        01   solve T*X=B, T upper triangular,
                        10   solve TRANS(T)*X=B, T lower triangular,
                        11   solve TRANS(T)*X=B, T upper triangular.

     On Return

         B         B contains the solution, if INFO .EQ. 0.
                   Otherwise B is unaltered.

         INFO      INTEGER
                   INFO contains zero if the system is nonsingular.
                   Otherwise INFO contains the index of
                   the first zero diagonal element of T.
```

```
***REFERENCES  J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W.
                 Stewart, LINPACK Users' Guide, SIAM, 1979.
***ROUTINES CALLED  SAXPY, SDOT
***REVISION HISTORY  (YYMMDD)
   780814  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

STRSM

```
      SUBROUTINE STRSM (SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA,
     $   B, LDB)
***BEGIN PROLOGUE  STRSM
***PURPOSE  Solve a real triangular system of equations with multiple
            right-hand sides.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B6
***TYPE      SINGLE PRECISION (STRSM-S, DTRSM-D, CTRSM-C)
***KEYWORDS  LEVEL 3 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J., (ANL)
           Duff, I., (AERE)
           Du Croz, J., (NAG)
           Hammarling, S. (NAG)
***DESCRIPTION

  STRSM  solves one of the matrix equations

     op( A )*X = alpha*B,   or   X*op( A ) = alpha*B,

  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 multiplication 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 triangular.

            DIAG = 'N' or 'n'   A is not assumed to be unit
                                triangular.

          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   - REAL            .
          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.

A       - REAL             array of DIMENSION ( LDA, k ), where k is m
          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 referenced.
          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 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.  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       - REAL             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.

***REFERENCES  Dongarra, J., Du Croz, J., Duff, I., and Hammarling, S.
                 A set of level 3 basic linear algebra subprograms.
                 ACM TOMS, Vol. 16, No. 1, pp. 1-17, March 1990.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   890208  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

STRSV

```
      SUBROUTINE STRSV (UPLO, TRANS, DIAG, N, A, LDA, X, INCX)
***BEGIN PROLOGUE  STRSV
***PURPOSE  Solve a real triangular system of linear equations.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      SINGLE PRECISION (STRSV-S, DTRSV-D, CTRSV-C)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

  STRSV  solves one of the systems of equations

     A*x = b,   or   A'*x = b,

  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
                                  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.

  A       - REAL             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 contain 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 triangular 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       - REAL             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.

***REFERENCES  Dongarra, J. J., Du Croz, J., Hammarling, S., and
                 Hanson, R. J.  An extended set of Fortran basic linear
                 algebra subprograms.  ACM TOMS, Vol. 14, No. 1,
                 pp. 1-17, March 1988.
***ROUTINES CALLED  LSAME, XERBLA
***REVISION HISTORY  (YYMMDD)
   861022  DATE WRITTEN
   910605  Modified to meet SLATEC prologue standards.  Only comment
           lines were modified.  (BKS)
   END PROLOGUE
```

TINVIT

```
      SUBROUTINE TINVIT (NM, N, D, E, E2, M, W, IND, Z, IERR, RV1, RV2,
     +   RV3, RV4, RV6)
***BEGIN PROLOGUE  TINVIT
***PURPOSE  Compute the eigenvectors of symmetric tridiagonal matrix
            corresponding to specified eigenvalues, using inverse
            iteration.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4C3
***TYPE      SINGLE PRECISION (TINVIT-S)
***KEYWORDS  EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine is a translation of the inverse iteration tech-
      nique in the ALGOL procedure TRISTURM by Peters and Wilkinson.
      HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 418-439(1971).

      This subroutine finds those eigenvectors of a TRIDIAGONAL
      SYMMETRIC matrix corresponding to specified eigenvalues,
      using inverse iteration.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameter, Z, as declared in the calling program
           dimension statement.  NM is an INTEGER variable.

         N is the order of the matrix.  N is an INTEGER variable.
           N must be less than or equal to NM.

         D contains the diagonal elements of the symmetric tridiagonal
           matrix.  D is a one-dimensional REAL array, dimensioned D(N).

         E contains the subdiagonal elements of the symmetric
           tridiagonal matrix in its last N-1 positions.  E(1) is
           arbitrary.  E is a one-dimensional REAL array, dimensioned
           E(N).

         E2 contains the squares of the corresponding elements of E,
           with zeros corresponding to negligible elements of E.
           E(I) is considered negligible if it is not larger than
           the product of the relative machine precision and the sum
           of the magnitudes of D(I) and D(I-1).  E2(1) must contain
           0.0e0 if the eigenvalues are in ascending order, or 2.0e0
           if the eigenvalues are in descending order.  If  BISECT,
           TRIDIB, or  IMTQLV  has been used to find the eigenvalues,
           their output E2 array is exactly what is expected here.
           E2 is a one-dimensional REAL array, dimensioned E2(N).

         M is the number of specified eigenvalues for which eigenvectors
           are to be determined.  M is an INTEGER variable.

         W contains the M eigenvalues in ascending or descending order.
           W is a one-dimensional REAL array, dimensioned W(M).

         IND contains in its first M positions the submatrix indices
```

```
          associated with the corresponding eigenvalues in W --
          1 for eigenvalues belonging to the first submatrix from
          the top, 2 for those belonging to the second submatrix, etc.
          If  BISECT  or  TRIDIB  has been used to determine the
          eigenvalues, their output IND array is suitable for input
          to TINVIT.  IND is a one-dimensional INTEGER array,
          dimensioned IND(M).

     On Output

       ** All input arrays are unaltered.**

        Z contains the associated set of orthonormal eigenvectors.
          Any vector which fails to converge is set to zero.
          Z is a two-dimensional REAL array, dimensioned Z(NM,M).

        IERR is an INTEGER flag set to
          Zero       for normal return,
          -J         if the eigenvector corresponding to the J-th
                     eigenvalue fails to converge in 5 iterations.

        RV1, RV2 and RV3 are one-dimensional REAL arrays used for
          temporary storage.  They are used to store the main diagonal
          and the two adjacent diagonals of the triangular matrix
          produced in the inverse iteration process.  RV1, RV2 and
          RV3 are dimensioned RV1(N), RV2(N) and RV3(N).

        RV4 and RV6 are one-dimensional REAL arrays used for temporary
          storage.  RV4 holds the multipliers of the Gaussian
          elimination process.  RV6 holds the approximate eigenvectors
          in this process.  RV4 and RV6 are dimensioned RV4(N) and
          RV6(N).

     Questions and comments should be directed to B. S. Garbow,
     APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
     ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

TQL1

```
      SUBROUTINE TQL1 (N, D, E, IERR)
***BEGIN PROLOGUE  TQL1
***PURPOSE  Compute the eigenvalues of symmetric tridiagonal matrix by
            the QL method.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4A5, D4C2A
***TYPE      SINGLE PRECISION (TQL1-S)
***KEYWORDS  EIGENVALUES OF A SYMMETRIC TRIDIAGONAL MATRIX, EISPACK,
             QL METHOD
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine is a translation of the ALGOL procedure TQL1,
      NUM. MATH. 11, 293-306(1968) by Bowdler, Martin, Reinsch, and
      Wilkinson.
      HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 227-240(1971).

      This subroutine finds the eigenvalues of a SYMMETRIC
      TRIDIAGONAL matrix by the QL method.

      On Input

         N is the order of the matrix.  N is an INTEGER variable.

         D contains the diagonal elements of the symmetric tridiagonal
           matrix.  D is a one-dimensional REAL array, dimensioned D(N).

         E contains the subdiagonal elements of the symmetric
           tridiagonal matrix in its last N-1 positions.  E(1) is
           arbitrary.  E is a one-dimensional REAL array, dimensioned
           E(N).

       On Output

         D contains the eigenvalues in ascending order.  If an
           error exit is made, the eigenvalues are correct and
           ordered for indices 1, 2, ..., IERR-1, but may not be
           the smallest eigenvalues.

         E has been destroyed.

         IERR is an INTEGER flag set to
           Zero       for normal return,
           J          if the J-th eigenvalue has not been
                      determined after 30 iterations.

      Calls PYTHAG(A,B) for sqrt(A**2 + B**2).

      Questions and comments should be directed to B. S. Garbow,
      APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
      ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
```

```
***ROUTINES CALLED  PYTHAG
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

TQL2

```
      SUBROUTINE TQL2 (NM, N, D, E, Z, IERR)
***BEGIN PROLOGUE  TQL2
***PURPOSE  Compute the eigenvalues and eigenvectors of symmetric
            tridiagonal matrix.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4A5, D4C2A
***TYPE      SINGLE PRECISION (TQL2-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine is a translation of the ALGOL procedure TQL2,
      NUM. MATH. 11, 293-306(1968) by Bowdler, Martin, Reinsch, and
      Wilkinson.
      HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 227-240(1971).

      This subroutine finds the eigenvalues and eigenvectors
      of a SYMMETRIC TRIDIAGONAL matrix by the QL method.
      The eigenvectors of a FULL SYMMETRIC matrix can also
      be found if  TRED2  has been used to reduce this
      full matrix to tridiagonal form.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameter, Z, as declared in the calling program
           dimension statement.  NM is an INTEGER variable.

         N is the order of the matrix.  N is an INTEGER variable.
           N must be less than or equal to NM.

         D contains the diagonal elements of the symmetric tridiagonal
           matrix.  D is a one-dimensional REAL array, dimensioned D(N).

         E contains the subdiagonal elements of the symmetric
           tridiagonal matrix in its last N-1 positions.  E(1) is
           arbitrary.  E is a one-dimensional REAL array, dimensioned
           E(N).

         Z contains the transformation matrix produced in the
           reduction by  TRED2, if performed.  If the eigenvectors
           of the tridiagonal matrix are desired, Z must contain
           the identity matrix.  Z is a two-dimensional REAL array,
           dimensioned Z(NM,N).

       On Output

         D contains the eigenvalues in ascending order.  If an
           error exit is made, the eigenvalues are correct but
           unordered for indices 1, 2, ..., IERR-1.

         E has been destroyed.

         Z contains orthonormal eigenvectors of the symmetric
           tridiagonal (or full) matrix.  If an error exit is made,
           Z contains the eigenvectors associated with the stored
```

```
          eigenvalues.

        IERR is an INTEGER flag set to
          Zero       for normal return,
          J          if the J-th eigenvalue has not been
                     determined after 30 iterations.

     Calls PYTHAG(A,B) for sqrt(A**2 + B**2).

     Questions and comments should be directed to B. S. Garbow,
     APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
     ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  PYTHAG
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

TQLRAT

```
      SUBROUTINE TQLRAT (N, D, E2, IERR)
***BEGIN PROLOGUE  TQLRAT
***PURPOSE  Compute the eigenvalues of symmetric tridiagonal matrix
            using a rational variant of the QL method.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4A5, D4C2A
***TYPE      SINGLE PRECISION (TQLRAT-S)
***KEYWORDS  EIGENVALUES OF A SYMMETRIC TRIDIAGONAL MATRIX, EISPACK,
             QL METHOD
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine is a translation of the ALGOL procedure TQLRAT.

      This subroutine finds the eigenvalues of a SYMMETRIC
      TRIDIAGONAL matrix by the rational QL method.

      On Input

         N is the order of the matrix.  N is an INTEGER variable.

         D contains the diagonal elements of the symmetric tridiagonal
           matrix.  D is a one-dimensional REAL array, dimensioned D(N).

         E2 contains the squares of the subdiagonal elements of the
           symmetric tridiagonal matrix in its last N-1 positions.
           E2(1) is arbitrary.  E2 is a one-dimensional REAL array,
           dimensioned E2(N).

       On Output

         D contains the eigenvalues in ascending order.  If an
           error exit is made, the eigenvalues are correct and
           ordered for indices 1, 2, ..., IERR-1, but may not be
           the smallest eigenvalues.

         E2 has been destroyed.

         IERR is an INTEGER flag set to
           Zero       for normal return,
           J          if the J-th eigenvalue has not been
                      determined after 30 iterations.

      Calls PYTHAG(A,B) for sqrt(A**2 + B**2).

      Questions and comments should be directed to B. S. Garbow,
      APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
      ------------------------------------------------------------------
```


***REFERENCES B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigensystem Routines - EISPACK Guide, Springer-Verlag, 1976.

C. H. Reinsch, Eigenvalues of a real, symmetric, tridiagonal matrix, Algorithm 464, Communications of the ACM 16, 11 (November 1973), pp. 689.

```
***ROUTINES CALLED  PYTHAG, R1MACH
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

TRBAK1

```
      SUBROUTINE TRBAK1 (NM, N, A, E, M, Z)
***BEGIN PROLOGUE  TRBAK1
***PURPOSE  Form the eigenvectors of real symmetric matrix from
            the eigenvectors of a symmetric tridiagonal matrix formed
            by TRED1.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4C4
***TYPE      SINGLE PRECISION (TRBAK1-S)
***KEYWORDS  EIGENVECTORS OF A REAL SYMMETRIC MATRIX, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine is a translation of the ALGOL procedure TRBAK1,
      NUM. MATH. 11, 181-195(1968) by Martin, Reinsch, and Wilkinson.
      HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 212-226(1971).

      This subroutine forms the eigenvectors of a REAL SYMMETRIC
      matrix by back transforming those of the corresponding
      symmetric tridiagonal matrix determined by  TRED1.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameters, A and Z, as declared in the calling
           program dimension statement.  NM is an INTEGER variable.

         N is the order of the matrix.  N is an INTEGER variable.
           N must be less than or equal to NM.

         A contains information about the orthogonal transformations
           used in the reduction by  TRED1  in its strict lower
           triangle.  A is a two-dimensional REAL array, dimensioned
           A(NM,N).

         E contains the subdiagonal elements of the tridiagonal matrix
           in its last N-1 positions.  E(1) is arbitrary.  These
           elements provide the remaining information about the
           orthogonal transformations.  E is a one-dimensional REAL
           array, dimensioned E(N).

         M is the number of columns of Z to be back transformed.
           M is an INTEGER variable.

         Z contains the eigenvectors to be back transformed in its
           first M columns.  Z is a two-dimensional REAL array,
           dimensioned Z(NM,M).

      On Output

         Z contains the transformed eigenvectors in its first M columns.

      Note that TRBAK1 preserves vector Euclidean norms.

      Questions and comments should be directed to B. S. Garbow,
      APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
      ------------------------------------------------------------------
```

```
***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

TRBAK3

```
      SUBROUTINE TRBAK3 (NM, N, NV, A, M, Z)
***BEGIN PROLOGUE  TRBAK3
***PURPOSE  Form the eigenvectors of a real symmetric matrix from the
            eigenvectors of a symmetric tridiagonal matrix formed
            by TRED3.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4C4
***TYPE      SINGLE PRECISION (TRBAK3-S)
***KEYWORDS  EIGENVECTORS OF A REAL SYMMETRIC MATRIX, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine is a translation of the ALGOL procedure TRBAK3,
      NUM. MATH. 11, 181-195(1968) by Martin, Reinsch, and Wilkinson.
      HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 212-226(1971).

      This subroutine forms the eigenvectors of a REAL SYMMETRIC
      matrix by back transforming those of the corresponding
      symmetric tridiagonal matrix determined by  TRED3.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameter, Z, as declared in the calling program
           dimension statement.  NM is an INTEGER variable.

         N is the order of the matrix.  N is an INTEGER variable.
           N must be less than or equal to NM.

         NV is an INTEGER variable set equal to the dimension of the
           array A as specified in the calling program.  NV must not
           be less than  N*(N+1)/2.

         A contains information about the orthogonal transformations
           used in the reduction by  TRED3  in its first N*(N+1)/2
           positions.  A is a one-dimensional REAL array, dimensioned
           A(NV).

         M is the number of columns of Z to be back transformed.
           M is an INTEGER variable.

         Z contains the eigenvectors to be back transformed in its
           first M columns.  Z is a two-dimensional REAL array,
           dimensioned Z(NM,M).

      On Output

         Z contains the transformed eigenvectors in its first M columns.

      Note that TRBAK3 preserves vector Euclidean norms.

      Questions and comments should be directed to b. s. Garbow,
      APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
      ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
```

```
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

TRED1

```
      SUBROUTINE TRED1 (NM, N, A, D, E, E2)
***BEGIN PROLOGUE  TRED1
***PURPOSE  Reduce a real symmetric matrix to symmetric tridiagonal
            matrix using orthogonal similarity transformations.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4C1B1
***TYPE      SINGLE PRECISION (TRED1-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine is a translation of the ALGOL procedure TRED1,
      NUM. MATH. 11, 181-195(1968) by Martin, Reinsch, and Wilkinson.
      HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 212-226(1971).

      This subroutine reduces a REAL SYMMETRIC matrix
      to a symmetric tridiagonal matrix using
      orthogonal similarity transformations.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameter, A, as declared in the calling program
           dimension statement.  NM is an INTEGER variable.

         N is the order of the matrix A.  N is an INTEGER variable.
           N must be less than or equal to NM.

         A contains the real symmetric input matrix.  Only the lower
           triangle of the matrix need be supplied.  A is a two-
           dimensional REAL array, dimensioned A(NM,N).

      On Output

         A contains information about the orthogonal transformations
           used in the reduction in its strict lower triangle.  The
           full upper triangle of A is unaltered.

         D contains the diagonal elements of the symmetric tridiagonal
           matrix.  D is a one-dimensional REAL array, dimensioned D(N).

         E contains the subdiagonal elements of the symmetric
           tridiagonal matrix in its last N-1 positions.  E(1) is set
           to zero.  E is a one-dimensional REAL array, dimensioned
           E(N).

         E2 contains the squares of the corresponding elements of E.
           E2 may coincide with E if the squares are not needed.
           E2 is a one-dimensional REAL array, dimensioned E2(N).

      Questions and comments should be directed to B. S. Garbow,
      APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
      ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
```

```
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

TRED2

```
      SUBROUTINE TRED2 (NM, N, A, D, E, Z)
***BEGIN PROLOGUE  TRED2
***PURPOSE  Reduce a real symmetric matrix to a symmetric tridiagonal
            matrix using and accumulating orthogonal transformations.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4C1B1
***TYPE      SINGLE PRECISION (TRED2-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine is a translation of the ALGOL procedure TRED2,
      NUM. MATH. 11, 181-195(1968) by Martin, Reinsch, and Wilkinson.
      HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 212-226(1971).

      This subroutine reduces a REAL SYMMETRIC matrix to a
      symmetric tridiagonal matrix using and accumulating
      orthogonal similarity transformations.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameters, A and Z, as declared in the calling
           program dimension statement.  NM is an INTEGER variable.

         N is the order of the matrix A.  N is an INTEGER variable.
           N must be less than or equal to NM.

         A contains the real symmetric input matrix.  Only the lower
           triangle of the matrix need be supplied.  A is a two-
           dimensional REAL array, dimensioned A(NM,N).

      On Output

         D contains the diagonal elements of the symmetric tridiagonal
           matrix.  D is a one-dimensional REAL array, dimensioned D(N).

         E contains the subdiagonal elements of the symmetric
           tridiagonal matrix in its last N-1 positions.  E(1) is set
           to zero.  E is a one-dimensional REAL array, dimensioned
           E(N).

         Z contains the orthogonal transformation matrix produced in
           the reduction.  Z is a two-dimensional REAL array,
           dimensioned Z(NM,N).

         A and Z may coincide.  If distinct, A is unaltered.

      Questions and comments should be directed to B. S. Garbow,
      APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
      ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
```

```
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

TRED3

```
      SUBROUTINE TRED3 (N, NV, A, D, E, E2)
***BEGIN PROLOGUE  TRED3
***PURPOSE  Reduce a real symmetric matrix stored in packed form to
            symmetric tridiagonal matrix using orthogonal
            transformations.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4C1B1
***TYPE      SINGLE PRECISION (TRED3-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine is a translation of the ALGOL procedure TRED3,
      NUM. MATH. 11, 181-195(1968) by Martin, Reinsch, and Wilkinson.
      HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 212-226(1971).

