UCID-19631,19632,19633

SLATEC3 (DACOSH through DS2Y)

Table of Contents

Preface 8
Introduction 9
    Using SLATEC Documentation 9
    Loading SLATEC Under UNICOS 9
Subroutine Descriptions 11
    DACOSH 11
    DAI 12
    DAIE 13
    DASINH 14
    DASUM 15
    DATANH 16
    DAVINT 17
    DAWS 19
    DAXPY 20
    DBCG 21
    DBESI 26
    DBESI0 28
    DBESI1 29
    DBESJ 30
    DBESJ0 32
    DBESJ1 33
    DBESK 34
    DBESK0 36
    DBESK1 37
    DBESKS 38
    DBESY 39
    DBESY0 41
    DBESY1 42
    DBETA 43
    DBETAI 44
    DBFQAD 45
    DBHIN 47
    DBI 50
    DBIE 51
    DBINOM 53
    DBINT4 54
    DBINTK 56
    DBNDAC 58
    DBNDSL 62
    DBOCLS 66
    DBOLS 75
    DBSI0E 82
    DBSI1E 83

DBSK0E 84
DBSK1E 85
DBSKES 86
DBSKIN 87
DBSPDR 89
DBSPEV 91
DBSPPP 93
DBSPVD 95
DBSPVN 97
DBSQAD 99
DBVALU 100
DBVSUP 102
DCBRT 109
DCDOT 110
DCG 111
DCGN 116
DCGS 121
DCHDC 126
DCHDD 128
DCHEX 131
DCHFDV 134
DCHFEV 136
DCHU 138
DCHUD 139
DCKDER 141
DCOPY 143
DCOPYM 144
DCOSDG 145
DCOT 146
DCOV 147
DCPPLT 150
DCSEVL 153
DCV 154
DDASSL 156
DDAWS 172
DDEABM 173
DDEBDF 183
DDERKF 196
DDOT 207
DDRIV1 208
DDRIV2 213
DDRIV3 219
DE1 232
DEABM 234
DEBDF 244
DEFC 257

DEI 261
DERF 262
DERFC 263
DERKF 264
DEXINT 274
DEXPRL 276
DFAC 277
DFC 278
DFZERO 285
DGAMI 287
DGAMIC 288
DGAMIT 289
DGAMLM 290
DGAMMA 291
DGAMR 292
DGAUS8 293
DGBCO 295
DGBDI 298
DGBFA 300
DGBMV 302
DGBSL 305
DGECO 307
DGEDI 309
DGEFA 311
DGEFS 313
DGEMM 315
DGEMV 318
DGER 320
DGESL 322
DGLSS 324
DGMRES 327
DGTSL 334
DHFTI 336
DINTP 339
DINTRV 340
DIR 342
DLBETA 347
DLGAMS 348
DLI 349
DLLSIA 350
DLLTI2 353
DLNGAM 355
DLNREL 356
DLPDOC 357
DLSEI 365
DNBCO 372

DNBDI 375
DNBFA 376
DNBFS 378
DNBSL 381
DNLS1 383
DNLS1E 394
DNRM2 403
DNSQ 405
DNSQE 413
DOMN 419
DP1VLU 424
DPBCO 425
DPBDI 427
DPBFA 428
DPBSL 430
DPCHBS 432
DPCHCM 435
DPCHFD 438
DPCHFE 441
DPCHIA 443
DPCHIC 445
DPCHID 449
DPCHIM 451
DPCHSP 454
DPCOEF 457
DPFQAD 458
DPLINT 460
DPOCH 461
DPOCH1 462
DPOCO 463
DPODI 465
DPOFA 467
DPOFS 468
DPOLCF 470
DPOLFT 472
DPOLVL 475
DPOSL 477
DPPCO 479
DPPDI 481
DPPERM 483
DPPFA 484
DPPQAD 486
DPPSL 487
DPPVAL 489
DPSI 491
DPSIFN 492

DPSORT 494
DPTSL 496
DQAG 497
DQAGE 500
DQAGI 503
DQAGIE 506
DQAGP 509
DQAGPE 513
DQAGS 517
DQAGSE 520
DQAWC 523
DQAWCE 526
DQAWF 529
DQAWFE 533
DQAWO 537
DQAWOE 541
DQAWS 545
DQAWSE 548
DQC25C 551
DQC25F 553
DQC25S 555
DQDOTA 557
DQDOTI 559
DQK15 561
DQK15I 563
DQK15W 565
DQK21 567
DQK31 569
DQK41 571
DQK51 573
DQK61 575
DQMOMO 577
DQNC79 579
DQNG 581
DQRDC 583
DQRSL 585
DRC 588
DRC3JJ 593
DRC3JM 596
DRC6J 599
DRD 602
DRF 608
DRJ 613
DROT 618
DROTG 619
DROTM 621

DROTMG 623
DS2LT 625
DS2Y 627
Disclaimer 630
Structural Keyword Index 631
Date and Revisions 636

Preface

Scope: SLATEC3 contains brief descriptions ("prologues") for the SLATEC (version 4.1) mathematical library subroutines with names from DACOSH through DS2Y.