      This subroutine reduces a REAL SYMMETRIC matrix, stored as
      a one-dimensional array, to a symmetric tridiagonal matrix
      using orthogonal similarity transformations.

      On Input

         N is the order of the matrix A.  N is an INTEGER variable.

         NV is an INTEGER variable set equal to the dimension of the
           array A as specified in the calling program.  NV must not
           be less than  N*(N+1)/2.

         A contains the lower triangle, stored row-wise, of the real
           symmetric packed matrix.  A is a one-dimensional REAL
           array, dimensioned A(NV).

      On Output

         A contains information about the orthogonal transformations
           used in the reduction in its first N*(N+1)/2 positions.

         D contains the diagonal elements of the symmetric tridiagonal
           matrix.  D is a one-dimensional REAL array, dimensioned D(N).

         E contains the subdiagonal elements of the symmetric
           tridiagonal matrix in its last N-1 positions.  E(1) is set
           to zero.  E is a one-dimensional REAL array, dimensioned
           E(N).

         E2 contains the squares of the corresponding elements of E.
           E2 may coincide with E if the squares are not needed.
           E2 is a one-dimensional REAL array, dimensioned E2(N).

      Questions and comments should be directed to B. S. Garbow,
      APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
      ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
```

```
                 1976.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

TRI3

```
      SUBROUTINE TRI3 (M, A, B, C, K, Y1, Y2, Y3, TCOS, D, W1, W2, W3)
***BEGIN PROLOGUE  TRI3
***SUBSIDIARY
***PURPOSE  Subsidiary to GENBUN
***LIBRARY   SLATEC
***TYPE      SINGLE PRECISION (TRI3-S, CMPTR3-C)
***AUTHOR  (UNKNOWN)
***DESCRIPTION

      Subroutine to solve three linear systems whose common coefficient
      matrix is a rational function in the matrix given by

                   TRIDIAGONAL (...,A(I),B(I),C(I),...)

***SEE ALSO  GENBUN
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   801001  DATE WRITTEN
   890206  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900402  Added TYPE section.  (WRB)
   END PROLOGUE
```

TRIDIB

```
      SUBROUTINE TRIDIB (N, EPS1, D, E, E2, M11, M, LB, UB, W, IND,
     +   IERR, RV4, RV5)
***BEGIN PROLOGUE  TRIDIB
***PURPOSE  Compute the eigenvalues of a symmetric tridiagonal matrix
            in a given interval using Sturm sequencing.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4A5, D4C2A
***TYPE      SINGLE PRECISION (TRIDIB-S)
***KEYWORDS  EIGENVALUES OF A REAL SYMMETRIC MATRIX, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine is a translation of the ALGOL procedure BISECT,
      NUM. MATH. 9, 386-393(1967) by Barth, Martin, and Wilkinson.
      HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 249-256(1971).

      This subroutine finds those eigenvalues of a TRIDIAGONAL
      SYMMETRIC matrix between specified boundary indices,
      using bisection.

      On Input

         N is the order of the matrix.  N is an INTEGER variable.

         EPS1 is an absolute error tolerance for the computed eigen-
           values.  If the input EPS1 is non-positive, it is reset for
           each submatrix to a default value, namely, minus the product
           of the relative machine precision and the 1-norm of the
           submatrix.  EPS1 is a REAL variable.

         D contains the diagonal elements of the symmetric tridiagonal
           matrix.  D is a one-dimensional REAL array, dimensioned D(N).

         E contains the subdiagonal elements of the symmetric
           tridiagonal matrix in its last N-1 positions.  E(1) is
           arbitrary.  E is a one-dimensional REAL array, dimensioned
           E(N).

         E2 contains the squares of the corresponding elements of E.
           E2(1) is arbitrary.  E2 is a one-dimensional REAL array,
           dimensioned E2(N).

         M11 specifies the lower boundary index for the set of desired
           eigenvalues.  M11 is an INTEGER variable.

         M specifies the number of eigenvalues desired.  The upper
           boundary index M22 is then obtained as M22=M11+M-1.
           M is an INTEGER variable.

      On Output

         EPS1 is unaltered unless it has been reset to its
           (last) default value.

         D and E are unaltered.
```

```
         Elements of E2, corresponding to elements of E regarded
           as negligible, have been replaced by zero causing the
           matrix to split into a direct sum of submatrices.
           E2(1) is also set to zero.

         LB and UB define an interval containing exactly the desired
           eigenvalues.  LB and UB are REAL variables.

         W contains, in its first M positions, the eigenvalues
           between indices M11 and M22 in ascending order.
           W is a one-dimensional REAL array, dimensioned W(M).

         IND contains in its first M positions the submatrix indices
           associated with the corresponding eigenvalues in W --
           1 for eigenvalues belonging to the first submatrix from
           the top, 2 for those belonging to the second submatrix, etc.
           IND is an one-dimensional INTEGER array, dimensioned IND(M).

         IERR is an INTEGER flag set to
           Zero       for normal return,
           3*N+1      if multiple eigenvalues at index M11 make
                      unique selection of LB impossible,
           3*N+2      if multiple eigenvalues at index M22 make
                      unique selection of UB impossible.

         RV4 and RV5 are one-dimensional REAL arrays used for temporary
           storage of the lower and upper bounds for the eigenvalues in
           the bisection process.  RV4 and RV5 are dimensioned RV4(N)
           and RV5(N).

      Note that subroutine TQL1, IMTQL1, or TQLRAT is generally faster
      than TRIDIB, if more than N/4 eigenvalues are to be found.

      Questions and comments should be directed to B. S. Garbow,
      APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
      ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  R1MACH
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890531  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

TSTURM

```
      SUBROUTINE TSTURM (NM,N, EPS1, D, E, E2, LB, MM, M, W, Z,
     +   IERR, RV1, RV2, RV3, RV4, RV5, RV6)
***BEGIN PROLOGUE  TSTURM
***PURPOSE  Find those eigenvalues of a symmetric tridiagonal matrix
            in a given interval and their associated eigenvectors by
            Sturm sequencing.
***LIBRARY   SLATEC (EISPACK)
***CATEGORY  D4A5, D4C2A
***TYPE      SINGLE PRECISION (TSTURM-S)
***KEYWORDS  EIGENVALUES, EIGENVECTORS, EISPACK
***AUTHOR  Smith, B. T., et al.
***DESCRIPTION

      This subroutine finds those eigenvalues of a TRIDIAGONAL
      SYMMETRIC matrix which lie in a specified interval and their
      associated eigenvectors, using bisection and inverse iteration.

      On Input

         NM must be set to the row dimension of the two-dimensional
           array parameter, Z, as declared in the calling program
           dimension statement.  NM is an INTEGER variable.

         N is the order of the matrix.  N is an INTEGER variable.
           N must be less than or equal to NM.

         EPS1 is an absolute error tolerance for the computed eigen-
           values.  It should be chosen so that the accuracy of these
           eigenvalues is commensurate with relative perturbations of
           the order of the relative machine precision in the matrix
           elements.  If the input EPS1 is non-positive, it is reset
           for each submatrix to a default value, namely, minus the
           product of the relative machine precision and the 1-norm of
           the submatrix.  EPS1 is a REAL variable.

         D contains the diagonal elements of the symmetric tridiagonal
           matrix.  D is a one-dimensional REAL array, dimensioned D(N).

         E contains the subdiagonal elements of the symmetric
           tridiagonal matrix in its last N-1 positions.  E(1) is
           arbitrary.  E is a one-dimensional REAL array, dimensioned
           E(N).

         E2 contains the squares of the corresponding elements of E.
           E2(1) is arbitrary.  E2 is a one-dimensional REAL array,
           dimensioned E2(N).

         LB and UB define the interval to be searched for eigenvalues.
           If LB is not less than UB, no eigenvalues will be found.
           LB and UB are REAL variables.

         MM should be set to an upper bound for the number of
           eigenvalues in the interval.  MM is an INTEGER variable.
           WARNING -  If more than MM eigenvalues are determined to lie
           in the interval, an error return is made with no values or
           vectors found.
```

```
     On Output

        EPS1 is unaltered unless it has been reset to its
          (last) default value.

        D and E are unaltered.

        Elements of E2, corresponding to elements of E regarded as
          negligible, have been replaced by zero causing the matrix to
          split into a direct sum of submatrices.  E2(1) is also set
          to zero.

        M is the number of eigenvalues determined to lie in (LB,UB).
          M is an INTEGER variable.

        W contains the M eigenvalues in ascending order if the matrix
          does not split.  If the matrix splits, the eigenvalues are
          in ascending order for each submatrix.  If a vector error
          exit is made, W contains those values already found.  W is a
          one-dimensional REAL array, dimensioned W(MM).

        Z contains the associated set of orthonormal eigenvectors.
          If an error exit is made, Z contains those vectors already
          found.  Z is a one-dimensional REAL array, dimensioned
          Z(NM,MM).

        IERR is an INTEGER flag set to
          Zero       for normal return,
          3*N+1      if M exceeds MM no eigenvalues or eigenvectors
                     are computed,
          4*N+J      if the eigenvector corresponding to the J-th
                     eigenvalue fails to converge in 5 iterations, then
                     the eigenvalues and eigenvectors in W and Z should
                     be correct for indices 1, 2, ..., J-1.

        RV1, RV2, RV3, RV4, RV5, and RV6 are temporary storage arrays,
          dimensioned RV1(N), RV2(N), RV3(N), RV4(N), RV5(N), and
          RV6(N).

     The ALGOL procedure STURMCNT contained in TRISTURM
     appears in TSTURM in-line.

     Questions and comments should be directed to B. S. Garbow,
     APPLIED MATHEMATICS DIVISION, ARGONNE NATIONAL LABORATORY
     ------------------------------------------------------------------

***REFERENCES  B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow,
                 Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigen-
                 system Routines - EISPACK Guide, Springer-Verlag,
                 1976.
***ROUTINES CALLED  R1MACH
***REVISION HISTORY  (YYMMDD)
   760101  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890531  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

ULSIA

```
      SUBROUTINE ULSIA (A, MDA, M, N, B, MDB, NB, RE, AE, KEY, MODE,
     +   NP, KRANK, KSURE, RNORM, W, LW, IWORK, LIW, INFO)
***BEGIN PROLOGUE  ULSIA
***PURPOSE  Solve an underdetermined linear system of equations by
            performing an LQ factorization of the matrix using
            Householder transformations.  Emphasis is put on detecting
            possible rank deficiency.
***LIBRARY   SLATEC
***CATEGORY  D9
***TYPE      SINGLE PRECISION (ULSIA-S, DULSIA-D)
***KEYWORDS  LINEAR LEAST SQUARES, LQ FACTORIZATION,
             UNDERDETERMINED LINEAR SYSTEM
***AUTHOR  Manteuffel, T. A., (LANL)
***DESCRIPTION

      ULSIA computes the minimal length solution(s) to the problem AX=B
      where A is an M by N matrix with M.LE.N and B is the M by NB
      matrix of right hand sides.  User input bounds on the uncertainty
      in the elements of A are used to detect numerical rank deficiency.
      The algorithm employs a row and column pivot strategy to
      minimize the growth of uncertainty and round-off errors.

      ULSIA requires (MDA+1)*N + (MDB+1)*NB + 6*M dimensioned space

    ******************************************************************
    *                                                                *
    *         WARNING - All input arrays are changed on exit.        *
    *                                                                *
    ******************************************************************

      Input..

      A(,)           Linear coefficient matrix of AX=B, with MDA the
       MDA,M,N       actual first dimension of A in the calling program.
                     M is the row dimension (no. of EQUATIONS of the
                     problem) and N the col dimension (no. of UNKNOWNS).
                     Must have MDA.GE.M and M.LE.N.

      B(,)           Right hand side(s), with MDB the actual first
       MDB,NB        dimension of B in the calling program. NB is the
                     number of M by 1 right hand sides.  Since the
                     solution is returned in B, must have MDB.GE.N. If
                     NB = 0, B is never accessed.

    ******************************************************************
    *                                                                *
    *         Note - Use of RE and AE are what make this             *
    *                code significantly different from               *
    *                other linear least squares solvers.             *
    *                However, the inexperienced user is              *
    *                advised to set RE=0.,AE=0.,KEY=0.               *
    *                                                                *
    ******************************************************************

      RE(),AE(),KEY
      RE()           RE() is a vector of length N such that RE(I) is
```

the maximum relative uncertainty in row I of the matrix A. The values of RE() must be between 0 and 1. A minimum of 10*machine precision will be enforced.

AE() — AE() is a vector of length N such that AE(I) is the maximum absolute uncertainty in row I of the matrix A. The values of AE() must be greater than or equal to 0.

KEY — For ease of use, RE and AE may be input as either vectors or scalars. If a scalar is input, the algorithm will use that value for each column of A. The parameter KEY indicates whether scalars or vectors are being input.

```
     KEY=0     RE scalar  AE scalar
     KEY=1     RE vector  AE scalar
     KEY=2     RE scalar  AE vector
     KEY=3     RE vector  AE vector
```

MODE — The integer MODE indicates how the routine is to react if rank deficiency is detected.

```
If MODE = 0 return immediately, no solution
          1 compute truncated solution
          2 compute minimal length least squares sol
```

The inexperienced user is advised to set MODE=0

NP — The first NP rows of A will not be interchanged with other rows even though the pivot strategy would suggest otherwise.
The inexperienced user is advised to set NP=0.

WORK() — A real work array dimensioned 5*M. However, if RE or AE have been specified as vectors, dimension WORK 4*M. If both RE and AE have been specified as vectors, dimension WORK 3*M.

LW — Actual dimension of WORK

IWORK() — Integer work array dimensioned at least N+M.

LIW — Actual dimension of IWORK.

INFO — Is a flag which provides for the efficient solution of subsequent problems involving the same A but different B.

```
If INFO = 0 original call
   INFO = 1 subsequent calls
```

On subsequent calls, the user must supply A, KRANK, LW, IWORK, LIW, and the first 2*M locations of WORK as output by the original call to ULSIA. MODE must be equal to the value of MODE in the original call. If MODE.LT.2, only the first N locations of WORK are accessed. AE, RE, KEY, and NP are not accessed.

```
     Output..

     A(,)           Contains the lower triangular part of the reduced
                    matrix and the transformation information. It togeth
                    with the first M elements of WORK (see below)
                    completely specify the LQ factorization of A.

     B(,)           Contains the N by NB solution matrix for X.

     KRANK,KSURE    The numerical rank of A,  based upon the relative
                    and absolute bounds on uncertainty, is bounded
                    above by KRANK and below by KSURE. The algorithm
                    returns a solution based on KRANK. KSURE provides
                    an indication of the precision of the rank.

     RNORM()        Contains the Euclidean length of the NB residual
                    vectors  B(I)-AX(I), I=1,NB. If the matrix A is of
                    full rank, then RNORM=0.0.

     WORK()         The first M locations of WORK contain values
                    necessary to reproduce the Householder
                    transformation.

     IWORK()        The first N locations contain the order in
                    which the columns of A were used. The next
                    M locations contain the order in which the
                    rows of A were used.

     INFO           Flag to indicate status of computation on completion
                   -1   Parameter error(s)
                    0 - Rank deficient, no solution
                    1 - Rank deficient, truncated solution
                    2 - Rank deficient, minimal length least squares sol
                    3 - Numerical rank 0, zero solution
                    4 - Rank .LT. NP
                    5 - Full rank

***REFERENCES  T. Manteuffel, An interval analysis approach to rank
                 determination in linear least squares problems,
                 Report SAND80-0655, Sandia Laboratories, June 1980.
***ROUTINES CALLED  R1MACH, U11US, U12US, XERMSG
***REVISION HISTORY  (YYMMDD)
   810801  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   891009  Removed unreferenced variable.  (WRB)
   891009  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900510  Fixed an error message.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

WNNLS

```
      SUBROUTINE WNNLS (W, MW, ME, MA, N, L, PRGOPT, X, RNORM, MODE,
     +   IWORK, WORK)
***BEGIN PROLOGUE  WNNLS
***PURPOSE  Solve a linearly constrained least squares problem with
            equality constraints and nonnegativity constraints on
            selected variables.
***LIBRARY   SLATEC
***CATEGORY  K1A2A
***TYPE      SINGLE PRECISION (WNNLS-S, DWNNLS-D)
***KEYWORDS  CONSTRAINED LEAST SQUARES, CURVE FITTING, DATA FITTING,
             EQUALITY CONSTRAINTS, INEQUALITY CONSTRAINTS,
             NONNEGATIVITY CONSTRAINTS, QUADRATIC PROGRAMMING
***AUTHOR  Hanson, R. J., (SNLA)
           Haskell, K. H., (SNLA)
***DESCRIPTION

      Abstract

      This subprogram solves a linearly constrained least squares
      problem.  Suppose there are given matrices E and A of
      respective dimensions ME by N and MA by N, and vectors F
      and B of respective lengths ME and MA.  This subroutine
      solves the problem

                EX = F, (equations to be exactly satisfied)

                AX = B, (equations to be approximately satisfied,
                         in the least squares sense)

                subject to components L+1,...,N nonnegative

      Any values ME.GE.0, MA.GE.0 and 0.LE. L .LE.N are permitted.

      The problem is reposed as problem WNNLS

                (WT*E)X = (WT*F)
                (   A)    (   B), (least squares)
                subject to components L+1,...,N nonnegative.

      The subprogram chooses the heavy weight (or penalty parameter) WT.

      The parameters for WNNLS are

      INPUT..

      W(*,*),MDW,  The array W(*,*) is double subscripted with first
      ME,MA,N,L    dimensioning parameter equal to MDW.  For this
                   discussion let us call M = ME + MA.  Then MDW
                   must satisfy MDW.GE.M.  The condition MDW.LT.M
                   is an error.

                   The array W(*,*) contains the matrices and vectors

                        (E  F)
                        (A  B)
```

in rows and columns 1,...,M and 1,...,N+1 respectively. Columns 1,...,L correspond to unconstrained variables X(1),...,X(L). The remaining variables are constrained to be nonnegative. The condition L.LT.0 or L.GT.N is an error.

PRGOPT(*) This real-valued array is the option vector. If the user is satisfied with the nominal subprogram features set

PRGOPT(1)=1 (or PRGOPT(1)=1.0)

Otherwise PRGOPT(*) is a linked list consisting of groups of data of the following form

LINK
KEY
DATA SET

The parameters LINK and KEY are each one word. The DATA SET can be comprised of several words. The number of items depends on the value of KEY. The value of LINK points to the first entry of the next group of data within PRGOPT(*). The exception is when there are no more options to change. In that case LINK=1 and the values KEY and DATA SET are not referenced. The general layout of PRGOPT(*) is as follows.

```
...PRGOPT(1)=LINK1 (link to first entry of next group)
.  PRGOPT(2)=KEY1 (key to the option change)
.  PRGOPT(3)=DATA VALUE (data value for this change)
.       .
.       .
.       .
...PRGOPT(LINK1)=LINK2 (link to the first entry of
.                       next group)
.  PRGOPT(LINK1+1)=KEY2 (key to the option change)
.  PRGOPT(LINK1+2)=DATA VALUE
...     .
.       .
.       .
...PRGOPT(LINK)=1 (no more options to change)
```

Values of LINK that are nonpositive are errors. A value of LINK.GT.NLINK=100000 is also an error. This helps prevent using invalid but positive values of LINK that will probably extend beyond the program limits of PRGOPT(*). Unrecognized values of KEY are ignored. The order of the options is arbitrary and any number of options can be changed with the following restriction. To prevent cycling in the processing of the option array a count of the number of options changed is maintained. Whenever this count exceeds NOPT=1000 an error message is printed and the subprogram returns.

OPTIONS..

KEY=6
Scale the nonzero columns of the
entire data matrix
(E)
(A)
to have length one. The DATA SET for
this option is a single value. It must
be nonzero if unit length column scaling is
desired.

KEY=7
Scale columns of the entire data matrix
(E)
(A)
with a user-provided diagonal matrix.
The DATA SET for this option consists
of the N diagonal scaling factors, one for
each matrix column.

KEY=8
Change the rank determination tolerance from
the nominal value of SQRT(SRELPR). This quantity
can be no smaller than SRELPR, The arithmetic-
storage precision. The quantity used
here is internally restricted to be at
least SRELPR. The DATA SET for this option
is the new tolerance.

KEY=9
Change the blow-up parameter from the
nominal value of SQRT(SRELPR). The reciprocal of
this parameter is used in rejecting solution
components as too large when a variable is
first brought into the active set. Too large
means that the proposed component times the
reciprocal of the parameter is not less than
the ratio of the norms of the right-side
vector and the data matrix.
This parameter can be no smaller than SRELPR,
the arithmetic-storage precision.

For example, suppose we want to provide
a diagonal matrix to scale the problem
matrix and change the tolerance used for
determining linear dependence of dropped col
vectors. For these options the dimensions of
PRGOPT(*) must be at least N+6. The FORTRAN
statements defining these options would
be as follows.

PRGOPT(1)=N+3 (link to entry N+3 in PRGOPT(*))
PRGOPT(2)=7 (user-provided scaling key)

CALL SCOPY(N,D,1,PRGOPT(3),1) (copy the N
scaling factors from a user array called D(*)
into PRGOPT(3)-PRGOPT(N+2))

PRGOPT(N+3)=N+6 (link to entry N+6 of PRGOPT(*))

PRGOPT(N+4)=8 (linear dependence tolerance key)
PRGOPT(N+5)=... (new value of the tolerance)

PRGOPT(N+6)=1 (no more options to change)

IWORK(1), IWORK(2) — The amounts of working storage actually allocated for the working arrays WORK(*) and IWORK(*), respectively. These quantities are compared with the actual amounts of storage needed for WNNLS( ). Insufficient storage allocated for either WORK(*) or IWORK(*) is considered an error. This feature was included in WNNLS( ) because miscalculating the storage formulas for WORK(*) and IWORK(*) might very well lead to subtle and hard-to-find execution errors.

The length of WORK(*) must be at least

LW = ME+MA+5*N
This test will not be made if IWORK(1).LE.0.

The length of IWORK(*) must be at least

LIW = ME+MA+N
This test will not be made if IWORK(2).LE.0.

OUTPUT..

X(*) — An array dimensioned at least N, which will contain the N components of the solution vector on output.

RNORM — The residual norm of the solution. The value of RNORM contains the residual vector length of the equality constraints and least squares equations.

MODE — The value of MODE indicates the success or failure of the subprogram.

MODE = 0 Subprogram completed successfully.

= 1 Max. number of iterations (equal to 3*(N-L)) exceeded. Nearly all problems should complete in fewer than this number of iterations. An approximate solution and its corresponding residual vector length are in X(*) and RNORM.

= 2 Usage error occurred. The offending condition is noted with the error processing subprogram, XERMSG( ).

User-designated Working arrays..

WORK(*) — A real-valued working array of length at least M + 5*N.

IWORK(*) — An integer-valued working array of length at least

```
                  M+N.

***REFERENCES  K. H. Haskell and R. J. Hanson, An algorithm for
                 linear least squares problems with equality and
                 nonnegativity constraints, Report SAND77-0552, Sandia
                 Laboratories, June 1978.
               K. H. Haskell and R. J. Hanson, Selected algorithms for
                 the linearly constrained least squares problem - a
                 users guide, Report SAND78-1290, Sandia Laboratories,
                 August 1979.
               K. H. Haskell and R. J. Hanson, An algorithm for
                 linear least squares problems with equality and
                 nonnegativity constraints, Mathematical Programming
                 21 (1981), pp. 98-118.
               R. J. Hanson and K. H. Haskell, Two algorithms for the
                 linearly constrained least squares problem, ACM
                 Transactions on Mathematical Software, September 1982.
               C. L. Lawson and R. J. Hanson, Solving Least Squares
                 Problems, Prentice-Hall, Inc., 1974.
***ROUTINES CALLED  WNLSM, XERMSG
***REVISION HISTORY  (YYMMDD)
   790701  DATE WRITTEN
   890206  REVISION DATE from Version 3.2
   890618  Completely restructured and revised.  (WRB & RWC)
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900510  Convert XERRWV calls to XERMSG calls.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

XADD

```
      SUBROUTINE XADD (X, IX, Y, IY, Z, IZ, IERROR)
***BEGIN PROLOGUE  XADD
***PURPOSE  To provide single-precision floating-point arithmetic
            with an extended exponent range.
***LIBRARY   SLATEC
***CATEGORY  A3D
***TYPE      SINGLE PRECISION (XADD-S, DXADD-D)
***KEYWORDS  EXTENDED-RANGE SINGLE-PRECISION ARITHMETIC
***AUTHOR  Lozier, Daniel W., (National Bureau of Standards)
           Smith, John M., (NBS and George Mason University)
***DESCRIPTION
      REAL X, Y, Z
      INTEGER IX, IY, IZ

                  FORMS THE EXTENDED-RANGE SUM  (Z,IZ) =
                  (X,IX) + (Y,IY).  (Z,IZ) IS ADJUSTED
                  BEFORE RETURNING. THE INPUT OPERANDS
                  NEED NOT BE IN ADJUSTED FORM, BUT THEIR
                  PRINCIPAL PARTS MUST SATISFY
                  RADIX**(-2L).LE.ABS(X).LE.RADIX**(2L),
                  RADIX**(-2L).LE.ABS(Y).LE.RADIX**(2L).

***SEE ALSO  XSET
***REFERENCES  (NONE)
***ROUTINES CALLED  XADJ
***COMMON BLOCKS    XBLK2
***REVISION HISTORY  (YYMMDD)
   820712  DATE WRITTEN
   881020  Revised to meet SLATEC CML recommendations.  (DWL and JMS)
   901019  Revisions to prologue.  (DWL and WRB)
   901106  Changed all specific intrinsics to generic.  (WRB)
           Corrected order of sections in prologue and added TYPE
           section.  (WRB)
   920127  Revised PURPOSE section of prologue.  (DWL)
   END PROLOGUE
```

XADJ

```
      SUBROUTINE XADJ (X, IX, IERROR)
***BEGIN PROLOGUE  XADJ
***PURPOSE  To provide single-precision floating-point arithmetic
            with an extended exponent range.
***LIBRARY   SLATEC
***CATEGORY  A3D
***TYPE      SINGLE PRECISION (XADJ-S, DXADJ-D)
***KEYWORDS  EXTENDED-RANGE SINGLE-PRECISION ARITHMETIC
***AUTHOR  Lozier, Daniel W., (National Bureau of Standards)
           Smith, John M., (NBS and George Mason University)
***DESCRIPTION
      REAL X
      INTEGER IX

                    TRANSFORMS (X,IX) SO THAT
                    RADIX**(-L) .LE. ABS(X) .LT. RADIX**L.
                    ON MOST COMPUTERS THIS TRANSFORMATION DOES
                    NOT CHANGE THE MANTISSA OF X PROVIDED RADIX IS
                    THE NUMBER BASE OF SINGLE-PRECISION ARITHMETIC.

***SEE ALSO  XSET
***REFERENCES  (NONE)
***ROUTINES CALLED  XERMSG
***COMMON BLOCKS    XBLK2
***REVISION HISTORY  (YYMMDD)
   820712  DATE WRITTEN
   881020  Revised to meet SLATEC CML recommendations.  (DWL and JMS)
   901019  Revisions to prologue.  (DWL and WRB)
   901106  Changed all specific intrinsics to generic.  (WRB)
           Corrected order of sections in prologue and added TYPE
           section.  (WRB)
           CALLs to XERROR changed to CALLs to XERMSG.  (WRB)
   920127  Revised PURPOSE section of prologue.  (DWL)
   END PROLOGUE
```

XC210

```
      SUBROUTINE XC210 (K, Z, J, IERROR)
***BEGIN PROLOGUE  XC210
***PURPOSE  To provide single-precision floating-point arithmetic
            with an extended exponent range.
***LIBRARY   SLATEC
***CATEGORY  A3D
***TYPE      SINGLE PRECISION (XC210-S, DXC210-D)
***KEYWORDS  EXTENDED-RANGE SINGLE-PRECISION ARITHMETIC
***AUTHOR  Lozier, Daniel W., (National Bureau of Standards)
           Smith, John M., (NBS and George Mason University)
***DESCRIPTION
      INTEGER K, J
      REAL Z

                    GIVEN K THIS SUBROUTINE COMPUTES J AND Z
                    SUCH THAT  RADIX**K = Z*10**J, WHERE Z IS IN
                    THE RANGE 1/10 .LE. Z .LT. 1.
                    THE VALUE OF Z WILL BE ACCURATE TO FULL
                    SINGLE-PRECISION PROVIDED THE NUMBER
                    OF DECIMAL PLACES IN THE LARGEST
                    INTEGER PLUS THE NUMBER OF DECIMAL
                    PLACES CARRIED IN SINGLE-PRECISION DOES NOT
                    EXCEED 60. XC210 IS CALLED BY SUBROUTINE
                    XCON WHEN NECESSARY. THE USER SHOULD
                    NEVER NEED TO CALL XC210 DIRECTLY.

***SEE ALSO  XSET
***REFERENCES  (NONE)
***ROUTINES CALLED  XERMSG
***COMMON BLOCKS    XBLK3
***REVISION HISTORY  (YYMMDD)
   820712  DATE WRITTEN
   890126  Revised to meet SLATEC CML recommendations.  (DWL and JMS)
   901019  Revisions to prologue.  (DWL and WRB)
   901106  Changed all specific intrinsics to generic.  (WRB)
           Corrected order of sections in prologue and added TYPE
           section.  (WRB)
           CALLs to XERROR changed to CALLs to XERMSG.  (WRB)
   920127  Revised PURPOSE section of prologue.  (DWL)
   END PROLOGUE
```

XCON

```
      SUBROUTINE XCON (X, IX, IERROR)
***BEGIN PROLOGUE  XCON
***PURPOSE  To provide single-precision floating-point arithmetic
            with an extended exponent range.
***LIBRARY   SLATEC
***CATEGORY  A3D
***TYPE      SINGLE PRECISION (XCON-S, DXCON-D)
***KEYWORDS  EXTENDED-RANGE SINGLE-PRECISION ARITHMETIC
***AUTHOR  Lozier, Daniel W., (National Bureau of Standards)
           Smith, John M., (NBS and George Mason University)
***DESCRIPTION
      REAL X
      INTEGER IX

                    CONVERTS (X,IX) = X*RADIX**IX
                    TO DECIMAL FORM IN PREPARATION FOR
                    PRINTING, SO THAT (X,IX) = X*10**IX
                    WHERE 1/10 .LE. ABS(X) .LT. 1
                    IS RETURNED, EXCEPT THAT IF
                    (ABS(X),IX) IS BETWEEN RADIX**(-2L)
                    AND RADIX**(2L) THEN THE REDUCED
                    FORM WITH IX = 0 IS RETURNED.

***SEE ALSO  XSET
***REFERENCES  (NONE)
***ROUTINES CALLED  XADJ, XC210, XRED
***COMMON BLOCKS    XBLK2
***REVISION HISTORY  (YYMMDD)
   820712  DATE WRITTEN
   881020  Revised to meet SLATEC CML recommendations.  (DWL and JMS)
   901019  Revisions to prologue.  (DWL and WRB)
   901106  Changed all specific intrinsics to generic.  (WRB)
           Corrected order of sections in prologue and added TYPE
           section.  (WRB)
   920127  Revised PURPOSE section of prologue.  (DWL)
   END PROLOGUE
```

XERCLR

```
      SUBROUTINE XERCLR
***BEGIN PROLOGUE  XERCLR
***PURPOSE  Reset current error number to zero.
***LIBRARY   SLATEC (XERROR)
***CATEGORY  R3C
***TYPE      ALL (XERCLR-A)
***KEYWORDS  ERROR, XERROR
***AUTHOR  Jones, R. E., (SNLA)
***DESCRIPTION

      Abstract
         This routine simply resets the current error number to zero.
         This may be necessary in order to determine that a certain
         error has occurred again since the last time NUMXER was
         referenced.

***REFERENCES  R. E. Jones and D. K. Kahaner, XERROR, the SLATEC
                 Error-handling Package, SAND82-0800, Sandia
                 Laboratories, 1982.
***ROUTINES CALLED  J4SAVE
***REVISION HISTORY  (YYMMDD)
   790801  DATE WRITTEN
   861211  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

XERDMP

```
      SUBROUTINE XERDMP
***BEGIN PROLOGUE  XERDMP
***PURPOSE  Print the error tables and then clear them.
***LIBRARY   SLATEC (XERROR)
***CATEGORY  R3C
***TYPE      ALL (XERDMP-A)
***KEYWORDS  ERROR, XERROR
***AUTHOR  Jones, R. E., (SNLA)
***DESCRIPTION

      Abstract
         XERDMP prints the error tables, then clears them.

***REFERENCES  R. E. Jones and D. K. Kahaner, XERROR, the SLATEC
                 Error-handling Package, SAND82-0800, Sandia
                 Laboratories, 1982.
***ROUTINES CALLED  XERSVE
***REVISION HISTORY  (YYMMDD)
   790801  DATE WRITTEN
   861211  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900510  Changed call of XERSAV to XERSVE.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

XERMAX

```
      SUBROUTINE XERMAX (MAX)
***BEGIN PROLOGUE  XERMAX
***PURPOSE  Set maximum number of times any error message is to be
            printed.
***LIBRARY   SLATEC (XERROR)
***CATEGORY  R3C
***TYPE      ALL (XERMAX-A)
***KEYWORDS  ERROR, XERROR
***AUTHOR  Jones, R. E., (SNLA)
***DESCRIPTION

      Abstract
         XERMAX sets the maximum number of times any message
         is to be printed.  That is, non-fatal messages are
         not to be printed after they have occurred MAX times.
         Such non-fatal messages may be printed less than
         MAX times even if they occur MAX times, if error
         suppression mode (KONTRL=0) is ever in effect.

      Description of Parameter
       --Input--
         MAX - the maximum number of times any one message
               is to be printed.

***REFERENCES  R. E. Jones and D. K. Kahaner, XERROR, the SLATEC
                 Error-handling Package, SAND82-0800, Sandia
                 Laboratories, 1982.
***ROUTINES CALLED  J4SAVE
***REVISION HISTORY  (YYMMDD)
   790801  DATE WRITTEN
   861211  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

XERMSG

```
      SUBROUTINE XERMSG (LIBRAR, SUBROU, MESSG, NERR, LEVEL)
***BEGIN PROLOGUE  XERMSG
***PURPOSE  Process error messages for SLATEC and other libraries.
***LIBRARY   SLATEC (XERROR)
***CATEGORY  R3C
***TYPE      ALL (XERMSG-A)
***KEYWORDS  ERROR MESSAGE, XERROR
***AUTHOR  Fong, Kirby, (NMFECC at LLNL)
***DESCRIPTION
```

XERMSG processes a diagnostic message in a manner determined by the value of LEVEL and the current value of the library error control flag, KONTRL. See subroutine XSETF for details.

LIBRAR A character constant (or character variable) with the name of the library. This will be 'SLATEC' for the SLATEC Common Math Library. The error handling package is general enough to be used by many libraries simultaneously, so it is desirable for the routine that detects and reports an error to identify the library name as well as the routine name.

SUBROU A character constant (or character variable) with the name of the routine that detected the error. Usually it is the name of the routine that is calling XERMSG. There are some instances where a user callable library routine calls lower level subsidiary routines where the error is detected. In such cases it may be more informative to supply the name of the routine the user called rather than the name of the subsidiary routine that detected the error.

MESSG A character constant (or character variable) with the text of the error or warning message. In the example below, the message is a character constant that contains a generic message.

```
      CALL XERMSG ('SLATEC', 'MMPY',
     *'THE ORDER OF THE MATRIX EXCEEDS THE ROW DIMENSION',
     *3, 1)
```

It is possible (and is sometimes desirable) to generate a specific message--e.g., one that contains actual numeric values. Specific numeric values can be converted into character strings using formatted WRITE statements into character variables. This is called standard Fortran internal file I/O and is exemplified in the first three lines of the following example. You can also catenate substrings of characters to construct the error message. Here is an example showing the use of both writing to an internal file and catenating character strings.

```
      CHARACTER*5 CHARN, CHARL
      WRITE (CHARN,10) N
      WRITE (CHARL,10) LDA
   10 FORMAT(I5)
```

```
      CALL XERMSG ('SLATEC', 'MMPY', 'THE ORDER'//CHARN//
     *   ' OF THE MATRIX EXCEEDS ITS ROW DIMENSION OF'//
     *   CHARL, 3, 1)
```

There are two subtleties worth mentioning. One is that the // for character catenation is used to construct the error message so that no single character constant is continued to the next line. This avoids confusion as to whether there are trailing blanks at the end of the line. The second is that by catenating the parts of the message as an actual argument rather than encoding the entire message into one large character variable, we avoid having to know how long the message will be in order to declare an adequate length for that large character variable. XERMSG calls XERPRN to print the message using multiple lines if necessary. If the message is very long, XERPRN will break it into pieces of 72 characters (as requested by XERMSG) for printing on multiple lines. Also, XERMSG asks XERPRN to prefix each line with ' * ' so that the total line length could be 76 characters. Note also that XERPRN scans the error message backwards to ignore trailing blanks. Another feature is that the substring '$$' is treated as a new line sentinel by XERPRN. If you want to construct a multiline message without having to count out multiples of 72 characters, just use '$$' as a separator. '$$' obviously must occur within 72 characters of the start of each line to have its intended effect since XERPRN is asked to wrap around at 72 characters in addition to looking for '$$'.

NERR An integer value that is chosen by the library routine's author. It must be in the range -99 to 999 (three printable digits). Each distinct error should have its own error number. These error numbers should be described in the machine readable documentation for the routine. The error numbers need be unique only within each routine, so it is reasonable for each routine to start enumerating errors from 1 and proceeding to the next integer.

LEVEL An integer value in the range 0 to 2 that indicates the level (severity) of the error. Their meanings are

-1 A warning message. This is used if it is not clear that there really is an error, but the user's attention may be needed. An attempt is made to only print this message once.

0 A warning message. This is used if it is not clear that there really is an error, but the user's attention may be needed.

1 A recoverable error. This is used even if the error is so serious that the routine cannot return any useful answer. If the user has told the error package to return after recoverable errors, then XERMSG will return to the Library routine which can then return to the user's routine. The user may also permit the error package to terminate the program upon encountering a recoverable error.