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/slatec3/slatec3.pdf
on the SCF: https://lc.llnl.gov/LCdocs/slatec3/slatec3_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 (THIS DOCUMENT) 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 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

DACOSH

```
      DOUBLE PRECISION FUNCTION DACOSH (X)
***BEGIN PROLOGUE  DACOSH
***PURPOSE  Compute the arc hyperbolic cosine.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C4C
***TYPE      DOUBLE PRECISION (ACOSH-S, DACOSH-D, CACOSH-C)
***KEYWORDS  ACOSH, ARC HYPERBOLIC COSINE, ELEMENTARY FUNCTIONS, FNLIB,
             INVERSE HYPERBOLIC COSINE
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DACOSH(X) calculates the double precision arc hyperbolic cosine for
 double precision argument X.  The result is returned on the
 positive branch.

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, XERMSG
***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)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   END PROLOGUE
```

DAI

```
      DOUBLE PRECISION FUNCTION DAI (X)
***BEGIN PROLOGUE  DAI
***PURPOSE  Evaluate the Airy function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10D
***TYPE      DOUBLE PRECISION (AI-S, DAI-D)
***KEYWORDS  AIRY FUNCTION, FNLIB, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DAI(X) calculates the double precision Airy function for double
 precision argument X.

 Series for AIF        on the interval -1.00000E+00 to  1.00000E+00
                                        with weighted error   8.37E-33
                                         log weighted error  32.08
                                 significant figures required  30.87
                                      decimal places required  32.63

 Series for AIG        on the interval -1.00000E+00 to  1.00000E+00
                                        with weighted error   7.47E-34
                                         log weighted error  33.13
                                 significant figures required  31.50
                                      decimal places required  33.68

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, D9AIMP, DAIE, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   920618  Removed space from variable names.  (RWC, WRB)
   END PROLOGUE
```

DAIE

```
      DOUBLE PRECISION FUNCTION DAIE (X)
***BEGIN PROLOGUE  DAIE
***PURPOSE  Calculate the Airy function for a negative argument and an
            exponentially scaled Airy function for a non-negative
            argument.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10D
***TYPE      DOUBLE PRECISION (AIE-S, DAIE-D)
***KEYWORDS  EXPONENTIALLY SCALED AIRY FUNCTION, FNLIB,
             SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DAIE(X) calculates the Airy function or the exponentially scaled
 Airy function depending on the value of the argument.  The function
 and argument are both double precision.

 Evaluate AI(X) for X .LE. 0.0 and AI(X)*EXP(ZETA) where
 ZETA = 2/3 * X**(3/2)  for X .GE. 0.0

 Series for AIF        on the interval -1.00000E+00 to  1.00000E+00
                                        with weighted error   8.37E-33
                                         log weighted error  32.08
                                significant figures required  30.87
                                     decimal places required  32.63

 Series for AIG        on the interval -1.00000E+00 to  1.00000E+00
                                        with weighted error   7.47E-34
                                         log weighted error  33.13
                                significant figures required  31.50
                                     decimal places required  33.68

 Series for AIP1       on the interval  1.25000E-01 to  1.00000E+00
                                        with weighted error   3.69E-32
                                         log weighted error  31.43
                                significant figures required  29.55
                                     decimal places required  32.31

 Series for AIP2       on the interval  0.          to  1.25000E-01
                                        with weighted error   3.48E-32
                                         log weighted error  31.46
                                significant figures required  28.74
                                     decimal places required  32.24

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, D9AIMP, DCSEVL, INITDS
***REVISION HISTORY  (YYMMDD)
   770701  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)
   920618  Removed space from variable names.  (RWC, WRB)
   END PROLOGUE
```

DASINH

```
      DOUBLE PRECISION FUNCTION DASINH (X)
***BEGIN PROLOGUE  DASINH
***PURPOSE  Compute the arc hyperbolic sine.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C4C
***TYPE      DOUBLE PRECISION (ASINH-S, DASINH-D, CASINH-C)
***KEYWORDS  ARC HYPERBOLIC SINE, ASINH, ELEMENTARY FUNCTIONS, FNLIB,
             INVERSE HYPERBOLIC SINE
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DASINH(X) calculates the double precision arc hyperbolic
 sine for double precision argument X.

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DCSEVL, INITDS
***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
```

DASUM

```
      DOUBLE PRECISION FUNCTION DASUM (N, DX, INCX)
***BEGIN PROLOGUE  DASUM
***PURPOSE  Compute the sum of the magnitudes of the elements of a
            vector.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A3A
***TYPE      DOUBLE 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)
       DX  double precision vector with N elements
     INCX  storage spacing between elements of DX

     --Output--
    DASUM  double precision result (zero if N .LE. 0)

     Returns sum of magnitudes of double precision DX.
     DASUM = sum from 0 to N-1 of ABS(DX(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
   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)
   900821  Modified to correct problem with a negative increment.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DATANH

```
      DOUBLE PRECISION FUNCTION DATANH (X)
***BEGIN PROLOGUE  DATANH
***PURPOSE  Compute the arc hyperbolic tangent.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C4C
***TYPE      DOUBLE PRECISION (ATANH-S, DATANH-D, CATANH-C)
***KEYWORDS  ARC HYPERBOLIC TANGENT, ATANH, ELEMENTARY FUNCTIONS,
             FNLIB, INVERSE HYPERBOLIC TANGENT
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DATANH(X) calculates the double precision arc hyperbolic
 tangent for double precision argument X.