```
            2  A fatal error.  XERMSG will not return to its caller
               after it receives a fatal error.  This level should
               hardly ever be used; it is much better to allow the
               user a chance to recover.  An example of one of the few
               cases in which it is permissible to declare a level 2
               error is a reverse communication Library routine that
               is likely to be called repeatedly until it integrates
               across some interval.  If there is a serious error in
               the input such that another step cannot be taken and
               the Library routine is called again without the input
               error having been corrected by the caller, the Library
               routine will probably be called forever with improper
               input.  In this case, it is reasonable to declare the
               error to be fatal.

    Each of the arguments to XERMSG is input; none will be modified by
    XERMSG.  A routine may make multiple calls to XERMSG with warning
    level messages; however, after a call to XERMSG with a recoverable
    error, the routine should return to the user.  Do not try to call
    XERMSG with a second recoverable error after the first recoverable
    error because the error package saves the error number.  The user
    can retrieve this error number by calling another entry point in
    the error handling package and then clear the error number when
    recovering from the error.  Calling XERMSG in succession causes the
    old error number to be overwritten by the latest error number.
    This is considered harmless for error numbers associated with
    warning messages but must not be done for error numbers of serious
    errors.  After a call to XERMSG with a recoverable error, the user
    must be given a chance to call NUMXER or XERCLR to retrieve or
    clear the error number.
***REFERENCES  R. E. Jones and D. K. Kahaner, XERROR, the SLATEC
                 Error-handling Package, SAND82-0800, Sandia
                 Laboratories, 1982.
***ROUTINES CALLED  FDUMP, J4SAVE, XERCNT, XERHLT, XERPRN, XERSVE
***REVISION HISTORY  (YYMMDD)
   880101  DATE WRITTEN
   880621  REVISED AS DIRECTED AT SLATEC CML MEETING OF FEBRUARY 1988.
           THERE ARE TWO BASIC CHANGES.
           1.  A NEW ROUTINE, XERPRN, IS USED INSTEAD OF XERPRT TO
               PRINT MESSAGES.  THIS ROUTINE WILL BREAK LONG MESSAGES
               INTO PIECES FOR PRINTING ON MULTIPLE LINES.  '$$' IS
               ACCEPTED AS A NEW LINE SENTINEL.  A PREFIX CAN BE
               ADDED TO EACH LINE TO BE PRINTED.  XERMSG USES EITHER
               ' ***' OR ' *  ' AND LONG MESSAGES ARE BROKEN EVERY
               72 CHARACTERS (AT MOST) SO THAT THE MAXIMUM LINE
               LENGTH OUTPUT CAN NOW BE AS GREAT AS 76.
           2.  THE TEXT OF ALL MESSAGES IS NOW IN UPPER CASE SINCE THE
               FORTRAN STANDARD DOCUMENT DOES NOT ADMIT THE EXISTENCE
               OF LOWER CASE.
   880708  REVISED AFTER THE SLATEC CML MEETING OF JUNE 29 AND 30.
           THE PRINCIPAL CHANGES ARE
           1.  CLARIFY COMMENTS IN THE PROLOGUES
           2.  RENAME XRPRNT TO XERPRN
           3.  REWORK HANDLING OF '$$' IN XERPRN TO HANDLE BLANK LINES
               SIMILAR TO THE WAY FORMAT STATEMENTS HANDLE THE /
               CHARACTER FOR NEW RECORDS.
   890706  REVISED WITH THE HELP OF FRED FRITSCH AND REG CLEMENS TO
           CLEAN UP THE CODING.
   890721  REVISED TO USE NEW FEATURE IN XERPRN TO COUNT CHARACTERS IN
```

```
        PREFIX.
891013  REVISED TO CORRECT COMMENTS.
891214  Prologue converted to Version 4.0 format.  (WRB)
900510  Changed test on NERR to be -9999999 < NERR < 99999999, but
        NERR .ne. 0, and on LEVEL to be -2 < LEVEL < 3.  Added
        LEVEL=-1 logic, changed calls to XERSAV to XERSVE, and
        XERCTL to XERCNT.  (RWC)
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

XGETF

```
      SUBROUTINE XGETF (KONTRL)
***BEGIN PROLOGUE  XGETF
***PURPOSE  Return the current value of the error control flag.
***LIBRARY   SLATEC (XERROR)
***CATEGORY  R3C
***TYPE      ALL (XGETF-A)
***KEYWORDS  ERROR, XERROR
***AUTHOR  Jones, R. E., (SNLA)
***DESCRIPTION

   Abstract
        XGETF returns the current value of the error control flag
        in KONTRL.  See subroutine XSETF for flag value meanings.
        (KONTRL is an output parameter only.)

***REFERENCES  R. E. Jones and D. K. Kahaner, XERROR, the SLATEC
                 Error-handling Package, SAND82-0800, Sandia
                 Laboratories, 1982.
***ROUTINES CALLED  J4SAVE
***REVISION HISTORY  (YYMMDD)
   790801  DATE WRITTEN
   861211  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

XGETUA

```
      SUBROUTINE XGETUA (IUNITA, N)
***BEGIN PROLOGUE  XGETUA
***PURPOSE  Return unit number(s) to which error messages are being
            sent.
***LIBRARY   SLATEC (XERROR)
***CATEGORY  R3C
***TYPE      ALL (XGETUA-A)
***KEYWORDS  ERROR, XERROR
***AUTHOR  Jones, R. E., (SNLA)
***DESCRIPTION

      Abstract
         XGETUA may be called to determine the unit number or numbers
         to which error messages are being sent.
         These unit numbers may have been set by a call to XSETUN,
         or a call to XSETUA, or may be a default value.

      Description of Parameters
       --Output--
         IUNIT - an array of one to five unit numbers, depending
                 on the value of N.  A value of zero refers to the
                 default unit, as defined by the I1MACH machine
                 constant routine.  Only IUNIT(1),...,IUNIT(N) are
                 defined by XGETUA.  The values of IUNIT(N+1),...,
                 IUNIT(5) are not defined (for N .LT. 5) or altered
                 in any way by XGETUA.
         N     - the number of units to which copies of the
                 error messages are being sent.  N will be in the
                 range from 1 to 5.

***REFERENCES  R. E. Jones and D. K. Kahaner, XERROR, the SLATEC
                 Error-handling Package, SAND82-0800, Sandia
                 Laboratories, 1982.
***ROUTINES CALLED  J4SAVE
***REVISION HISTORY  (YYMMDD)
   790801  DATE WRITTEN
   861211  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

XGETUN

```
      SUBROUTINE XGETUN (IUNIT)
***BEGIN PROLOGUE  XGETUN
***PURPOSE  Return the (first) output file to which error messages
            are being sent.
***LIBRARY   SLATEC (XERROR)
***CATEGORY  R3C
***TYPE      ALL (XGETUN-A)
***KEYWORDS  ERROR, XERROR
***AUTHOR  Jones, R. E., (SNLA)
***DESCRIPTION

      Abstract
         XGETUN gets the (first) output file to which error messages
         are being sent.  To find out if more than one file is being
         used, one must use the XGETUA routine.

      Description of Parameter
       --Output--
         IUNIT - the logical unit number of the  (first) unit to
                 which error messages are being sent.
                 A value of zero means that the default file, as
                 defined by the I1MACH routine, is being used.

***REFERENCES  R. E. Jones and D. K. Kahaner, XERROR, the SLATEC
                 Error-handling Package, SAND82-0800, Sandia
                 Laboratories, 1982.
***ROUTINES CALLED  J4SAVE
***REVISION HISTORY  (YYMMDD)
   790801  DATE WRITTEN
   861211  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

XLEGF

```
      SUBROUTINE XLEGF (DNU1, NUDIFF, MU1, MU2, THETA, ID, PQA, IPQA,
     1   IERROR)
***BEGIN PROLOGUE  XLEGF
***PURPOSE  Compute normalized Legendre polynomials and associated
            Legendre functions.
***LIBRARY   SLATEC
***CATEGORY  C3A2, C9
***TYPE      SINGLE PRECISION (XLEGF-S, DXLEGF-D)
***KEYWORDS  LEGENDRE FUNCTIONS
***AUTHOR  Smith, John M., (NBS and George Mason University)
***DESCRIPTION

   XLEGF: Extended-range Single-precision Legendre Functions

   A feature of the XLEGF subroutine for Legendre functions is
 the use of extended-range arithmetic, a software extension of
 ordinary floating-point arithmetic that greatly increases the
 exponent range of the representable numbers. This avoids the
 need for scaling the solutions to lie within the exponent range
 of the most restrictive manufacturer's hardware. The increased
 exponent range is achieved by allocating an integer storage
 location together with each floating-point storage location.

   The interpretation of the pair (X,I) where X is floating-point
 and I is integer is X*(IR**I) where IR is the internal radix of
 the computer arithmetic.

   This subroutine computes one of the following vectors:

 1. Legendre function of the first kind of negative order, either
    a. P(-MU1,NU,X), P(-MU1-1,NU,X), ..., P(-MU2,NU,X) or
    b. P(-MU,NU1,X), P(-MU,NU1+1,X), ..., P(-MU,NU2,X)
 2. Legendre function of the second kind, either
    a. Q(MU1,NU,X), Q(MU1+1,NU,X), ..., Q(MU2,NU,X) or
    b. Q(MU,NU1,X), Q(MU,NU1+1,X), ..., Q(MU,NU2,X)
 3. Legendre function of the first kind of positive order, either
    a. P(MU1,NU,X), P(MU1+1,NU,X), ..., P(MU2,NU,X) or
    b. P(MU,NU1,X), P(MU,NU1+1,X), ..., P(MU,NU2,X)
 4. Normalized Legendre polynomials, either
    a. PN(MU1,NU,X), PN(MU1+1,NU,X), ..., PN(MU2,NU,X) or
    b. PN(MU,NU1,X), PN(MU,NU1+1,X), ..., PN(MU,NU2,X)

 where X = COS(THETA).

   The input values to XLEGF are DNU1, NUDIFF, MU1, MU2, THETA,
 and ID. These must satisfy

    DNU1 is REAL and greater than or equal to -0.5;
    NUDIFF is INTEGER and non-negative;
    MU1 is INTEGER and non-negative;
    MU2 is INTEGER and greater than or equal to MU1;
    THETA is REAL and in the half-open interval (0,PI/2];
    ID is INTEGER and equal to 1, 2, 3 or 4;

 and  additionally either NUDIFF = 0 or MU2 = MU1.
```

If ID=1 and NUDIFF=0, a vector of type 1a above is computed with NU=DNU1.

If ID=1 and MU1=MU2, a vector of type 1b above is computed with NU1=DNU1, NU2=DNU1+NUDIFF and MU=MU1.

If ID=2 and NUDIFF=0, a vector of type 2a above is computed with NU=DNU1.

If ID=2 and MU1=MU2, a vector of type 2b above is computed with NU1=DNU1, NU2=DNU1+NUDIFF and MU=MU1.

If ID=3 and NUDIFF=0, a vector of type 3a above is computed with NU=DNU1.

If ID=3 and MU1=MU2, a vector of type 3b above is computed with NU1=DNU1, NU2=DNU1+NUDIFF and MU=MU1.

If ID=4 and NUDIFF=0, a vector of type 4a above is computed with NU=DNU1.

If ID=4 and MU1=MU2, a vector of type 4b above is computed with NU1=DNU1, NU2=DNU1+NUDIFF and MU=MU1.

In each case the vector of computed Legendre function values is returned in the extended-range vector (PQA(I),IPQA(I)). The length of this vector is either MU2-MU1+1 or NUDIFF+1.

Where possible, XLEGF returns IPQA(I) as zero. In this case the value of the Legendre function is contained entirely in PQA(I), so it can be used in subsequent computations without further consideration of extended-range arithmetic. If IPQA(I) is nonzero, then the value of the Legendre function is not representable in floating-point because of underflow or overflow. The program that calls XLEGF must test IPQA(I) to ensure correct usage.

IERROR is an error indicator. If no errors are detected, IERROR=0 when control returns to the calling routine. If an error is detected, IERROR is returned as nonzero. The calling routine must check the value of IERROR.

If IERROR=110 or 111, invalid input was provided to XLEGF.
If IERROR=101,102,103, or 104, invalid input was provided to XSET.
If IERROR=105 or 106, an internal consistency error occurred in XSET (probably due to a software malfunction in the library routine I1MACH).
If IERROR=107, an overflow or underflow of an extended-range number was detected in XADJ.
If IERROR=108, an overflow or underflow of an extended-range number was detected in XC210.

***SEE ALSO XSET
***REFERENCES Olver and Smith, Associated Legendre Functions on the Cut, J Comp Phys, v 51, n 3, Sept 1983, pp 502--518.
Smith, Olver and Lozier, Extended-Range Arithmetic and Normalized Legendre Polynomials, ACM Trans on Math Softw, v 7, n 1, March 1981, pp 93--105.
***ROUTINES CALLED XERMSG, XPMU, XPMUP, XPNRM, XPQNU, XQMU, XQNU, XRED, XSET
***REVISION HISTORY (YYMMDD)

```
820728  DATE WRITTEN
890126  Revised to meet SLATEC CML recommendations.  (DWL and JMS)
901019  Revisions to prologue.  (DWL and WRB)
901106  Changed all specific intrinsics to generic.  (WRB)
        Corrected order of sections in prologue and added TYPE
        section.  (WRB)
        CALLs to XERROR changed to CALLs to XERMSG.  (WRB)
920127  Revised PURPOSE section of prologue.  (DWL)
END PROLOGUE
```

XNRMP

```
      SUBROUTINE XNRMP (NU, MU1, MU2, DARG, MODE, DPN, IPN, ISIG,
     1   IERROR)
***BEGIN PROLOGUE  XNRMP
***PURPOSE  Compute normalized Legendre polynomials.
***LIBRARY   SLATEC
***CATEGORY  C3A2, C9
***TYPE      SINGLE PRECISION (XNRMP-S, DXNRMP-D)
***KEYWORDS  LEGENDRE FUNCTIONS
***AUTHOR  Lozier, Daniel W., (National Bureau of Standards)
           Smith, John M., (NBS and George Mason University)
***DESCRIPTION

        SUBROUTINE TO CALCULATE NORMALIZED LEGENDRE POLYNOMIALS
        (DXNRMP is double-precision version)
        XNRMP calculates normalized Legendre polynomials of varying
        order and fixed argument and degree. The order MU and degree
        NU are non-negative integers and the argument is real. Because
        the algorithm requires the use of numbers outside the normal
        machine range, this subroutine employs a special arithmetic
        called extended-range arithmetic. See J.M. Smith, F.W.J. Olver,
        and D.W. Lozier, Extended-Range Arithmetic and Normalized
        Legendre Polynomials, ACM Transactions on Mathematical Soft-
        ware, 93-105, March 1981, for a complete description of the
        algorithm and special arithmetic. Also see program comments
        in XSET.

        The normalized Legendre polynomials are multiples of the
        associated Legendre polynomials of the first kind where the
        normalizing coefficients are chosen so as to make the integral
        from -1 to 1 of the square of each function equal to 1. See
        E. Jahnke, F. Emde and F. Losch, Tables of Higher Functions,
        McGraw-Hill, New York, 1960, p. 121.

        The input values to XNRMP are NU, MU1, MU2, SARG, and MODE.
        These must satisfy
          1. NU .GE. 0 specifies the degree of the normalized Legendre
             polynomial that is wanted.
          2. MU1 .GE. 0 specifies the lowest-order normalized Legendre
             polynomial that is wanted.
          3. MU2 .GE. MU1 specifies the highest-order normalized Leg-
             endre polynomial that is wanted.
         4a. MODE = 1 and -1.0 .LE. SARG .LE. 1.0 specifies that
             Normalized Legendre(NU, MU, SARG) is wanted for MU = MU1,
             MU1 + 1, ..., MU2.
         4b. MODE = 2 and -3.14159... .LT. SARG .LT. 3.14159... spec-
             ifies that Normalized Legendre(NU, MU, COS(SARG)) is want-
             ed for MU = MU1, MU1 + 1, ..., MU2.

        The output of XNRMP consists of the two vectors SPN and IPN
        and the error estimate ISIG. The computed values are stored as
        extended-range numbers such that
             (SPN(1),IPN(1))=NORMALIZED LEGENDRE(NU,MU1,X)
             (SPN(2),IPN(2))=NORMALIZED LEGENDRE(NU,MU1+1,X)
                .
                .
             (SPN(K),IPN(K))=NORMALIZED LEGENDRE(NU,MU2,X)
```

where K = MU2 - MU1 + 1 and X = SARG or COS(SARG) according to whether MODE = 1 or 2. Finally, ISIG is an estimate of the number of decimal digits lost through rounding errors in the computation. For example if SARG is accurate to 12 significant decimals, then the computed function values are accurate to 12 - ISIG significant decimals (except in neighborhoods of zeros).

The interpretation of (SPN(I),IPN(I)) is SPN(I)*(IR**IPN(I)) where IR is the internal radix of the computer arithmetic. When IPN(I) = 0 the value of the normalized Legendre polynomial is contained entirely in SPN(I) and subsequent single-precision computations can be performed without further consideration of extended-range arithmetic. However, if IPN(I) .NE. 0 the corresponding value of the normalized Legendre polynomial cannot be represented in single-precision because of overflow or underflow. THE USER MUST TEST IPN(I) IN HIS/HER PROGRAM. In the case that IPN(I) is nonzero, the user should try using double precision if it has a wider exponent range. If double precision fails, the user could rewrite his/her program to use extended-range arithmetic.

The interpretation of (SPN(I),IPN(I)) can be changed to SPN(I)*(10**IPN(I)) by calling the extended-range subroutine XCON. This should be done before printing the computed values. As an example of usage, the Fortran coding

```
      J = K
      DO 20 I = 1, K
      CALL XCON(SPN(I), IPN(I),IERROR)
      IF (IERROR.NE.0) RETURN
      PRINT 10, SPN(I), IPN(I)
   10 FORMAT(1X, E30.8 , I15)
      IF ((IPN(I) .EQ. 0) .OR. (J .LT. K)) GO TO 20
      J = I - 1
   20 CONTINUE
```

will print all computed values and determine the largest J such that IPN(1) = IPN(2) = ... = IPN(J) = 0. Because of the change of representation caused by calling XCON, (SPN(I), IPN(I)) for I = J+1, J+2, ... cannot be used in subsequent extended-range computations.

IERROR is an error indicator. If no errors are detected, IERROR=0 when control returns to the calling routine. If an error is detected, IERROR is returned as nonzero. The calling routine must check the value of IERROR.

If IERROR=112 or 113, invalid input was provided to XNRMP.
If IERROR=101,102,103, or 104, invalid input was provided to XSET.
If IERROR=105 or 106, an internal consistency error occurred in XSET (probably due to a software malfunction in the library routine I1MACH).
If IERROR=107, an overflow or underflow of an extended-range number was detected in XADJ.
If IERROR=108, an overflow or underflow of an extended-range number was detected in XC210.

***SEE ALSO XSET
***REFERENCES Smith, Olver and Lozier, Extended-Range Arithmetic and Normalized Legendre Polynomials, ACM Trans on Math

```
                 Softw, v 7, n 1, March 1981, pp 93--105.
***ROUTINES CALLED  XADD, XADJ, XERMSG, XRED, XSET
***REVISION HISTORY  (YYMMDD)
   820712  DATE WRITTEN
   881020  Revised to meet SLATEC CML recommendations.  (DWL and JMS)
   901019  Revisions to prologue.  (DWL and WRB)
   901106  Changed all specific intrinsics to generic.  (WRB)
           Corrected order of sections in prologue and added TYPE
           section.  (WRB)
           CALLs to XERROR changed to CALLs to XERMSG.  (WRB)
   920127  Revised PURPOSE section of prologue.  (DWL)
   END PROLOGUE
```

XRED

```
      SUBROUTINE XRED (X, IX, IERROR)
***BEGIN PROLOGUE  XRED
***PURPOSE  To provide single-precision floating-point arithmetic
            with an extended exponent range.
***LIBRARY   SLATEC
***CATEGORY  A3D
***TYPE      SINGLE PRECISION (XRED-S, DXRED-D)
***KEYWORDS  EXTENDED-RANGE SINGLE-PRECISION ARITHMETIC
***AUTHOR  Lozier, Daniel W., (National Bureau of Standards)
           Smith, John M., (NBS and George Mason University)
***DESCRIPTION
      REAL X
      INTEGER IX

                  IF
                  RADIX**(-2L) .LE. (ABS(X),IX) .LE. RADIX**(2L)
                  THEN XRED TRANSFORMS (X,IX) SO THAT IX=0.
                  IF (X,IX) IS OUTSIDE THE ABOVE RANGE,
                  THEN XRED TAKES NO ACTION.
                  THIS SUBROUTINE IS USEFUL IF THE
                  RESULTS OF EXTENDED-RANGE CALCULATIONS
                  ARE TO BE USED IN SUBSEQUENT ORDINARY
                  SINGLE-PRECISION CALCULATIONS.

***SEE ALSO  XSET
***REFERENCES  (NONE)
***ROUTINES CALLED  (NONE)
***COMMON BLOCKS    XBLK2
***REVISION HISTORY  (YYMMDD)
   820712  DATE WRITTEN
   881020  Revised to meet SLATEC CML recommendations.  (DWL and JMS)
   901019  Revisions to prologue.  (DWL and WRB)
   901106  Changed all specific intrinsics to generic.  (WRB)
           Corrected order of sections in prologue and added TYPE
           section.  (WRB)
   920127  Revised PURPOSE section of prologue.  (DWL)
   END PROLOGUE
```

XSET

```
     SUBROUTINE XSET (IRAD, NRADPL, DZERO, NBITS, IERROR)
***BEGIN PROLOGUE  XSET
***PURPOSE  To provide single-precision floating-point arithmetic
            with an extended exponent range.
***LIBRARY   SLATEC
***CATEGORY  A3D
***TYPE      SINGLE PRECISION (XSET-S, DXSET-D)
***KEYWORDS  EXTENDED-RANGE SINGLE-PRECISION ARITHMETIC
***AUTHOR  Lozier, Daniel W., (National Bureau of Standards)
           Smith, John M., (NBS and George Mason University)
***DESCRIPTION

   SUBROUTINE  XSET  MUST BE CALLED PRIOR TO CALLING ANY OTHER
 EXTENDED-RANGE SUBROUTINE. IT CALCULATES AND STORES SEVERAL
 MACHINE-DEPENDENT CONSTANTS IN COMMON BLOCKS. THE USER MUST
 SUPPLY FOUR CONSTANTS THAT PERTAIN TO HIS PARTICULAR COMPUTER.
 THE CONSTANTS ARE

          IRAD = THE INTERNAL BASE OF SINGLE-PRECISION
                 ARITHMETIC IN THE COMPUTER.
        NRADPL = THE NUMBER OF RADIX PLACES CARRIED IN
                 THE SINGLE-PRECISION REPRESENTATION.
         DZERO = THE SMALLEST OF 1/DMIN, DMAX, DMAXLN WHERE
                 DMIN = THE SMALLEST POSITIVE SINGLE-PRECISION
                 NUMBER OR AN UPPER BOUND TO THIS NUMBER,
                 DMAX = THE LARGEST SINGLE-PRECISION NUMBER
                 OR A LOWER BOUND TO THIS NUMBER,
                 DMAXLN = THE LARGEST SINGLE-PRECISION NUMBER
                 SUCH THAT LOG10(DMAXLN) CAN BE COMPUTED BY THE
                 FORTRAN SYSTEM (ON MOST SYSTEMS DMAXLN = DMAX).
         NBITS = THE NUMBER OF BITS (EXCLUSIVE OF SIGN) IN
                 AN INTEGER COMPUTER WORD.

 ALTERNATIVELY, ANY OR ALL OF THE CONSTANTS CAN BE GIVEN
 THE VALUE 0 (0.0 FOR DZERO). IF A CONSTANT IS ZERO, XSET TRIES
 TO ASSIGN AN APPROPRIATE VALUE BY CALLING I1MACH
 (SEE P.A.FOX, A.D.HALL, N.L.SCHRYER, ALGORITHM 528 FRAMEWORK
 FOR A PORTABLE LIBRARY, ACM TRANSACTIONS ON MATH SOFTWARE,
 V.4, NO.2, JUNE 1978, 177-188).

   THIS IS THE SETTING-UP SUBROUTINE FOR A PACKAGE OF SUBROUTINES
 THAT FACILITATE THE USE OF EXTENDED-RANGE ARITHMETIC. EXTENDED-RANGE
 ARITHMETIC ON A PARTICULAR COMPUTER IS DEFINED ON THE SET OF NUMBERS
 OF THE FORM

                (X,IX) = X*RADIX**IX

 WHERE X IS A SINGLE-PRECISION NUMBER CALLED THE PRINCIPAL PART,
 IX IS AN INTEGER CALLED THE AUXILIARY INDEX, AND RADIX IS THE
 INTERNAL BASE OF THE SINGLE-PRECISION ARITHMETIC.  OBVIOUSLY,
 EACH REAL NUMBER IS REPRESENTABLE WITHOUT ERROR BY MORE THAN ONE
 EXTENDED-RANGE FORM.  CONVERSIONS BETWEEN  DIFFERENT FORMS ARE
 ESSENTIAL IN CARRYING OUT ARITHMETIC OPERATIONS.  WITH THE CHOICE
 OF RADIX WE HAVE MADE, AND THE SUBROUTINES WE HAVE WRITTEN, THESE
 CONVERSIONS ARE PERFORMED WITHOUT ERROR (AT LEAST ON MOST COMPUTERS).
 (SEE SMITH, J.M., OLVER, F.W.J., AND LOZIER, D.W., EXTENDED-RANGE
```

ARITHMETIC AND NORMALIZED LEGENDRE POLYNOMIALS, ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE, MARCH 1981).

AN EXTENDED-RANGE NUMBER (X,IX) IS SAID TO BE IN ADJUSTED FORM IF X AND IX ARE ZERO OR

```
RADIX**(-L) .LE. ABS(X) .LT. RADIX**L
```

IS SATISFIED, WHERE L IS A COMPUTER-DEPENDENT INTEGER DEFINED IN THIS SUBROUTINE. TWO EXTENDED-RANGE NUMBERS IN ADJUSTED FORM CAN BE ADDED, SUBTRACTED, MULTIPLIED OR DIVIDED (IF THE DIVISOR IS NONZERO) WITHOUT CAUSING OVERFLOW OR UNDERFLOW IN THE PRINCIPAL PART OF THE RESULT. WITH PROPER USE OF THE EXTENDED-RANGE SUBROUTINES, THE ONLY OVERFLOW THAT CAN OCCUR IS INTEGER OVERFLOW IN THE AUXILIARY INDEX. IF THIS IS DETECTED, THE SOFTWARE CALLS XERROR (A GENERAL ERROR-HANDLING FORTRAN SUBROUTINE PACKAGE).

MULTIPLICATION AND DIVISION IS PERFORMED BY SETTING

```
(X,IX)*(Y,IY) = (X*Y,IX+IY)
```

OR

```
(X,IX)/(Y,IY) = (X/Y,IX-IY).
```

PRE-ADJUSTMENT OF THE OPERANDS IS ESSENTIAL TO AVOID OVERFLOW OR UNDERFLOW OF THE PRINCIPAL PART. SUBROUTINE XADJ (SEE BELOW) MAY BE CALLED TO TRANSFORM ANY EXTENDED-RANGE NUMBER INTO ADJUSTED FORM.

ADDITION AND SUBTRACTION REQUIRE THE USE OF SUBROUTINE XADD (SEE BELOW). THE INPUT OPERANDS NEED NOT BE IN ADJUSTED FORM. HOWEVER, THE RESULT OF ADDITION OR SUBTRACTION IS RETURNED IN ADJUSTED FORM. THUS, FOR EXAMPLE, IF (X,IX),(Y,IY), (U,IU), AND (V,IV) ARE IN ADJUSTED FORM, THEN

```
(X,IX)*(Y,IY) + (U,IU)*(V,IV)
```

CAN BE COMPUTED AND STORED IN ADJUSTED FORM WITH NO EXPLICIT CALLS TO XADJ.

WHEN AN EXTENDED-RANGE NUMBER IS TO BE PRINTED, IT MUST BE CONVERTED TO AN EXTENDED-RANGE FORM WITH DECIMAL RADIX. SUBROUTINE XCON IS PROVIDED FOR THIS PURPOSE.

THE SUBROUTINES CONTAINED IN THIS PACKAGE ARE

SUBROUTINE XADD

USAGE

```
CALL XADD(X,IX,Y,IY,Z,IZ,IERROR)
IF (IERROR.NE.0) RETURN
```

DESCRIPTION

FORMS THE EXTENDED-RANGE SUM (Z,IZ) = (X,IX) + (Y,IY). (Z,IZ) IS ADJUSTED BEFORE RETURNING. THE INPUT OPERANDS NEED NOT BE IN ADJUSTED FORM, BUT THEIR PRINCIPAL PARTS MUST SATISFY
RADIX**(-2L).LE.ABS(X).LE.RADIX**(2L),
RADIX**(-2L).LE.ABS(Y).LE.RADIX**(2L).

SUBROUTINE XADJ

USAGE

```
                  CALL XADJ(X,IX,IERROR)
                  IF (IERROR.NE.0) RETURN
 DESCRIPTION
                  TRANSFORMS (X,IX) SO THAT
                  RADIX**(-L) .LE. ABS(X) .LT. RADIX**L.
                  ON MOST COMPUTERS THIS TRANSFORMATION DOES
                  NOT CHANGE THE MANTISSA OF X PROVIDED RADIX IS
                  THE NUMBER BASE OF SINGLE-PRECISION ARITHMETIC.

    SUBROUTINE XC210
 USAGE
                  CALL XC210(K,Z,J,IERROR)
                  IF (IERROR.NE.0) RETURN
 DESCRIPTION
                  GIVEN K THIS SUBROUTINE COMPUTES J AND Z
                  SUCH THAT  RADIX**K = Z*10**J, WHERE Z IS IN
                  THE RANGE 1/10 .LE. Z .LT. 1.
                  THE VALUE OF Z WILL BE ACCURATE TO FULL
                  SINGLE-PRECISION PROVIDED THE NUMBER
                  OF DECIMAL PLACES IN THE LARGEST
                  INTEGER PLUS THE NUMBER OF DECIMAL
                  PLACES CARRIED IN SINGLE-PRECISION DOES NOT
                  EXCEED 60. XC210 IS CALLED BY SUBROUTINE
                  XCON WHEN NECESSARY. THE USER SHOULD
                  NEVER NEED TO CALL XC210 DIRECTLY.

    SUBROUTINE XCON
 USAGE
                  CALL XCON(X,IX,IERROR)
                  IF (IERROR.NE.0) RETURN
 DESCRIPTION
                  CONVERTS (X,IX) = X*RADIX**IX
                  TO DECIMAL FORM IN PREPARATION FOR
                  PRINTING, SO THAT (X,IX) = X*10**IX
                  WHERE 1/10 .LE. ABS(X) .LT. 1
                  IS RETURNED, EXCEPT THAT IF
                  (ABS(X),IX) IS BETWEEN RADIX**(-2L)
                  AND RADIX**(2L) THEN THE REDUCED
                  FORM WITH IX = 0 IS RETURNED.

    SUBROUTINE XRED
 USAGE
                  CALL XRED(X,IX,IERROR)
                  IF (IERROR.NE.0) RETURN
 DESCRIPTION
                  IF
                  RADIX**(-2L) .LE. (ABS(X),IX) .LE. RADIX**(2L)
                  THEN XRED TRANSFORMS (X,IX) SO THAT IX=0.
                  IF (X,IX) IS OUTSIDE THE ABOVE RANGE,
                  THEN XRED TAKES NO ACTION.
                  THIS SUBROUTINE IS USEFUL IF THE
                  RESULTS OF EXTENDED-RANGE CALCULATIONS
                  ARE TO BE USED IN SUBSEQUENT ORDINARY
                  SINGLE-PRECISION CALCULATIONS.
```

***REFERENCES Smith, Olver and Lozier, Extended-Range Arithmetic and
Normalized Legendre Polynomials, ACM Trans on Math
Softw, v 7, n 1, March 1981, pp 93--105.

***ROUTINES CALLED I1MACH, XERMSG
***COMMON BLOCKS XBLK1, XBLK2, XBLK3

```
***REVISION HISTORY  (YYMMDD)
   820712  DATE WRITTEN
   881020  Revised to meet SLATEC CML recommendations.  (DWL and JMS)
   901019  Revisions to prologue.  (DWL and WRB)
   901106  Changed all specific intrinsics to generic.  (WRB)
           Corrected order of sections in prologue and added TYPE
           section.  (WRB)
           CALLs to XERROR changed to CALLs to XERMSG.  (WRB)
   920127  Revised PURPOSE section of prologue.  (DWL)
   END PROLOGUE
```

XSETF

```
      SUBROUTINE XSETF (KONTRL)
***BEGIN PROLOGUE  XSETF
***PURPOSE  Set the error control flag.
***LIBRARY   SLATEC (XERROR)
***CATEGORY  R3A
***TYPE      ALL (XSETF-A)
***KEYWORDS  ERROR, XERROR
***AUTHOR  Jones, R. E., (SNLA)
***DESCRIPTION

      Abstract
         XSETF sets the error control flag value to KONTRL.
         (KONTRL is an input parameter only.)
         The following table shows how each message is treated,
         depending on the values of KONTRL and LEVEL.  (See XERMSG
         for description of LEVEL.)

         If KONTRL is zero or negative, no information other than the
         message itself (including numeric values, if any) will be
         printed.  If KONTRL is positive, introductory messages,
         trace-backs, etc., will be printed in addition to the message.

               ABS(KONTRL)
         LEVEL        0              1              2
         value
           2        fatal          fatal          fatal

           1     not printed      printed         fatal

           0     not printed      printed        printed

          -1     not printed      printed        printed
                                   only           only
                                   once           once

***REFERENCES  R. E. Jones and D. K. Kahaner, XERROR, the SLATEC
                 Error-handling Package, SAND82-0800, Sandia
                 Laboratories, 1982.
***ROUTINES CALLED  J4SAVE, XERMSG
***REVISION HISTORY  (YYMMDD)
   790801  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890531  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900510  Change call to XERRWV to XERMSG.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

XSETUA

```
      SUBROUTINE XSETUA (IUNITA, N)
***BEGIN PROLOGUE  XSETUA
***PURPOSE  Set logical unit numbers (up to 5) to which error
            messages are to be sent.
***LIBRARY   SLATEC (XERROR)
***CATEGORY  R3B
***TYPE      ALL (XSETUA-A)
***KEYWORDS  ERROR, XERROR
***AUTHOR  Jones, R. E., (SNLA)
***DESCRIPTION

      Abstract
         XSETUA may be called to declare a list of up to five
         logical units, each of which is to receive a copy of
         each error message processed by this package.
         The purpose of XSETUA is to allow simultaneous printing
         of each error message on, say, a main output file,
         an interactive terminal, and other files such as graphics
         communication files.

      Description of Parameters
       --Input--
         IUNIT - an array of up to five unit numbers.
                 Normally these numbers should all be different
                 (but duplicates are not prohibited.)
         N     - the number of unit numbers provided in IUNIT
                 must have 1 .LE. N .LE. 5.

***REFERENCES  R. E. Jones and D. K. Kahaner, XERROR, the SLATEC
                 Error-handling Package, SAND82-0800, Sandia
                 Laboratories, 1982.
***ROUTINES CALLED  J4SAVE, XERMSG
***REVISION HISTORY  (YYMMDD)
   790801  DATE WRITTEN
   861211  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900510  Change call to XERRWV to XERMSG.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

XSETUN

```
      SUBROUTINE XSETUN (IUNIT)
***BEGIN PROLOGUE  XSETUN
***PURPOSE  Set output file to which error messages are to be sent.
***LIBRARY   SLATEC (XERROR)
***CATEGORY  R3B
***TYPE      ALL (XSETUN-A)
***KEYWORDS  ERROR, XERROR
***AUTHOR  Jones, R. E., (SNLA)
***DESCRIPTION

      Abstract
         XSETUN sets the output file to which error messages are to
         be sent.  Only one file will be used.  See XSETUA for
         how to declare more than one file.

      Description of Parameter
       --Input--
         IUNIT - an input parameter giving the logical unit number
                 to which error messages are to be sent.

***REFERENCES  R. E. Jones and D. K. Kahaner, XERROR, the SLATEC
                 Error-handling Package, SAND82-0800, Sandia
                 Laboratories, 1982.
***ROUTINES CALLED  J4SAVE
***REVISION HISTORY  (YYMMDD)
   790801  DATE WRITTEN
   861211  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

ZAIRY

```
      SUBROUTINE ZAIRY (ZR, ZI, ID, KODE, AIR, AII, NZ, IERR)
***BEGIN PROLOGUE  ZAIRY
***PURPOSE  Compute the Airy function Ai(z) or its derivative dAi/dz
            for complex argument z.  A scaling option is available
            to help avoid underflow and overflow.
***LIBRARY   SLATEC
***CATEGORY  C10D
***TYPE      COMPLEX (CAIRY-C, ZAIRY-C)
***KEYWORDS  AIRY FUNCTION, BESSEL FUNCTION OF ORDER ONE THIRD,
             BESSEL FUNCTION OF ORDER TWO THIRDS
***AUTHOR  Amos, D. E., (SNL)
***DESCRIPTION

                    ***A DOUBLE PRECISION ROUTINE***
         On KODE=1, ZAIRY computes the complex Airy function Ai(z)
         or its derivative dAi/dz on ID=0 or ID=1 respectively. On
         KODE=2, a scaling option exp(zeta)*Ai(z) or exp(zeta)*dAi/dz
         is provided to remove the exponential decay in -pi/3<arg(z)
         <pi/3 and the exponential growth in pi/3<abs(arg(z))<pi where
         zeta=(2/3)*z**(3/2).

         While the Airy functions Ai(z) and dAi/dz are analytic in
         the whole z-plane, the corresponding scaled functions defined
         for KODE=2 have a cut along the negative real axis.

         Input
           ZR     - DOUBLE PRECISION real part of argument Z
           ZI     - DOUBLE PRECISION imag part of argument Z
           ID     - Order of derivative, ID=0 or ID=1
           KODE   - A parameter to indicate the scaling option
                    KODE=1  returns
                            AI=Ai(z)  on ID=0
                            AI=dAi/dz on ID=1
                            at z=Z
                        =2  returns
                            AI=exp(zeta)*Ai(z)  on ID=0
                            AI=exp(zeta)*dAi/dz on ID=1
                            at z=Z where zeta=(2/3)*z**(3/2)

         Output
           AIR    - DOUBLE PRECISION real part of result
           AII    - DOUBLE PRECISION imag part of result
           NZ     - Underflow indicator
                    NZ=0    Normal return
                    NZ=1    AI=0 due to underflow in
                            -pi/3<arg(Z)<pi/3 on KODE=1
           IERR   - Error flag
                    IERR=0  Normal return     - COMPUTATION COMPLETED
                    IERR=1  Input error       - NO COMPUTATION
                    IERR=2  Overflow          - NO COMPUTATION
                            (Re(Z) too large with KODE=1)
                    IERR=3  Precision warning - COMPUTATION COMPLETED
                            (Result has less than half precision)
                    IERR=4  Precision error   - NO COMPUTATION
                            (Result has no precision)
                    IERR=5  Algorithmic error - NO COMPUTATION
```

(Termination condition not met)

*Long Description:

Ai(z) and dAi/dz are computed from K Bessel functions by

```
       Ai(z) =  c*sqrt(z)*K(1/3,zeta)
      dAi/dz = -c*   z   *K(2/3,zeta)
           c =  1/(pi*sqrt(3))
        zeta =  (2/3)*z**(3/2)
```

when abs(z)>1 and from power series when abs(z)<=1.