 Series for ATNH       on the interval  0.          to  2.50000E-01
                                       with weighted error   6.86E-32
                                        log weighted error  31.16
                                 significant figures required  30.00
                                      decimal places required  31.88

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DCSEVL, INITDS, XERMSG
***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)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   END PROLOGUE
```

DAVINT

```
      SUBROUTINE DAVINT (X, Y, N, XLO, XUP, ANS, IERR)
***BEGIN PROLOGUE  DAVINT
***PURPOSE  Integrate a function tabulated at arbitrarily spaced
            abscissas using overlapping parabolas.
***LIBRARY   SLATEC
***CATEGORY  H2A1B2
***TYPE      DOUBLE PRECISION (AVINT-S, DAVINT-D)
***KEYWORDS  INTEGRATION, QUADRATURE, TABULATED DATA
***AUTHOR  Jones, R. E., (SNLA)
***DESCRIPTION

      Abstract
          DAVINT integrates a function tabulated at arbitrarily spaced
          abscissas.  The limits of integration need not coincide
          with the tabulated abscissas.

          A method of overlapping parabolas fitted to the data is used
          provided that there are at least 3 abscissas between the
          limits of integration.  DAVINT also handles two special cases.
          If the limits of integration are equal, DAVINT returns a
          result of zero regardless of the number of tabulated values.
          If there are only two function values, DAVINT uses the
          trapezoid rule.

      Description of Parameters
          The user must dimension all arrays appearing in the call list
               X(N), Y(N)

          Input--
       X    - DOUBLE PRECISION array of abscissas, which must be in
              increasing order.
       Y    - DOUBLE PRECISION array of function values. i.e.,
                 Y(I)=FUNC(X(I))
       N    - The integer number of function values supplied.
                 N .GE. 2 unless XLO = XUP.
       XLO  - DOUBLE PRECISION lower limit of integration
       XUP  - DOUBLE PRECISION upper limit of integration.  Must have
               XLO.LE.XUP

          Output--
       ANS  - Double Precision computed approximate value of integral
       IERR - A status code
            --Normal Code
                 =1 Means the requested integration was performed.
            --Abnormal Codes
                 =2 Means XUP was less than XLO.
                 =3 Means the number of X(I) between XLO and XUP
                    (inclusive) was less than 3 and neither of the two
                    special cases described in the abstract occurred.
                    No integration was performed.
                 =4 Means the restriction X(I+1).GT.X(I) was violated.
                 =5 Means the number N of function values was .lt. 2.
                    ANS is set to zero if IERR=2,3,4,or 5.

    DAVINT is documented completely in SC-M-69-335
    Original program from *Numerical Integration* by Davis & Rabinowitz
```

```
    Adaptation and modifications by Rondall E Jones.

***REFERENCES  R. E. Jones, Approximate integrator of functions
                 tabulated at arbitrarily spaced abscissas,
                 Report SC-M-69-335, Sandia Laboratories, 1969.
***ROUTINES CALLED  XERMSG
***REVISION HISTORY  (YYMMDD)
   690901  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
```

DAWS

```
      FUNCTION DAWS (X)
***BEGIN PROLOGUE  DAWS
***PURPOSE  Compute Dawson's function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C8C
***TYPE      SINGLE PRECISION (DAWS-S, DDAWS-D)
***KEYWORDS  DAWSON'S FUNCTION, FNLIB, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DAWS(X) calculates Dawson's integral for real argument X.

 Series for DAW        on the interval  0.          to  1.00000D+00
                                        with weighted error   3.83E-17
                                         log weighted error  16.42
                                 significant figures required  15.78
                                      decimal places required  16.97

 Series for DAW2       on the interval  0.          to  1.60000D+01
                                        with weighted error   5.17E-17
                                         log weighted error  16.29
                                 significant figures required  15.90
                                      decimal places required  17.02

 Series for DAWA       on the interval  0.          to  6.25000D-02
                                        with weighted error   2.24E-17
                                         log weighted error  16.65
                                 significant figures required  14.73
                                      decimal places required  17.36

***REFERENCES  (NONE)
***ROUTINES CALLED  CSEVL, INITS, R1MACH, XERMSG
***REVISION HISTORY  (YYMMDD)
   780401  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)
   920618  Removed space from variable names.  (RWC, WRB)
   END PROLOGUE
```

DAXPY

```
      SUBROUTINE DAXPY (N, DA, DX, INCX, DY, INCY)
***BEGIN PROLOGUE  DAXPY
***PURPOSE  Compute a constant times a vector plus a vector.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A7
***TYPE      DOUBLE 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)
       DA  double precision scalar multiplier
       DX  double precision vector with N elements
     INCX  storage spacing between elements of DX
       DY  double precision vector with N elements
     INCY  storage spacing between elements of DY

     --Output--
       DY  double precision result (unchanged if N .LE. 0)

     Overwrite double precision DY with double precision DA*DX + DY.
     For I = 0 to N-1, replace  DY(LY+I*INCY) with DA*DX(LX+I*INCX) +
       DY(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
```

DBCG

```
      SUBROUTINE DBCG(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  DBCG
***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      DOUBLE 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)
      DOUBLE PRECISION B(N), X(N), A(NELT), TOL, ERR, R(N), Z(N), P(N)
      DOUBLE PRECISION RR(N), ZZ(N), PP(N), DZ(N)
      DOUBLE PRECISION RWORK(USER DEFINED)
      EXTERNAL MATVEC, MTTVEC, MSOLVE, MTSOLV