In most complex variable computation, one must evaluate elementary functions. When the magnitude of Z is large, losses of significance by argument reduction occur. Consequently, if the magnitude of ZETA=(2/3)*Z**(3/2) exceeds U1=SQRT(0.5/UR), then losses exceeding half precision are likely and an error flag IERR=3 is triggered where UR=MAX(D1MACH(4),1.0D-18) is double precision unit roundoff limited to 18 digits precision. Also, if the magnitude of ZETA is larger than U2=0.5/UR, then all significance is lost and IERR=4. In order to use the INT function, ZETA must be further restricted not to exceed U3=I1MACH(9)=LARGEST INTEGER. Thus, the magnitude of ZETA must be restricted by MIN(U2,U3). In IEEE arithmetic, U1,U2, and U3 are approximately 2.0E+3, 4.2E+6, 2.1E+9 in single precision and 4.7E+7, 2.3E+15, 2.1E+9 in double precision. This makes U2 limiting is single precision and U3 limiting in double precision. This means that the magnitude of Z cannot exceed approximately 3.4E+4 in single precision and 2.1E+6 in double precision. This also means that one can expect to retain, in the worst cases on 32-bit machines, no digits in single precision and only 6 digits in double precision.

The approximate relative error in the magnitude of a complex Bessel function can be expressed as P*10**S where P=MAX(UNIT ROUNDOFF,1.0E-18) is the nominal precision and 10**S represents the increase in error due to argument reduction in the elementary functions. Here, S=MAX(1,ABS(LOG10(ABS(Z))), ABS(LOG10(FNU))) approximately (i.e., S=MAX(1,ABS(EXPONENT OF ABS(Z),ABS(EXPONENT OF FNU))). However, the phase angle may have only absolute accuracy. This is most likely to occur when one component (in magnitude) is larger than the other by several orders of magnitude. If one component is 10**K larger than the other, then one can expect only MAX(ABS(LOG10(P))-K, 0) significant digits; or, stated another way, when K exceeds the exponent of P, no significant digits remain in the smaller component. However, the phase angle retains absolute accuracy because, in complex arithmetic with precision P, the smaller component will not (as a rule) decrease below P times the magnitude of the larger component. In these extreme cases, the principal phase angle is on the order of +P, -P, PI/2-P, or -PI/2+P.

***REFERENCES

1. M. Abramowitz and I. A. Stegun, Handbook of Mathematical Functions, National Bureau of Standards Applied Mathematics Series 55, U. S. Department of Commerce, Tenth Printing (1972) or later.
2. D. E. Amos, Computation of Bessel Functions of

Complex Argument and Large Order, Report SAND83-0643, Sandia National Laboratories, Albuquerque, NM, May 1983.
3. D. E. Amos, A Subroutine Package for Bessel Functions of a Complex Argument and Nonnegative Order, Report SAND85-1018, Sandia National Laboratory, Albuquerque, NM, May 1985.
4. D. E. Amos, A portable package for Bessel functions of a complex argument and nonnegative order, ACM Transactions on Mathematical Software, 12 (September 1986), pp. 265-273.

```
***ROUTINES CALLED  D1MACH, I1MACH, ZABS, ZACAI, ZBKNU, ZEXP, ZSQRT
***REVISION HISTORY  (YYMMDD)
   830501  DATE WRITTEN
   890801  REVISION DATE from Version 3.2
   910415  Prologue converted to Version 4.0 format.  (BAB)
   920128  Category corrected.  (WRB)
   920811  Prologue revised.  (DWL)
   930122  Added ZEXP and ZSQRT to EXTERNAL statement.  (RWC)
   END PROLOGUE
```

ZBESH

```
      SUBROUTINE ZBESH (ZR, ZI, FNU, KODE, M, N, CYR, CYI, NZ, IERR)
***BEGIN PROLOGUE  ZBESH
***PURPOSE  Compute a sequence of the Hankel functions H(m,a,z)
            for superscript m=1 or 2, real nonnegative orders a=b,
            b+1,... where b>0, and nonzero complex argument z.  A
            scaling option is available to help avoid overflow.
***LIBRARY   SLATEC
***CATEGORY  C10A4
***TYPE      COMPLEX (CBESH-C, ZBESH-C)
***KEYWORDS  BESSEL FUNCTIONS OF COMPLEX ARGUMENT,
             BESSEL FUNCTIONS OF THE THIRD KIND, H BESSEL FUNCTIONS,
             HANKEL FUNCTIONS
***AUTHOR  Amos, D. E., (SNL)
***DESCRIPTION

                    ***A DOUBLE PRECISION ROUTINE***
         On KODE=1, ZBESH computes an N member sequence of complex
         Hankel (Bessel) functions CY(L)=H(M,FNU+L-1,Z) for super-
         script M=1 or 2, real nonnegative orders FNU+L-1, L=1,...,
         N, and complex nonzero Z in the cut plane -pi<arg(Z)<=pi
         where Z=ZR+i*ZI.  On KODE=2, CBESH returns the scaled
         functions

            CY(L) = H(M,FNU+L-1,Z)*exp(-(3-2*M)*Z*i),  i**2=-1

         which removes the exponential behavior in both the upper
         and lower half planes.  Definitions and notation are found
         in the NBS Handbook of Mathematical Functions (Ref. 1).

         Input
           ZR     - DOUBLE PRECISION real part of nonzero argument Z
           ZI     - DOUBLE PRECISION imag part of nonzero argument Z
           FNU    - DOUBLE PRECISION initial order, FNU>=0
           KODE   - A parameter to indicate the scaling option
                    KODE=1  returns
                            CY(L)=H(M,FNU+L-1,Z), L=1,...,N
                        =2  returns
                            CY(L)=H(M,FNU+L-1,Z)*exp(-(3-2M)*Z*i),
                            L=1,...,N
           M      - Superscript of Hankel function, M=1 or 2
           N      - Number of terms in the sequence, N>=1

         Output
           CYR    - DOUBLE PRECISION real part of result vector
           CYI    - DOUBLE PRECISION imag part of result vector
           NZ     - Number of underflows set to zero
                    NZ=0    Normal return
                    NZ>0    CY(L)=0 for NZ values of L (if M=1 and
                            Im(Z)>0 or if M=2 and Im(Z)<0, then
                            CY(L)=0 for L=1,...,NZ; in the com-
                            plementary half planes, the underflows
                            may not be in an uninterrupted sequence)
           IERR   - Error flag
                    IERR=0  Normal return     - COMPUTATION COMPLETED
                    IERR=1  Input error       - NO COMPUTATION
                    IERR=2  Overflow          - NO COMPUTATION
```

```
                    (abs(Z) too small and/or FNU+N-1
                    too large)
            IERR=3  Precision warning - COMPUTATION COMPLETED
                    (Result has half precision or less
                    because abs(Z) or FNU+N-1 is large)
            IERR=4  Precision error   - NO COMPUTATION
                    (Result has no precision because
                    abs(Z) or FNU+N-1 is too large)
            IERR=5  Algorithmic error - NO COMPUTATION
                    (Termination condition not met)
```

*Long Description:

The computation is carried out by the formula

```
   H(m,a,z) = (1/t)*exp(-a*t)*K(a,z*exp(-t))
          t = (3-2*m)*i*pi/2
```

where the K Bessel function is computed as described in the prologue to CBESK.

Exponential decay of H(m,a,z) occurs in the upper half z plane for m=1 and the lower half z plane for m=2. Exponential growth occurs in the complementary half planes. Scaling by exp(-(3-2*m)*z*i) removes the exponential behavior in the whole z plane as z goes to infinity.

For negative orders, the formula

```
   H(m,-a,z) = H(m,a,z)*exp((3-2*m)*a*pi*i)
```

can be used.

In most complex variable computation, one must evaluate elementary functions. When the magnitude of Z or FNU+N-1 is large, losses of significance by argument reduction occur. Consequently, if either one exceeds U1=SQRT(0.5/UR), then losses exceeding half precision are likely and an error flag IERR=3 is triggered where UR=MAX(D1MACH(4),1.0D-18) is double precision unit roundoff limited to 18 digits precision. Also, if either is larger than U2=0.5/UR, then all significance is lost and IERR=4. In order to use the INT function, arguments must be further restricted not to exceed the largest machine integer, U3=I1MACH(9). Thus, the magnitude of Z and FNU+N-1 is restricted by MIN(U2,U3). In IEEE arithmetic, U1,U2, and U3 approximate 2.0E+3, 4.2E+6, 2.1E+9 in single precision and 4.7E+7, 2.3E+15 and 2.1E+9 in double precision. This makes U2 limiting in single precision and U3 limiting in double precision. This means that one can expect to retain, in the worst cases on IEEE machines, no digits in single precision and only 6 digits in double precision. Similar considerations hold for other machines.

The approximate relative error in the magnitude of a complex Bessel function can be expressed as P*10**S where P=MAX(UNIT ROUNDOFF,1.0E-18) is the nominal precision and 10**S represents the increase in error due to argument reduction in the elementary functions. Here, S=MAX(1,ABS(LOG10(ABS(Z))), ABS(LOG10(FNU))) approximately (i.e., S=MAX(1,ABS(EXPONENT OF ABS(Z),ABS(EXPONENT OF FNU))). However, the phase angle may

```
         have only absolute accuracy.  This is most likely to occur
         when one component (in magnitude) is larger than the other by
         several orders of magnitude.  If one component is 10**K larger
         than the other, then one can expect only MAX(ABS(LOG10(P))-K,
         0) significant digits; or, stated another way, when K exceeds
         the exponent of P, no significant digits remain in the smaller
         component.  However, the phase angle retains absolute accuracy
         because, in complex arithmetic with precision P, the smaller
         component will not (as a rule) decrease below P times the
         magnitude of the larger component.  In these extreme cases,
         the principal phase angle is on the order of +P, -P, PI/2-P,
         or -PI/2+P.
```


```
***REFERENCES  1. M. Abramowitz and I. A. Stegun, Handbook of Mathe-
                 matical Functions, National Bureau of Standards
                 Applied Mathematics Series 55, U. S. Department
                 of Commerce, Tenth Printing (1972) or later.
               2. D. E. Amos, Computation of Bessel Functions of
                 Complex Argument, Report SAND83-0086, Sandia National
                 Laboratories, Albuquerque, NM, May 1983.
               3. D. E. Amos, Computation of Bessel Functions of
                 Complex Argument and Large Order, Report SAND83-0643,
                 Sandia National Laboratories, Albuquerque, NM, May
                 1983.
               4. D. E. Amos, A Subroutine Package for Bessel Functions
                 of a Complex Argument and Nonnegative Order, Report
                 SAND85-1018, Sandia National Laboratory, Albuquerque,
                 NM, May 1985.
               5. D. E. Amos, A portable package for Bessel functions
                 of a complex argument and nonnegative order, ACM
                 Transactions on Mathematical Software, 12 (September
                 1986), pp. 265-273.
```


```
***ROUTINES CALLED  D1MACH, I1MACH, ZABS, ZACON, ZBKNU, ZBUNK, ZUOIK
***REVISION HISTORY  (YYMMDD)
   830501  DATE WRITTEN
   890801  REVISION DATE from Version 3.2
   910415  Prologue converted to Version 4.0 format.  (BAB)
   920128  Category corrected.  (WRB)
   920811  Prologue revised.  (DWL)
   END PROLOGUE
```

ZBESI

```
      SUBROUTINE ZBESI (ZR, ZI, FNU, KODE, N, CYR, CYI, NZ, IERR)
***BEGIN PROLOGUE  ZBESI
***PURPOSE  Compute a sequence of the Bessel functions I(a,z) for
            complex argument z and real nonnegative orders a=b,b+1,
            b+2,... where b>0.  A scaling option is available to
            help avoid overflow.
***LIBRARY   SLATEC
***CATEGORY  C10B4
***TYPE      COMPLEX (CBESI-C, ZBESI-C)
***KEYWORDS  BESSEL FUNCTIONS OF COMPLEX ARGUMENT, I BESSEL FUNCTIONS,
             MODIFIED BESSEL FUNCTIONS
***AUTHOR  Amos, D. E., (SNL)
***DESCRIPTION

                    ***A DOUBLE PRECISION ROUTINE***
         On KODE=1, ZBESI computes an N-member sequence of complex
         Bessel functions CY(L)=I(FNU+L-1,Z) for real nonnegative
         orders FNU+L-1, L=1,...,N and complex Z in the cut plane
         -pi<arg(Z)<=pi where Z=ZR+i*ZI.  On KODE=2, CBESI returns
         the scaled functions

            CY(L) = exp(-abs(X))*I(FNU+L-1,Z), L=1,...,N and X=Re(Z)

         which removes the exponential growth in both the left and
         right half-planes as Z goes to infinity.

         Input
           ZR     - DOUBLE PRECISION real part of argument Z
           ZI     - DOUBLE PRECISION imag part of argument Z
           FNU    - DOUBLE PRECISION initial order, FNU>=0
           KODE   - A parameter to indicate the scaling option
                    KODE=1  returns
                            CY(L)=I(FNU+L-1,Z), L=1,...,N
                        =2  returns
                            CY(L)=exp(-abs(X))*I(FNU+L-1,Z), L=1,...,N
                            where X=Re(Z)
           N      - Number of terms in the sequence, N>=1

         Output
           CYR    - DOUBLE PRECISION real part of result vector
           CYI    - DOUBLE PRECISION imag part of result vector
           NZ     - Number of underflows set to zero
                    NZ=0    Normal return
                    NZ>0    CY(L)=0, L=N-NZ+1,...,N
           IERR   - Error flag
                    IERR=0  Normal return     - COMPUTATION COMPLETED
                    IERR=1  Input error       - NO COMPUTATION
                    IERR=2  Overflow          - NO COMPUTATION
                            (Re(Z) too large on KODE=1)
                    IERR=3  Precision warning - COMPUTATION COMPLETED
                            (Result has half precision or less
                            because abs(Z) or FNU+N-1 is large)
                    IERR=4  Precision error   - NO COMPUTATION
                            (Result has no precision because
                            abs(Z) or FNU+N-1 is too large)
                    IERR=5  Algorithmic error - NO COMPUTATION
```

(Termination condition not met)

*Long Description:

The computation of I(a,z) is carried out by the power series for small abs(z), the asymptotic expansion for large abs(z), the Miller algorithm normalized by the Wronskian and a Neumann series for intermediate magnitudes of z, and the uniform asymptotic expansions for I(a,z) and J(a,z) for large orders a. Backward recurrence is used to generate sequences or reduce orders when necessary.

The calculations above are done in the right half plane and continued into the left half plane by the formula

```
I(a,z*exp(t)) = exp(t*a)*I(a,z), Re(z)>0
            t = i*pi or -i*pi
```

For negative orders, the formula

```
I(-a,z) = I(a,z) + (2/pi)*sin(pi*a)*K(a,z)
```

can be used. However, for large orders close to integers the the function changes radically. When a is a large positive integer, the magnitude of I(-a,z)=I(a,z) is a large negative power of ten. But when a is not an integer, K(a,z) dominates in magnitude with a large positive power of ten and the most that the second term can be reduced is by unit roundoff from the coefficient. Thus, wide changes can occur within unit roundoff of a large integer for a. Here, large means a>abs(z).

In most complex variable computation, one must evaluate elementary functions. When the magnitude of Z or FNU+N-1 is large, losses of significance by argument reduction occur. Consequently, if either one exceeds U1=SQRT(0.5/UR), then losses exceeding half precision are likely and an error flag IERR=3 is triggered where UR=MAX(D1MACH(4),1.0D-18) is double precision unit roundoff limited to 18 digits precision. Also, if either is larger than U2=0.5/UR, then all significance is lost and IERR=4. In order to use the INT function, arguments must be further restricted not to exceed the largest machine integer, U3=I1MACH(9). Thus, the magnitude of Z and FNU+N-1 is restricted by MIN(U2,U3). In IEEE arithmetic, U1,U2, and U3 approximate 2.0E+3, 4.2E+6, 2.1E+9 in single precision and 4.7E+7, 2.3E+15 and 2.1E+9 in double precision. This makes U2 limiting in single precision and U3 limiting in double precision. This means that one can expect to retain, in the worst cases on IEEE machines, no digits in single precision and only 6 digits in double precision. Similar considerations hold for other machines.

The approximate relative error in the magnitude of a complex Bessel function can be expressed as P*10**S where P=MAX(UNIT ROUNDOFF,1.0E-18) is the nominal precision and 10**S represents the increase in error due to argument reduction in the elementary functions. Here, S=MAX(1,ABS(LOG10(ABS(Z))), ABS(LOG10(FNU))) approximately (i.e., S=MAX(1,ABS(EXPONENT OF ABS(Z),ABS(EXPONENT OF FNU))). However, the phase angle may have only absolute accuracy. This is most likely to occur

when one component (in magnitude) is larger than the other by several orders of magnitude. If one component is 10**K larger than the other, then one can expect only MAX(ABS(LOG10(P))-K, 0) significant digits; or, stated another way, when K exceeds the exponent of P, no significant digits remain in the smaller component. However, the phase angle retains absolute accuracy because, in complex arithmetic with precision P, the smaller component will not (as a rule) decrease below P times the magnitude of the larger component. In these extreme cases, the principal phase angle is on the order of +P, -P, PI/2-P, or -PI/2+P.

***REFERENCES

1. M. Abramowitz and I. A. Stegun, Handbook of Mathematical Functions, National Bureau of Standards Applied Mathematics Series 55, U. S. Department of Commerce, Tenth Printing (1972) or later.
2. D. E. Amos, Computation of Bessel Functions of Complex Argument, Report SAND83-0086, Sandia National Laboratories, Albuquerque, NM, May 1983.
3. D. E. Amos, Computation of Bessel Functions of Complex Argument and Large Order, Report SAND83-0643, Sandia National Laboratories, Albuquerque, NM, May 1983.
4. D. E. Amos, A Subroutine Package for Bessel Functions of a Complex Argument and Nonnegative Order, Report SAND85-1018, Sandia National Laboratory, Albuquerque, NM, May 1985.
5. D. E. Amos, A portable package for Bessel functions of a complex argument and nonnegative order, ACM Transactions on Mathematical Software, 12 (September 1986), pp. 265-273.