      CALL DBCG(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       Double Precision B(N).
         Right-hand side vector.
 X      :INOUT    Double Precision 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       Double Precision 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 double precision 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 double precision 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 /DSLBLK/ 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 /DSLBLK/ 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    Double Precision.
        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      Double Precision.
        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*D1MACH(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     Double Precision R(N).
Z      :WORK     Double Precision Z(N).
P      :WORK     Double Precision P(N).
RR     :WORK     Double Precision RR(N).
ZZ     :WORK     Double Precision ZZ(N).
PP     :WORK     Double Precision PP(N).
DZ     :WORK     Double Precision DZ(N).
        Double Precision arrays used for workspace.
RWORK  :WORK     Double Precision RWORK(USER DEFINED).
        Double Precision 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 DSDBCG and DSLUBC 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 double precision 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, and 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.

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  DSDBCG, DSLUBC
***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  D1MACH, DAXPY, DCOPY, DDOT, ISDBCG
***REVISION HISTORY  (YYMMDD)
   890404  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)
   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 DSLBLK.  (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
```

DBESI

```
      SUBROUTINE DBESI (X, ALPHA, KODE, N, Y, NZ)
***BEGIN PROLOGUE  DBESI
***PURPOSE  Compute an N member sequence of I Bessel functions
            I/SUB(ALPHA+K-1)/(X), K=1,...,N or scaled Bessel functions
            EXP(-X)*I/SUB(ALPHA+K-1)/(X), K=1,...,N for nonnegative
            ALPHA and X.
***LIBRARY   SLATEC
***CATEGORY  C10B3
***TYPE      DOUBLE PRECISION (BESI-S, DBESI-D)
***KEYWORDS  I BESSEL FUNCTION, SPECIAL FUNCTIONS
***AUTHOR  Amos, D. E., (SNLA)
           Daniel, S. L., (SNLA)
***DESCRIPTION

     Abstract  **** a double precision routine ****
         DBESI computes an N member sequence of I Bessel functions
         I/sub(ALPHA+K-1)/(X), K=1,...,N or scaled Bessel functions
         EXP(-X)*I/sub(ALPHA+K-1)/(X), K=1,...,N for nonnegative ALPHA
         and X.  A combination of the power series, the asymptotic
         expansion for X to infinity, and the uniform asymptotic
         expansion for NU to infinity are applied over subdivisions of
         the (NU,X) plane.  For values not covered by one of these
         formulae, the order is incremented by an integer so that one
         of these formulae apply.  Backward recursion is used to reduce
         orders by integer values.  The asymptotic expansion for X to
         infinity is used only when the entire sequence (specifically
         the last member) lies within the region covered by the
         expansion.  Leading terms of these expansions are used to test
         for over or underflow where appropriate.  If a sequence is
         requested and the last member would underflow, the result is
         set to zero and the next lower order tried, etc., until a
         member comes on scale or all are set to zero.  An overflow
         cannot occur with scaling.

         The maximum number of significant digits obtainable
         is the smaller of 14 and the number of digits carried in
         double precision arithmetic.

     Description of Arguments

         Input      X,ALPHA are double precision
           X      - X .GE. 0.0D0
           ALPHA  - order of first member of the sequence,
                    ALPHA .GE. 0.0D0
           KODE   - a parameter to indicate the scaling option
                    KODE=1 returns
                           Y(K)=        I/sub(ALPHA+K-1)/(X),
                                K=1,...,N
                    KODE=2 returns
                           Y(K)=EXP(-X)*I/sub(ALPHA+K-1)/(X),
                                K=1,...,N
           N      - number of members in the sequence, N .GE. 1

         Output     Y is double precision
           Y      - a vector whose first N components contain
                    values for I/sub(ALPHA+K-1)/(X) or scaled
```

```
                    values for EXP(-X)*I/sub(ALPHA+K-1)/(X),
                    K=1,...,N depending on KODE
          NZ     - number of components of Y set to zero due to
                    underflow,
                    NZ=0   , normal return, computation completed
                    NZ .NE. 0, last NZ components of Y set to zero,
                             Y(K)=0.0D0, K=N-NZ+1,...,N.

     Error Conditions
         Improper input arguments - a fatal error
         Overflow with KODE=1 - a fatal error
         Underflow - a non-fatal error(NZ .NE. 0)

***REFERENCES  D. E. Amos, S. L. Daniel and M. K. Weston, CDC 6600
                 subroutines IBESS and JBESS for Bessel functions
                 I(NU,X) and J(NU,X), X .GE. 0, NU .GE. 0, ACM
                 Transactions on Mathematical Software 3, (1977),
                 pp. 76-92.
               F. W. J. Olver, Tables of Bessel Functions of Moderate
                 or Large Orders, NPL Mathematical Tables 6, Her
                 Majesty's Stationery Office, London, 1962.
***ROUTINES CALLED  D1MACH, DASYIK, DLNGAM, I1MACH, XERMSG
***REVISION HISTORY  (YYMMDD)
   750101  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  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)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DBESI0

```
      DOUBLE PRECISION FUNCTION DBESI0 (X)
***BEGIN PROLOGUE  DBESI0
***PURPOSE  Compute the hyperbolic Bessel function of the first kind
            of order zero.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10B1
***TYPE      DOUBLE PRECISION (BESI0-S, DBESI0-D)
***KEYWORDS  FIRST KIND, FNLIB, HYPERBOLIC BESSEL FUNCTION,
             MODIFIED BESSEL FUNCTION, ORDER ZERO, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBESI0(X) calculates the double precision modified (hyperbolic)
 Bessel function of the first kind of order zero and double
 precision argument X.

 Series for BI0        on the interval  0.          to  9.00000E+00
                                       with weighted error   9.51E-34
                                        log weighted error  33.02
                                 significant figures required  33.31
                                      decimal places required  33.65

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DBSI0E, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   END PROLOGUE
```

DBESI1

```
      DOUBLE PRECISION FUNCTION DBESI1 (X)
***BEGIN PROLOGUE  DBESI1
***PURPOSE  Compute the modified (hyperbolic) Bessel function of the
            first kind of order one.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10B1
***TYPE      DOUBLE PRECISION (BESI1-S, DBESI1-D)
***KEYWORDS  FIRST KIND, FNLIB, HYPERBOLIC BESSEL FUNCTION,
             MODIFIED BESSEL FUNCTION, ORDER ONE, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBESI1(X) calculates the double precision modified (hyperbolic)
 Bessel function of the first kind of order one and double precision
 argument X.

 Series for BI1        on the interval  0.          to  9.00000E+00
                                       with weighted error   1.44E-32
                                        log weighted error  31.84
                                 significant figures required  31.45
                                      decimal places required  32.46

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DBSI1E, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   END PROLOGUE
```

DBESJ

```
      SUBROUTINE DBESJ (X, ALPHA, N, Y, NZ)
***BEGIN PROLOGUE  DBESJ
***PURPOSE  Compute an N member sequence of J Bessel functions
            J/SUB(ALPHA+K-1)/(X), K=1,...,N for non-negative ALPHA
            and X.
***LIBRARY   SLATEC
***CATEGORY  C10A3
***TYPE      DOUBLE PRECISION (BESJ-S, DBESJ-D)
***KEYWORDS  J BESSEL FUNCTION, SPECIAL FUNCTIONS
***AUTHOR  Amos, D. E., (SNLA)
           Daniel, S. L., (SNLA)
           Weston, M. K., (SNLA)
***DESCRIPTION

      Abstract  **** a double precision routine ****
          DBESJ computes an N member sequence of J Bessel functions
          J/sub(ALPHA+K-1)/(X), K=1,...,N for non-negative ALPHA and X.
          A combination of the power series, the asymptotic expansion
          for X to infinity and the uniform asymptotic expansion for
          NU to infinity are applied over subdivisions of the (NU,X)
          plane.  For values of (NU,X) not covered by one of these
          formulae, the order is incremented or decremented by integer
          values into a region where one of the formulae apply. Backward
          recursion is applied to reduce orders by integer values except
          where the entire sequence lies in the oscillatory region.  In
          this case forward recursion is stable and values from the
          asymptotic expansion for X to infinity start the recursion
          when it is efficient to do so. Leading terms of the series and
          uniform expansion are tested for underflow.  If a sequence is
          requested and the last member would underflow, the result is
          set to zero and the next lower order tried, etc., until a
          member comes on scale or all members are set to zero.
          Overflow cannot occur.

          The maximum number of significant digits obtainable
          is the smaller of 14 and the number of digits carried in
          double precision arithmetic.

      Description of Arguments

          Input      X,ALPHA are double precision
            X      - X .GE. 0.0D0
            ALPHA  - order of first member of the sequence,
                     ALPHA .GE. 0.0D0
            N      - number of members in the sequence, N .GE. 1

          Output     Y is double precision
            Y      - a vector whose first N components contain
                     values for J/sub(ALPHA+K-1)/(X), K=1,...,N
            NZ     - number of components of Y set to zero due to
                     underflow,
                     NZ=0   , normal return, computation completed
                     NZ .NE. 0, last NZ components of Y set to zero,
                              Y(K)=0.0D0, K=N-NZ+1,...,N.

      Error Conditions
```

```
         Improper input arguments - a fatal error
         Underflow  - a non-fatal error (NZ .NE. 0)

***REFERENCES  D. E. Amos, S. L. Daniel and M. K. Weston, CDC 6600
                 subroutines IBESS and JBESS for Bessel functions
                 I(NU,X) and J(NU,X), X .GE. 0, NU .GE. 0, ACM
                 Transactions on Mathematical Software 3, (1977),
                 pp. 76-92.
               F. W. J. Olver, Tables of Bessel Functions of Moderate
                 or Large Orders, NPL Mathematical Tables 6, Her
                 Majesty's Stationery Office, London, 1962.
***ROUTINES CALLED  D1MACH, DASYJY, DJAIRY, DLNGAM, I1MACH, XERMSG
***REVISION HISTORY  (YYMMDD)
   750101  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  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)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DBESJ0

```
      DOUBLE PRECISION FUNCTION DBESJ0 (X)
***BEGIN PROLOGUE  DBESJ0
***PURPOSE  Compute the Bessel function of the first kind of order
            zero.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10A1
***TYPE      DOUBLE PRECISION (BESJ0-S, DBESJ0-D)
***KEYWORDS  BESSEL FUNCTION, FIRST KIND, FNLIB, ORDER ZERO,
             SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBESJ0(X) calculates the double precision Bessel function of
 the first kind of order zero for double precision argument X.