***ROUTINES CALLED D1MACH, I1MACH, ZABS, ZBINU

***REVISION HISTORY (YYMMDD)

```
830501  DATE WRITTEN
890801  REVISION DATE from Version 3.2
910415  Prologue converted to Version 4.0 format.  (BAB)
920128  Category corrected.  (WRB)
920811  Prologue revised.  (DWL)
END PROLOGUE
```

ZBESJ

```
      SUBROUTINE ZBESJ (ZR, ZI, FNU, KODE, N, CYR, CYI, NZ, IERR)
***BEGIN PROLOGUE  ZBESJ
***PURPOSE  Compute a sequence of the Bessel functions J(a,z) for
            complex argument z and real nonnegative orders a=b,b+1,
            b+2,... where b>0.  A scaling option is available to
            help avoid overflow.
***LIBRARY   SLATEC
***CATEGORY  C10A4
***TYPE      COMPLEX (CBESJ-C, ZBESJ-C)
***KEYWORDS  BESSEL FUNCTIONS OF COMPLEX ARGUMENT,
             BESSEL FUNCTIONS OF THE FIRST KIND, J BESSEL FUNCTIONS
***AUTHOR  Amos, D. E., (SNL)
***DESCRIPTION

                     ***A DOUBLE PRECISION ROUTINE***
         On KODE=1, ZBESJ computes an N member sequence of complex
         Bessel functions CY(L)=J(FNU+L-1,Z) for real nonnegative
         orders FNU+L-1, L=1,...,N and complex Z in the cut plane
         -pi<arg(Z)<=pi where Z=ZR+i*ZI.  On KODE=2, CBESJ returns
         the scaled functions

            CY(L) = exp(-abs(Y))*J(FNU+L-1,Z),  L=1,...,N and Y=Im(Z)

         which remove the exponential growth in both the upper and
         lower half planes as Z goes to infinity.  Definitions and
         notation are found in the NBS Handbook of Mathematical
         Functions (Ref. 1).

         Input
           ZR     - DOUBLE PRECISION real part of argument Z
           ZI     - DOUBLE PRECISION imag part of argument Z
           FNU    - DOUBLE PRECISION initial order, FNU>=0
           KODE   - A parameter to indicate the scaling option
                    KODE=1  returns
                            CY(L)=J(FNU+L-1,Z), L=1,...,N
                        =2  returns
                            CY(L)=J(FNU+L-1,Z)*exp(-abs(Y)), L=1,...,N
                            where Y=Im(Z)
           N      - Number of terms in the sequence, N>=1

         Output
           CYR    - DOUBLE PRECISION real part of result vector
           CYI    - DOUBLE PRECISION imag part of result vector
           NZ     - Number of underflows set to zero
                    NZ=0    Normal return
                    NZ>0    CY(L)=0, L=N-NZ+1,...,N
           IERR   - Error flag
                    IERR=0  Normal return     - COMPUTATION COMPLETED
                    IERR=1  Input error       - NO COMPUTATION
                    IERR=2  Overflow          - NO COMPUTATION
                            (Im(Z) too large on KODE=1)
                    IERR=3  Precision warning - COMPUTATION COMPLETED
                            (Result has half precision or less
                            because abs(Z) or FNU+N-1 is large)
                    IERR=4  Precision error   - NO COMPUTATION
                            (Result has no precision because
```

```
                    abs(Z) or FNU+N-1 is too large)
          IERR=5  Algorithmic error - NO COMPUTATION
                  (Termination condition not met)
```

*Long Description:

The computation is carried out by the formulae

```
J(a,z) = exp( a*pi*i/2)*I(a,-i*z),  Im(z)>=0

J(a,z) = exp(-a*pi*i/2)*I(a, i*z),  Im(z)<0
```

where the I Bessel function is computed as described in the prologue to CBESI.

For negative orders, the formula

```
J(-a,z) = J(a,z)*cos(a*pi) - Y(a,z)*sin(a*pi)
```

can be used. However, for large orders close to integers, the the function changes radically. When a is a large positive integer, the magnitude of J(-a,z)=J(a,z)*cos(a*pi) is a large negative power of ten. But when a is not an integer, Y(a,z) dominates in magnitude with a large positive power of ten and the most that the second term can be reduced is by unit roundoff from the coefficient. Thus, wide changes can occur within unit roundoff of a large integer for a. Here, large means a>abs(z).

In most complex variable computation, one must evaluate elementary functions. When the magnitude of Z or FNU+N-1 is large, losses of significance by argument reduction occur. Consequently, if either one exceeds U1=SQRT(0.5/UR), then losses exceeding half precision are likely and an error flag IERR=3 is triggered where UR=MAX(D1MACH(4),1.0D-18) is double precision unit roundoff limited to 18 digits precision. Also, if either is larger than U2=0.5/UR, then all significance is lost and IERR=4. In order to use the INT function, arguments must be further restricted not to exceed the largest machine integer, U3=I1MACH(9). Thus, the magnitude of Z and FNU+N-1 is restricted by MIN(U2,U3). In IEEE arithmetic, U1,U2, and U3 approximate 2.0E+3, 4.2E+6, 2.1E+9 in single precision and 4.7E+7, 2.3E+15 and 2.1E+9 in double precision. This makes U2 limiting in single precision and U3 limiting in double precision. This means that one can expect to retain, in the worst cases on IEEE machines, no digits in single precision and only 6 digits in double precision. Similar considerations hold for other machines.

The approximate relative error in the magnitude of a complex Bessel function can be expressed as P*10**S where P=MAX(UNIT ROUNDOFF,1.0E-18) is the nominal precision and 10**S represents the increase in error due to argument reduction in the elementary functions. Here, S=MAX(1,ABS(LOG10(ABS(Z))), ABS(LOG10(FNU))) approximately (i.e., S=MAX(1,ABS(EXPONENT OF ABS(Z),ABS(EXPONENT OF FNU))). However, the phase angle may have only absolute accuracy. This is most likely to occur when one component (in magnitude) is larger than the other by several orders of magnitude. If one component is 10**K larger than the other, then one can expect only MAX(ABS(LOG10(P))-K,

0) significant digits; or, stated another way, when K exceeds the exponent of P, no significant digits remain in the smaller component. However, the phase angle retains absolute accuracy because, in complex arithmetic with precision P, the smaller component will not (as a rule) decrease below P times the magnitude of the larger component. In these extreme cases, the principal phase angle is on the order of +P, -P, PI/2-P, or -PI/2+P.

***REFERENCES

1. M. Abramowitz and I. A. Stegun, Handbook of Mathematical Functions, National Bureau of Standards Applied Mathematics Series 55, U. S. Department of Commerce, Tenth Printing (1972) or later.
2. D. E. Amos, Computation of Bessel Functions of Complex Argument, Report SAND83-0086, Sandia National Laboratories, Albuquerque, NM, May 1983.
3. D. E. Amos, Computation of Bessel Functions of Complex Argument and Large Order, Report SAND83-0643, Sandia National Laboratories, Albuquerque, NM, May 1983.
4. D. E. Amos, A Subroutine Package for Bessel Functions of a Complex Argument and Nonnegative Order, Report SAND85-1018, Sandia National Laboratory, Albuquerque, NM, May 1985.
5. D. E. Amos, A portable package for Bessel functions of a complex argument and nonnegative order, ACM Transactions on Mathematical Software, 12 (September 1986), pp. 265-273.

***ROUTINES CALLED D1MACH, I1MACH, ZABS, ZBINU
***REVISION HISTORY (YYMMDD)
830501 DATE WRITTEN
890801 REVISION DATE from Version 3.2
910415 Prologue converted to Version 4.0 format. (BAB)
920128 Category corrected. (WRB)
920811 Prologue revised. (DWL)
END PROLOGUE

ZBESK

```
      SUBROUTINE ZBESK (ZR, ZI, FNU, KODE, N, CYR, CYI, NZ, IERR)
***BEGIN PROLOGUE  ZBESK
***PURPOSE  Compute a sequence of the Bessel functions K(a,z) for
            complex argument z and real nonnegative orders a=b,b+1,
            b+2,... where b>0.  A scaling option is available to
            help avoid overflow.
***LIBRARY   SLATEC
***CATEGORY  C10B4
***TYPE      COMPLEX (CBESK-C, ZBESK-C)
***KEYWORDS  BESSEL FUNCTIONS OF COMPLEX ARGUMENT, K BESSEL FUNCTIONS,
             MODIFIED BESSEL FUNCTIONS
***AUTHOR  Amos, D. E., (SNL)
***DESCRIPTION

                     ***A DOUBLE PRECISION ROUTINE***
         On KODE=1, ZBESK computes an N member sequence of complex
         Bessel functions CY(L)=K(FNU+L-1,Z) for real nonnegative
         orders FNU+L-1, L=1,...,N and complex Z.NE.0 in the cut
         plane -pi<arg(Z)<=pi where Z=ZR+i*ZI.  On KODE=2, CBESJ
         returns the scaled functions

            CY(L) = exp(Z)*K(FNU+L-1,Z),  L=1,...,N

         which remove the exponential growth in both the left and
         right half planes as Z goes to infinity.  Definitions and
         notation are found in the NBS Handbook of Mathematical
         Functions (Ref. 1).

         Input
           ZR     - DOUBLE PRECISION real part of nonzero argument Z
           ZI     - DOUBLE PRECISION imag part of nonzero argument Z
           FNU    - DOUBLE PRECISION initial order, FNU>=0
           KODE   - A parameter to indicate the scaling option
                    KODE=1  returns
                            CY(L)=K(FNU+L-1,Z), L=1,...,N
                        =2  returns
                            CY(L)=K(FNU+L-1,Z)*EXP(Z), L=1,...,N
           N      - Number of terms in the sequence, N>=1

         Output
           CYR    - DOUBLE PRECISION real part of result vector
           CYI    - DOUBLE PRECISION imag part of result vector
           NZ     - Number of underflows set to zero
                    NZ=0    Normal return
                    NZ>0    CY(L)=0 for NZ values of L (if Re(Z)>0
                            then CY(L)=0 for L=1,...,NZ; in the
                            complementary half plane the underflows
                            may not be in an uninterrupted sequence)
           IERR   - Error flag
                    IERR=0  Normal return     - COMPUTATION COMPLETED
                    IERR=1  Input error       - NO COMPUTATION
                    IERR=2  Overflow          - NO COMPUTATION
                            (abs(Z) too small and/or FNU+N-1
                            too large)
                    IERR=3  Precision warning - COMPUTATION COMPLETED
                            (Result has half precision or less
```

```
                        because abs(Z) or FNU+N-1 is large)
                IERR=4  Precision error   - NO COMPUTATION
                        (Result has no precision because
                        abs(Z) or FNU+N-1 is too large)
                IERR=5  Algorithmic error - NO COMPUTATION
                        (Termination condition not met)
```

*Long Description:

Equations of the reference are implemented to compute K(a,z) for small orders a and a+1 in the right half plane Re(z)>=0. Forward recurrence generates higher orders. The formula

```
K(a,z*exp((t)) = exp(-t)*K(a,z) - t*I(a,z),  Re(z)>0
             t = i*pi or -i*pi
```

continues K to the left half plane.

For large orders, K(a,z) is computed by means of its uniform asymptotic expansion.

For negative orders, the formula

```
K(-a,z) = K(a,z)
```

can be used.

CBESK assumes that a significant digit sinh function is available.

In most complex variable computation, one must evaluate elementary functions. When the magnitude of Z or FNU+N-1 is large, losses of significance by argument reduction occur. Consequently, if either one exceeds U1=SQRT(0.5/UR), then losses exceeding half precision are likely and an error flag IERR=3 is triggered where UR=MAX(D1MACH(4),1.0D-18) is double precision unit roundoff limited to 18 digits precision. Also, if either is larger than U2=0.5/UR, then all significance is lost and IERR=4. In order to use the INT function, arguments must be further restricted not to exceed the largest machine integer, U3=I1MACH(9). Thus, the magnitude of Z and FNU+N-1 is restricted by MIN(U2,U3). In IEEE arithmetic, U1,U2, and U3 approximate 2.0E+3, 4.2E+6, 2.1E+9 in single precision and 4.7E+7, 2.3E+15 and 2.1E+9 in double precision. This makes U2 limiting in single precision and U3 limiting in double precision. This means that one can expect to retain, in the worst cases on IEEE machines, no digits in single precision and only 6 digits in double precision. Similar considerations hold for other machines.

The approximate relative error in the magnitude of a complex Bessel function can be expressed as P*10**S where P=MAX(UNIT ROUNDOFF,1.0E-18) is the nominal precision and 10**S represents the increase in error due to argument reduction in the elementary functions. Here, S=MAX(1,ABS(LOG10(ABS(Z))), ABS(LOG10(FNU))) approximately (i.e., S=MAX(1,ABS(EXPONENT OF ABS(Z),ABS(EXPONENT OF FNU))). However, the phase angle may have only absolute accuracy. This is most likely to occur when one component (in magnitude) is larger than the other by several orders of magnitude. If one component is 10**K larger

```
         than the other, then one can expect only MAX(ABS(LOG10(P))-K,
         0) significant digits; or, stated another way, when K exceeds
         the exponent of P, no significant digits remain in the smaller
         component.  However, the phase angle retains absolute accuracy
         because, in complex arithmetic with precision P, the smaller
         component will not (as a rule) decrease below P times the
         magnitude of the larger component.  In these extreme cases,
         the principal phase angle is on the order of +P, -P, PI/2-P,
         or -PI/2+P.

***REFERENCES  1. M. Abramowitz and I. A. Stegun, Handbook of Mathe-
                 matical Functions, National Bureau of Standards
                 Applied Mathematics Series 55, U. S. Department
                 of Commerce, Tenth Printing (1972) or later.
               2. D. E. Amos, Computation of Bessel Functions of
                 Complex Argument, Report SAND83-0086, Sandia National
                 Laboratories, Albuquerque, NM, May 1983.
               3. D. E. Amos, Computation of Bessel Functions of
                 Complex Argument and Large Order, Report SAND83-0643,
                 Sandia National Laboratories, Albuquerque, NM, May
                 1983.
               4. D. E. Amos, A Subroutine Package for Bessel Functions
                 of a Complex Argument and Nonnegative Order, Report
                 SAND85-1018, Sandia National Laboratory, Albuquerque,
                 NM, May 1985.
               5. D. E. Amos, A portable package for Bessel functions
                 of a complex argument and nonnegative order, ACM
                 Transactions on Mathematical Software, 12 (September
                 1986), pp. 265-273.

***ROUTINES CALLED  D1MACH, I1MACH, ZABS, ZACON, ZBKNU, ZBUNK, ZUOIK
***REVISION HISTORY  (YYMMDD)
   830501  DATE WRITTEN
   890801  REVISION DATE from Version 3.2
   910415  Prologue converted to Version 4.0 format.  (BAB)
   920128  Category corrected.  (WRB)
   920811  Prologue revised.  (DWL)
   END PROLOGUE
```

ZBESY

```
      SUBROUTINE ZBESY (ZR, ZI, FNU, KODE, N, CYR, CYI, NZ, CWRKR,
     +   CWRKI, IERR)
***BEGIN PROLOGUE  ZBESY
***PURPOSE  Compute a sequence of the Bessel functions Y(a,z) for
            complex argument z and real nonnegative orders a=b,b+1,
            b+2,... where b>0.  A scaling option is available to
            help avoid overflow.
***LIBRARY   SLATEC
***CATEGORY  C10A4
***TYPE      COMPLEX (CBESY-C, ZBESY-C)
***KEYWORDS  BESSEL FUNCTIONS OF COMPLEX ARGUMENT,
             BESSEL FUNCTIONS OF SECOND KIND, WEBER'S FUNCTION,
             Y BESSEL FUNCTIONS
***AUTHOR  Amos, D. E., (SNL)
***DESCRIPTION

                     ***A DOUBLE PRECISION ROUTINE***
         On KODE=1, ZBESY computes an N member sequence of complex
         Bessel functions CY(L)=Y(FNU+L-1,Z) for real nonnegative
         orders FNU+L-1, L=1,...,N and complex Z in the cut plane
         -pi<arg(Z)<=pi where Z=ZR+i*ZI.  On KODE=2, CBESY returns
         the scaled functions

            CY(L) = exp(-abs(Y))*Y(FNU+L-1,Z),  L=1,...,N, Y=Im(Z)

         which remove the exponential growth in both the upper and
         lower half planes as Z goes to infinity.  Definitions and
         notation are found in the NBS Handbook of Mathematical
         Functions (Ref. 1).

         Input
           ZR     - DOUBLE PRECISION real part of nonzero argument Z
           ZI     - DOUBLE PRECISION imag part of nonzero argument Z
           FNU    - DOUBLE PRECISION initial order, FNU>=0
           KODE   - A parameter to indicate the scaling option
                    KODE=1  returns
                            CY(L)=Y(FNU+L-1,Z), L=1,...,N
                        =2  returns
                            CY(L)=Y(FNU+L-1,Z)*exp(-abs(Y)), L=1,...,N
                            where Y=Im(Z)
           N      - Number of terms in the sequence, N>=1
           CWRKR  - DOUBLE PRECISION work vector of dimension N
           CWRKI  - DOUBLE PRECISION work vector of dimension N

         Output
           CYR    - DOUBLE PRECISION real part of result vector
           CYI    - DOUBLE PRECISION imag part of result vector
           NZ     - Number of underflows set to zero
                    NZ=0    Normal return
                    NZ>0    CY(L)=0 for NZ values of L, usually on
                            KODE=2 (the underflows may not be in an
                            uninterrupted sequence)
           IERR   - Error flag
                    IERR=0  Normal return     - COMPUTATION COMPLETED
                    IERR=1  Input error       - NO COMPUTATION
                    IERR=2  Overflow          - NO COMPUTATION
```

```
                    (abs(Z) too small and/or FNU+N-1
                    too large)
            IERR=3  Precision warning - COMPUTATION COMPLETED
                    (Result has half precision or less
                    because abs(Z) or FNU+N-1 is large)
            IERR=4  Precision error   - NO COMPUTATION
                    (Result has no precision because
                    abs(Z) or FNU+N-1 is too large)
            IERR=5  Algorithmic error - NO COMPUTATION
                    (Termination condition not met)
```

*Long Description:

The computation is carried out by the formula

```
   Y(a,z) = (H(1,a,z) - H(2,a,z))/(2*i)
```

where the Hankel functions are computed as described in CBESH.

For negative orders, the formula

```
   Y(-a,z) = Y(a,z)*cos(a*pi) + J(a,z)*sin(a*pi)
```

can be used. However, for large orders close to half odd integers the function changes radically. When a is a large positive half odd integer, the magnitude of Y(-a,z)=J(a,z)* sin(a*pi) is a large negative power of ten. But when a is not a half odd integer, Y(a,z) dominates in magnitude with a large positive power of ten and the most that the second term can be reduced is by unit roundoff from the coefficient. Thus, wide changes can occur within unit roundoff of a large half odd integer. Here, large means a>abs(z).

In most complex variable computation, one must evaluate elementary functions. When the magnitude of Z or FNU+N-1 is large, losses of significance by argument reduction occur. Consequently, if either one exceeds U1=SQRT(0.5/UR), then losses exceeding half precision are likely and an error flag IERR=3 is triggered where UR=MAX(D1MACH(4),1.0D-18) is double precision unit roundoff limited to 18 digits precision. Also, if either is larger than U2=0.5/UR, then all significance is lost and IERR=4. In order to use the INT function, arguments must be further restricted not to exceed the largest machine integer, U3=I1MACH(9). Thus, the magnitude of Z and FNU+N-1 is restricted by MIN(U2,U3). In IEEE arithmetic, U1,U2, and U3 approximate 2.0E+3, 4.2E+6, 2.1E+9 in single precision and 4.7E+7, 2.3E+15 and 2.1E+9 in double precision. This makes U2 limiting in single precision and U3 limiting in double precision. This means that one can expect to retain, in the worst cases on IEEE machines, no digits in single precision and only 6 digits in double precision. Similar considerations hold for other machines.