 Series for BJ0        on the interval  0.          to  1.60000E+01
                                       with weighted error   4.39E-32
                                        log weighted error  31.36
                                 significant figures required  31.21
                                      decimal places required  32.00

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, D9B0MP, DCSEVL, INITDS
***REVISION HISTORY  (YYMMDD)
   770701  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
```

DBESJ1

```
      DOUBLE PRECISION FUNCTION DBESJ1 (X)
***BEGIN PROLOGUE  DBESJ1
***PURPOSE  Compute the Bessel function of the first kind of order one.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10A1
***TYPE      DOUBLE PRECISION (BESJ1-S, DBESJ1-D)
***KEYWORDS  BESSEL FUNCTION, FIRST KIND, FNLIB, ORDER ONE,
             SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBESJ1(X) calculates the double precision Bessel function of the
 first kind of order one for double precision argument X.

 Series for BJ1        on the interval  0.          to  1.60000E+01
                                       with weighted error   1.16E-33
                                        log weighted error  32.93
                                 significant figures required  32.36
                                      decimal places required  33.57

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, D9B1MP, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   780601  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)
   910401  Corrected error in code which caused values to have the
           wrong sign for arguments less than 4.0.  (WRB)
   END PROLOGUE
```

DBESK

```
      SUBROUTINE DBESK (X, FNU, KODE, N, Y, NZ)
***BEGIN PROLOGUE  DBESK
***PURPOSE  Implement forward recursion on the three term recursion
            relation for a sequence of non-negative order Bessel
            functions K/SUB(FNU+I-1)/(X), or scaled Bessel functions
            EXP(X)*K/SUB(FNU+I-1)/(X), I=1,...,N for real, positive
            X and non-negative orders FNU.
***LIBRARY   SLATEC
***CATEGORY  C10B3
***TYPE      DOUBLE PRECISION (BESK-S, DBESK-D)
***KEYWORDS  K BESSEL FUNCTION, SPECIAL FUNCTIONS
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Abstract  **** a double precision routine ****
          DBESK implements forward recursion on the three term
          recursion relation for a sequence of non-negative order Bessel
          functions K/sub(FNU+I-1)/(X), or scaled Bessel functions
          EXP(X)*K/sub(FNU+I-1)/(X), I=1,..,N for real X .GT. 0.0D0 and
          non-negative orders FNU.  If FNU .LT. NULIM, orders FNU and
          FNU+1 are obtained from DBSKNU to start the recursion.  If
          FNU .GE. NULIM, the uniform asymptotic expansion is used for
          orders FNU and FNU+1 to start the recursion.  NULIM is 35 or
          70 depending on whether N=1 or N .GE. 2.  Under and overflow
          tests are made on the leading term of the asymptotic expansion
          before any extensive computation is done.

          The maximum number of significant digits obtainable
          is the smaller of 14 and the number of digits carried in
          double precision arithmetic.

      Description of Arguments

          Input      X,FNU are double precision
            X      - X .GT. 0.0D0
            FNU    - order of the initial K function, FNU .GE. 0.0D0
            KODE   - a parameter to indicate the scaling option
                     KODE=1 returns Y(I)=       K/sub(FNU+I-1)/(X),
                                          I=1,...,N
                     KODE=2 returns Y(I)=EXP(X)*K/sub(FNU+I-1)/(X),
                                          I=1,...,N
            N      - number of members in the sequence, N .GE. 1

          Output     Y is double precision
            Y      - a vector whose first N components contain values
                     for the sequence
                     Y(I)=       k/sub(FNU+I-1)/(X), I=1,...,N  or
                     Y(I)=EXP(X)*K/sub(FNU+I-1)/(X), I=1,...,N
                     depending on KODE
            NZ     - number of components of Y set to zero due to
                     underflow with KODE=1,
                     NZ=0   , normal return, computation completed
                     NZ .NE. 0, first NZ components of Y set to zero
                              due to underflow, Y(I)=0.0D0, I=1,...,NZ

      Error Conditions
```

```
         Improper input arguments - a fatal error
         Overflow - a fatal error
         Underflow with KODE=1 -  a non-fatal error (NZ .NE. 0)

***REFERENCES  F. W. J. Olver, Tables of Bessel Functions of Moderate
                 or Large Orders, NPL Mathematical Tables 6, Her
                 Majesty's Stationery Office, London, 1962.
               N. M. Temme, On the numerical evaluation of the modified
                 Bessel function of the third kind, Journal of
                 Computational Physics 19, (1975), pp. 324-337.
***ROUTINES CALLED  D1MACH, DASYIK, DBESK0, DBESK1, DBSK0E, DBSK1E,
                    DBSKNU, I1MACH, XERMSG
***REVISION HISTORY  (YYMMDD)
   790201  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  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
```

DBESK0

```
      DOUBLE PRECISION FUNCTION DBESK0 (X)
***BEGIN PROLOGUE  DBESK0
***PURPOSE  Compute the modified (hyperbolic) Bessel function of the
            third kind of order zero.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10B1
***TYPE      DOUBLE PRECISION (BESK0-S, DBESK0-D)
***KEYWORDS  FNLIB, HYPERBOLIC BESSEL FUNCTION,
             MODIFIED BESSEL FUNCTION, ORDER ZERO, SPECIAL FUNCTIONS,
             THIRD KIND
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBESK0(X) calculates the double precision modified (hyperbolic)
 Bessel function of the third kind of order zero for double
 precision argument X.  The argument must be greater than zero
 but not so large that the result underflows.