The approximate relative error in the magnitude of a complex Bessel function can be expressed as P*10**S where P=MAX(UNIT ROUNDOFF,1.0E-18) is the nominal precision and 10**S represents the increase in error due to argument reduction in the elementary functions. Here, S=MAX(1,ABS(LOG10(ABS(Z))), ABS(LOG10(FNU))) approximately (i.e., S=MAX(1,ABS(EXPONENT OF ABS(Z),ABS(EXPONENT OF FNU))). However, the phase angle may

```
         have only absolute accuracy.  This is most likely to occur
         when one component (in magnitude) is larger than the other by
         several orders of magnitude.  If one component is 10**K larger
         than the other, then one can expect only MAX(ABS(LOG10(P))-K,
         0) significant digits; or, stated another way, when K exceeds
         the exponent of P, no significant digits remain in the smaller
         component.  However, the phase angle retains absolute accuracy
         because, in complex arithmetic with precision P, the smaller
         component will not (as a rule) decrease below P times the
         magnitude of the larger component.  In these extreme cases,
         the principal phase angle is on the order of +P, -P, PI/2-P,
         or -PI/2+P.
```


```
***REFERENCES  1. M. Abramowitz and I. A. Stegun, Handbook of Mathe-
                 matical Functions, National Bureau of Standards
                 Applied Mathematics Series 55, U. S. Department
                 of Commerce, Tenth Printing (1972) or later.
               2. D. E. Amos, Computation of Bessel Functions of
                 Complex Argument, Report SAND83-0086, Sandia National
                 Laboratories, Albuquerque, NM, May 1983.
               3. D. E. Amos, Computation of Bessel Functions of
                 Complex Argument and Large Order, Report SAND83-0643,
                 Sandia National Laboratories, Albuquerque, NM, May
                 1983.
               4. D. E. Amos, A Subroutine Package for Bessel Functions
                 of a Complex Argument and Nonnegative Order, Report
                 SAND85-1018, Sandia National Laboratory, Albuquerque,
                 NM, May 1985.
               5. D. E. Amos, A portable package for Bessel functions
                 of a complex argument and nonnegative order, ACM
                 Transactions on Mathematical Software, 12 (September
                 1986), pp. 265-273.
```


```
***ROUTINES CALLED  D1MACH, I1MACH, ZBESH
***REVISION HISTORY  (YYMMDD)
   830501  DATE WRITTEN
   890801  REVISION DATE from Version 3.2
   910415  Prologue converted to Version 4.0 format.  (BAB)
   920128  Category corrected.  (WRB)
   920811  Prologue revised.  (DWL)
   END PROLOGUE
```

ZBIRY

```
     SUBROUTINE ZBIRY (ZR, ZI, ID, KODE, BIR, BII, IERR)
***BEGIN PROLOGUE  ZBIRY
***PURPOSE  Compute the Airy function Bi(z) or its derivative dBi/dz
            for complex argument z.  A scaling option is available
            to help avoid overflow.
***LIBRARY   SLATEC
***CATEGORY  C10D
***TYPE      COMPLEX (CBIRY-C, ZBIRY-C)
***KEYWORDS  AIRY FUNCTION, BESSEL FUNCTION OF ORDER ONE THIRD,
             BESSEL FUNCTION OF ORDER TWO THIRDS
***AUTHOR  Amos, D. E., (SNL)
***DESCRIPTION

                      ***A DOUBLE PRECISION ROUTINE***
         On KODE=1, ZBIRY computes the complex Airy function Bi(z)
         or its derivative dBi/dz on ID=0 or ID=1 respectively.
         On KODE=2, a scaling option exp(abs(Re(zeta)))*Bi(z) or
         exp(abs(Re(zeta)))*dBi/dz is provided to remove the
         exponential behavior in both the left and right half planes
         where zeta=(2/3)*z**(3/2).

         The Airy functions Bi(z) and dBi/dz are analytic in the
         whole z-plane, and the scaling option does not destroy this
         property.

         Input
           ZR     - DOUBLE PRECISION real part of argument Z
           ZI     - DOUBLE PRECISION imag part of argument Z
           ID     - Order of derivative, ID=0 or ID=1
           KODE   - A parameter to indicate the scaling option
                    KODE=1  returns
                            BI=Bi(z)  on ID=0
                            BI=dBi/dz on ID=1
                            at z=Z
                        =2  returns
                            BI=exp(abs(Re(zeta)))*Bi(z)  on ID=0
                            BI=exp(abs(Re(zeta)))*dBi/dz on ID=1
                            at z=Z where zeta=(2/3)*z**(3/2)

         Output
           BIR    - DOUBLE PRECISION real part of result
           BII    - DOUBLE PRECISION imag part of result
           IERR   - Error flag
                    IERR=0  Normal return     - COMPUTATION COMPLETED
                    IERR=1  Input error       - NO COMPUTATION
                    IERR=2  Overflow          - NO COMPUTATION
                            (Re(Z) too large with KODE=1)
                    IERR=3  Precision warning - COMPUTATION COMPLETED
                            (Result has less than half precision)
                    IERR=4  Precision error   - NO COMPUTATION
                            (Result has no precision)
                    IERR=5  Algorithmic error - NO COMPUTATION
                            (Termination condition not met)

 *Long Description:
```

Bi(z) and dBi/dz are computed from I Bessel functions by

```
       Bi(z) =  c*sqrt(z)*( I(-1/3,zeta) + I(1/3,zeta) )
      dBi/dz =  c*   z   *( I(-2/3,zeta) + I(2/3,zeta) )
           c =  1/sqrt(3)
        zeta =  (2/3)*z**(3/2)
```

when abs(z)>1 and from power series when abs(z)<=1.

In most complex variable computation, one must evaluate elementary functions. When the magnitude of Z is large, losses of significance by argument reduction occur. Consequently, if the magnitude of ZETA=(2/3)*Z**(3/2) exceeds U1=SQRT(0.5/UR), then losses exceeding half precision are likely and an error flag IERR=3 is triggered where UR=MAX(D1MACH(4),1.0D-18) is double precision unit roundoff limited to 18 digits precision. Also, if the magnitude of ZETA is larger than U2=0.5/UR, then all significance is lost and IERR=4. In order to use the INT function, ZETA must be further restricted not to exceed U3=I1MACH(9)=LARGEST INTEGER. Thus, the magnitude of ZETA must be restricted by MIN(U2,U3). In IEEE arithmetic, U1,U2, and U3 are approximately 2.0E+3, 4.2E+6, 2.1E+9 in single precision and 4.7E+7, 2.3E+15, 2.1E+9 in double precision. This makes U2 limiting is single precision and U3 limiting in double precision. This means that the magnitude of Z cannot exceed approximately 3.4E+4 in single precision and 2.1E+6 in double precision. This also means that one can expect to retain, in the worst cases on 32-bit machines, no digits in single precision and only 6 digits in double precision.

The approximate relative error in the magnitude of a complex Bessel function can be expressed as P*10**S where P=MAX(UNIT ROUNDOFF,1.0E-18) is the nominal precision and 10**S represents the increase in error due to argument reduction in the elementary functions. Here, S=MAX(1,ABS(LOG10(ABS(Z))), ABS(LOG10(FNU))) approximately (i.e., S=MAX(1,ABS(EXPONENT OF ABS(Z),ABS(EXPONENT OF FNU))). However, the phase angle may have only absolute accuracy. This is most likely to occur when one component (in magnitude) is larger than the other by several orders of magnitude. If one component is 10**K larger than the other, then one can expect only MAX(ABS(LOG10(P))-K, 0) significant digits; or, stated another way, when K exceeds the exponent of P, no significant digits remain in the smaller component. However, the phase angle retains absolute accuracy because, in complex arithmetic with precision P, the smaller component will not (as a rule) decrease below P times the magnitude of the larger component. In these extreme cases, the principal phase angle is on the order of +P, -P, PI/2-P, or -PI/2+P.

***REFERENCES

1. M. Abramowitz and I. A. Stegun, Handbook of Mathematical Functions, National Bureau of Standards Applied Mathematics Series 55, U. S. Department of Commerce, Tenth Printing (1972) or later.
2. D. E. Amos, Computation of Bessel Functions of Complex Argument and Large Order, Report SAND83-0643, Sandia National Laboratories, Albuquerque, NM, May 1983.
3. D. E. Amos, A Subroutine Package for Bessel Functions

of a Complex Argument and Nonnegative Order, Report SAND85-1018, Sandia National Laboratory, Albuquerque, NM, May 1985.
4. D. E. Amos, A portable package for Bessel functions of a complex argument and nonnegative order, ACM Transactions on Mathematical Software, 12 (September 1986), pp. 265-273.

```
***ROUTINES CALLED  D1MACH, I1MACH, ZABS, ZBINU, ZDIV, ZSQRT
***REVISION HISTORY  (YYMMDD)
   830501  DATE WRITTEN
   890801  REVISION DATE from Version 3.2
   910415  Prologue converted to Version 4.0 format.  (BAB)
   920128  Category corrected.  (WRB)
   920811  Prologue revised.  (DWL)
   930122  Added ZSQRT to EXTERNAL statement.  (RWC)
   END PROLOGUE
```

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government thereof, and shall not be used for advertising or product endorsement purposes. (C) Copyright 1996 The Regents of the University of California. All rights reserved.

Structural Keyword Index

Keyword	Description
entire	This entire document.
title	The name of this document.
scope	Topics covered in SLATEC5.
availability	Machines on which these routines run.
who	Who to contact for assistance.
introduction	Brief overview of SLATEC5; and other SLATEC documentation.
index	This structural keyword index.
date	The latest revision date for SLATEC5.
revisions	Revision history of this document.

In addition, the name of every subroutine described in SLATEC5 is the keyword and link for retrieving its description. Included are:

Routine Name	Gams Cat.	Function Performed
REBAK	eispack	
REBAKB	eispack	
REDUC	eispack	
REDUC2	eispack	
RF	c	elementary-functions, special-functions
RFFTB1	j1	fast-fourier-transforms
RFFTF1	j1	fast-fourier-transforms
RFFTI1	j1	fast-fourier-transforms
RG	eispack	
RGAUSS	l	pseudo-random-numbers
RGG	eispack	
RJ	c	elementary-functions, special-functions
RPQR79	f	nonlinear-equations
RPZERO	f	nonlinear-equations
RS	eispack	
RSB	eispack	
RSG	eispack	
RSGAB	eispack	
RSGBA	eispack	
RSP	eispack	
RST	eispack	
RT	eispack	
RUNIF	l	pseudo-random-numbers
SASUM	d1a	vector-operations
SAXPY	d1a	vector-operations
SBCG	d2	linear-equations
SBHIN	n	data-handling
SBHIN	n	data-handling
SBOCLS	k	approximation
SBOLS	k	approximation
SCASUM	d1a	vector-operations

SCG	d2	linear-equations
SCGN	d2	linear-equations
SCGS	d2	linear-equations
SCHDC	linpack	cholesky-operations
SCHDD	linpack	cholesky-operations
SCHEX	linpack	cholesky-operations
SCHUD	linpack	cholesky-operations
SCNRM2	d1a	vector-operations
SCOPY	d1a	vector-operations
SCOPYM	d1a	vector-operations
SCOV	k	approximation
SCPPLT	n	data-handling
SCPPLT	n	data-handling
SDASSL	i1	ordinary-differential-equations
SDOT	d1a	vector-operations
SDRIV1	i1	ordinary-differential-equations
SDRIV2	i1	ordinary-differential-equations
SDRIV3	i1	ordinary-differential-equations
SDSDOT	d1a	vector-operations
SEPELI	i2	partial-differential-equations
SEPX4	i2	partial-differential-equations
SGBCO	linpack	general-band
SGBDI	linpack	general-band
SGBFA	linpack	general-band
SGBSL	linpack	general-band
SGECO	linpack	general
SGEDI	linpack	general
SGEEV	d4	eigenvalues, eigenvectors
SGEFA	linpack	general
SGEFS	d2	linear-equations
SGEIR	d2	linear-equations
SGEMM	d1b	matrix-operations
SGEMV	d1b	matrix-operations
SGER	d1b	matrix-operations
SGESL	linpack	general
SGLSS	d9	overdetermined-systems, least-squares
SGMRES	d2	linear-equations
SGTSL	linpack	general-tridiagonal
SINDG	c	elementary-functions, special-functions
SINQB	j1	fast-fourier-transforms
SINQF	j1	fast-fourier-transforms
SINQI	j1	fast-fourier-transforms
SINT	j1	fast-fourier-transforms
SINTI	j1	fast-fourier-transforms
SINTRP	i1	ordinary-differential-equations
SIR	d2	linear-equations
SLLTI2	d2	linear-equations
SLPDOC	d2	linear-equations
SNBCO	d2	linear-equations
SNBDI	d3	determinants
SNBFA	d2	linear-equations
SNBFS	d2	linear-equations
SNBIR	d2	linear-equations
SNBSL	d2	linear-equations
SNLS1	k	approximation
SNLS1E	k	approximation
SNRM2	d1a	vector-operations
SNSQ	f	nonlinear-equations
SNSQE	f	nonlinear-equations
SOMN	d2	linear-equations

SOS	f	nonlinear-equations
SPBCO	linpack	hermitian-positive-definite-band
SPBDI	linpack	hermitian-positive-definite-band
SPBFA	linpack	hermitian-positive-definite-band
SPBSL	linpack	hermitian-positive-definite-band
SPENC	c	elementary-functions, special-functions
SPLP	g	optimization
SPOCO	linpack	hermitian-positive-definite
SPODI	linpack	hermitian-positive-definite
SPOFA	linpack	hermitian-positive-definite
SPOFS	d2	linear-equations
SPOIR	d2	linear-equations
SPOSL	linpack	hermitian-positive-definite
SPPCO	linpack	hermitian-positive-definite
SPPDI	linpack	hermitian-positive-definite
SPPERM	n	data-handling
SPPFA	linpack	hermitian-positive-definite
SPPSL	linpack	hermitian-positive-definite
SPSORT	n	data-handling
SPTSL	linpack	positive-definite-tridiagonal
SQRDC	d5	qr-decomposition
SQRSL	d5	qr-decomposition
SROT	d1a	vector-operations
SROTG	d1a	vector-operations
SROTM	d1a	vector-operations
SROTMG	d1a	vector-operations
SS2LT	d2	linear-equations
SS2Y	d1b	matrix-operations
SSBMV	d1b	matrix-operations
SSCAL	d1a	vector-operations
SSD2S	d2	linear-equations
SSDBCG	d2	linear-equations
SSDCG	d2	linear-equations
SSDCGN	d2	linear-equations
SSDCGS	d2	linear-equations
SSDGMR	d2	linear-equations
SSDI	d1b	matrix-operations
SSDOMN	d2	linear-equations
SSDS	d2	linear-equations
SSDSCL	d2	linear-equations
SSGS	d2	linear-equations
SSICCG	d2	linear-equations
SSICO	linpack	symmetric
SSICS	d2	linear-equations
SSIDI	linpack	symmetric
SSIEV	d4	eigenvalues, eigenvectors
SSIFA	linpack	symmetric
SSILUR	d2	linear-equations
SSILUS	d2	linear-equations
SSISL	linpack	symmetric
SSJAC	d2	linear-equations
SSLI	d2	linear-equations
SSLI2	d2	linear-equations
SSLLTI	d2	linear-equations
SSLUBC	d2	linear-equations
SSLUCN	d2	linear-equations
SSLUCS	d2	linear-equations
SSLUGM	d2	linear-equations
SSLUI	d2	linear-equations
SSLUI2	d2	linear-equations

SSLUI4	d2	linear-equations
SSLUOM	d2	linear-equations
SSLUTI	d2	linear-equations
SSMMI2	d2	linear-equations
SSMMTI	d2	linear-equations
SSMTV	d1b	matrix-operations
SSMV	d1b	matrix-operations
SSORT	n	data-handling
SSPCO	linpack	symmetric
SSPDI	linpack	symmetric
SSPEV	d4	eigenvalues, eigenvectors
SSPFA	linpack	symmetric
SSPMV	d1b	matrix-operations
SSPR	d1b	matrix-operations
SSPR2	d1b	matrix-operations
SSPSL	linpack	symmetric
SSVDC	d6	singular-value-decomposition
SSWAP	d1a	vector-operations
SSYMM	d1b	matrix-operations
SSYMV	d1b	matrix-operations
SSYR	d1b	matrix-operations
SSYR2	d1b	matrix-operations
SSYR2K	d1b	matrix-operations
SSYRK	d1b	matrix-operations
STBMV	d1b	matrix-operations
STBSV	d1b	matrix-operations
STEPS	i1	ordinary-differential-equations
STIN	n	data-handling
STOUT	n	data-handling
STPMV	d1b	matrix-operations
STPSV	d1b	matrix-operations
STRCO	linpack	triangular
STRDI	linpack	triangular
STRMM	d1b	matrix-operations
STRMV	d1b	matrix-operations
STRSL	linpack	triangular
STRSM	d1b	matrix-operations
STRSV	d1b	matrix-operations
TINVIT	eispack	
TQL1	eispack	
TQL2	eispack	
TQLRAT	eispack	
TRBAK1	eispack	
TRBAK3	eispack	
TRED1	eispack	
TRED2	eispack	
TRED3	eispack	
TRIDIB	eispack	
TSTURM	eispack	
ULSIA	d9	overdetermined-systems, least-squares
WNNLS	k	approximation
XADD	a	arithmetic-functions
XADJ	a	arithmetic-functions
XC210	a	arithmetic-functions
XCON	a	arithmetic-functions
XERCLR	r3	diagnostics, error-handling
XERDMP	r3	diagnostics, error-handling
XERMAX	r3	diagnostics, error-handling
XGETF	r3	diagnostics, error-handling
XGETUA	r3	diagnostics, error-handling

XGETUN	r3	diagnostics, error-handling
XLEGF	c	elementary-functions, special functions
XNRMP	c	elementary-functions, special functions
XRED	a	arithmetic-functions
XSET	a	arithmetic-functions
XSETF	r3	diagnostics, error-handling
XSETUA	r3	diagnostics, error-handling
XSETUN	r3	diagnostics, error-handling
ZAIRY	c	elementary-functions, special-functions
ZBESH	c	elementary-functions, special-functions
ZBESI	c	elementary-functions, special-functions
ZBESJ	c	elementary-functions, special-functions
ZBESK	c	elementary-functions, special-functions
ZBESY	c	elementary-functions, special-functions
ZBIRY	c	elementary-functions, special-functions

Date and Revisions

Revision date	Keyword affected	Description of changes
07May96	entire	Text updated for SLATEC version 4.1. Adapted for LC (from NERSC).
31Oct91	background loading-slatec entire	New keyword for document comparisons. New loading instructions for UNICOS, CSOS. Text upgraded to cover SLATEC version 4.0.
30Nov87	entire	Text upgraded to cover SLATEC version 3.1. Page index added; keyword index expanded.
26Oct82	entire	First edition of new writeup.

TRG (07May96)

UCID-19631,19632,19633

Privacy and Legal Notice (URL: http://www.llnl.gov/disclaimer.html)

TRG (07May96) Contact on the OCF: lc-hotline@llnl.gov, on the SCF: lc-hotline@pop.llnl.gov