 Series for BK0        on the interval  0.          to  4.00000E+00
                                       with weighted error   3.08E-33
                                        log weighted error  32.51
                                 significant figures required  32.05
                                      decimal places required  33.11

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DBESI0, DBSK0E, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   END PROLOGUE
```

DBESK1

```
      DOUBLE PRECISION FUNCTION DBESK1 (X)
***BEGIN PROLOGUE  DBESK1
***PURPOSE  Compute the modified (hyperbolic) Bessel function of the
            third kind of order one.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10B1
***TYPE      DOUBLE PRECISION (BESK1-S, DBESK1-D)
***KEYWORDS  FNLIB, HYPERBOLIC BESSEL FUNCTION,
             MODIFIED BESSEL FUNCTION, ORDER ONE, SPECIAL FUNCTIONS,
             THIRD KIND
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBESK1(X) calculates the double precision modified (hyperbolic)
 Bessel function of the third kind of order one for double precision
 argument X.  The argument must be large enough that the result does
 not overflow and small enough that the result does not underflow.

 Series for BK1        on the interval  0.          to  4.00000E+00
                                       with weighted error   9.16E-32
                                        log weighted error  31.04
                                 significant figures required  30.61
                                      decimal places required  31.64

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DBESI1, DBSK1E, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   END PROLOGUE
```

DBESKS

```
      SUBROUTINE DBESKS (XNU, X, NIN, BK)
***BEGIN PROLOGUE  DBESKS
***PURPOSE  Compute a sequence of modified Bessel functions of the
            third kind of fractional order.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10B3
***TYPE      DOUBLE PRECISION (BESKS-S, DBESKS-D)
***KEYWORDS  FNLIB, FRACTIONAL ORDER, MODIFIED BESSEL FUNCTION,
             SEQUENCE OF BESSEL FUNCTIONS, SPECIAL FUNCTIONS,
             THIRD KIND
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBESKS computes a sequence of modified Bessel functions of the third
 kind of order XNU + I at X, where X .GT. 0, XNU lies in (-1,1),
 and I = 0, 1, ... , NIN - 1, if NIN is positive and I = 0, 1, ... ,
 NIN + 1, if NIN is negative.  On return, the vector BK(.) contains
 the results at X for order starting at XNU.  XNU, X, and BK are
 double precision.  NIN is an integer.

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DBSKES, XERMSG
***REVISION HISTORY  (YYMMDD)
   770601  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)
   END PROLOGUE
```

DBESY

```
      SUBROUTINE DBESY (X, FNU, N, Y)
***BEGIN PROLOGUE  DBESY
***PURPOSE  Implement forward recursion on the three term recursion
            relation for a sequence of non-negative order Bessel
            functions Y/SUB(FNU+I-1)/(X), I=1,...,N for real, positive
            X and non-negative orders FNU.
***LIBRARY   SLATEC
***CATEGORY  C10A3
***TYPE      DOUBLE PRECISION (BESY-S, DBESY-D)
***KEYWORDS  SPECIAL FUNCTIONS, Y BESSEL FUNCTION
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Abstract  **** a double precision routine ****
          DBESY implements forward recursion on the three term
          recursion relation for a sequence of non-negative order Bessel
          functions Y/sub(FNU+I-1)/(X), I=1,N for real X .GT. 0.0D0 and
          non-negative orders FNU.  If FNU .LT. NULIM, orders FNU and
          FNU+1 are obtained from DBSYNU which computes by a power
          series for X .LE. 2, the K Bessel function of an imaginary
          argument for 2 .LT. X .LE. 20 and the asymptotic expansion for
          X .GT. 20.

          If FNU .GE. NULIM, the uniform asymptotic expansion is coded
          in DASYJY for orders FNU and FNU+1 to start the recursion.
          NULIM is 70 or 100 depending on whether N=1 or N .GE. 2.  An
          overflow test is made on the leading term of the asymptotic
          expansion before any extensive computation is done.

          The maximum number of significant digits obtainable
          is the smaller of 14 and the number of digits carried in
          double precision arithmetic.

      Description of Arguments

          Input
            X      - X .GT. 0.0D0
            FNU    - order of the initial Y function, FNU .GE. 0.0D0
            N      - number of members in the sequence, N .GE. 1

          Output
            Y      - a vector whose first N components contain values
                     for the sequence Y(I)=Y/sub(FNU+I-1)/(X), I=1,N.

      Error Conditions
          Improper input arguments - a fatal error
          Overflow - a fatal error

***REFERENCES  F. W. J. Olver, Tables of Bessel Functions of Moderate
                 or Large Orders, NPL Mathematical Tables 6, Her
                 Majesty's Stationery Office, London, 1962.
               N. M. Temme, On the numerical evaluation of the modified
                 Bessel function of the third kind, Journal of
                 Computational Physics 19, (1975), pp. 324-337.
               N. M. Temme, On the numerical evaluation of the ordinary
                 Bessel function of the second kind, Journal of
```

```
                 Computational Physics 21, (1976), pp. 343-350.
***ROUTINES CALLED  D1MACH, DASYJY, DBESY0, DBESY1, DBSYNU, DYAIRY,
                    I1MACH, XERMSG
***REVISION HISTORY  (YYMMDD)
   800501  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  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
```

DBESY0

```
      DOUBLE PRECISION FUNCTION DBESY0 (X)
***BEGIN PROLOGUE  DBESY0
***PURPOSE  Compute the Bessel function of the second kind of order
            zero.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10A1
***TYPE      DOUBLE PRECISION (BESY0-S, DBESY0-D)
***KEYWORDS  BESSEL FUNCTION, FNLIB, ORDER ZERO, SECOND KIND,
             SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBESY0(X) calculates the double precision Bessel function of the
 second kind of order zero for double precision argument X.

 Series for BY0        on the interval  0.          to  1.60000E+01
                                       with weighted error   8.14E-32
                                        log weighted error  31.09
                                 significant figures required  30.31
                                      decimal places required  31.73

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, D9B0MP, DBESJ0, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   END PROLOGUE
```

DBESY1

```
      DOUBLE PRECISION FUNCTION DBESY1 (X)
***BEGIN PROLOGUE  DBESY1
***PURPOSE  Compute the Bessel function of the second kind of order
            one.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10A1
***TYPE      DOUBLE PRECISION (BESY1-S, DBESY1-D)
***KEYWORDS  BESSEL FUNCTION, FNLIB, ORDER ONE, SECOND KIND,
             SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBESY1(X) calculates the double precision Bessel function of the
 second kind of order for double precision argument X.

 Series for BY1        on the interval  0.          to  1.60000E+01
                                       with weighted error   8.65E-33
                                        log weighted error  32.06
                                 significant figures required  32.17
                                      decimal places required  32.71

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, D9B1MP, DBESJ1, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   END PROLOGUE
```

DBETA

```
      DOUBLE PRECISION FUNCTION DBETA (A, B)
***BEGIN PROLOGUE  DBETA
***PURPOSE  Compute the complete Beta function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C7B
***TYPE      DOUBLE PRECISION (BETA-S, DBETA-D, CBETA-C)
***KEYWORDS  COMPLETE BETA FUNCTION, FNLIB, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBETA(A,B) calculates the double precision complete beta function
 for double precision arguments A and B.

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DGAMLM, DGAMMA, DLBETA, XERMSG
***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)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900727  Added EXTERNAL statement.  (WRB)
   END PROLOGUE
```

DBETAI

```
      DOUBLE PRECISION FUNCTION DBETAI (X, PIN, QIN)
***BEGIN PROLOGUE  DBETAI
***PURPOSE  Calculate the incomplete Beta function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C7F
***TYPE      DOUBLE PRECISION (BETAI-S, DBETAI-D)
***KEYWORDS  FNLIB, INCOMPLETE BETA FUNCTION, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

   DBETAI calculates the DOUBLE PRECISION incomplete beta function.

   The incomplete beta function ratio is the probability that a
   random variable from a beta distribution having parameters PIN and
   QIN will be less than or equal to X.

     -- Input Arguments -- All arguments are DOUBLE PRECISION.
   X      upper limit of integration.  X must be in (0,1) inclusive.
   PIN    first beta distribution parameter.  PIN must be .GT. 0.0.
   QIN    second beta distribution parameter.  QIN must be .GT. 0.0.

***REFERENCES  Nancy E. Bosten and E. L. Battiste, Remark on Algorithm
                 179, Communications of the ACM 17, 3 (March 1974),
                 pp. 156.
***ROUTINES CALLED  D1MACH, DLBETA, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  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)
   920528  DESCRIPTION and REFERENCES sections revised.  (WRB)
   END PROLOGUE
```

DBFQAD

```
      SUBROUTINE DBFQAD (F, T, BCOEF, N, K, ID, X1, X2, TOL, QUAD, IERR,
     +   WORK)
***BEGIN PROLOGUE  DBFQAD
***PURPOSE  Compute the integral of a product of a function and a
            derivative of a K-th order B-spline.
***LIBRARY   SLATEC
***CATEGORY  H2A2A1, E3, K6
***TYPE      DOUBLE PRECISION (BFQAD-S, DBFQAD-D)
***KEYWORDS  INTEGRAL OF B-SPLINE, QUADRATURE
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Abstract    **** a double precision routine ****

          DBFQAD computes the integral on (X1,X2) of a product of a
          function F and the ID-th derivative of a K-th order B-spline,
          using the B-representation (T,BCOEF,N,K).  (X1,X2) must be a
          subinterval of T(K) .LE. X .LE. T(N+1).  An integration rou-
          tine, DBSGQ8 (a modification of GAUS8), integrates the product
          on subintervals of (X1,X2) formed by included (distinct) knots

          The maximum number of significant digits obtainable in
          DBSQAD is the smaller of 18 and the number of digits
          carried in double precision arithmetic.

      Description of Arguments
          Input      F,T,BCOEF,X1,X2,TOL are double precision
            F      - external function of one argument for the
                     integrand BF(X)=F(X)*DBVALU(T,BCOEF,N,K,ID,X,INBV,
                     WORK)
            T      - knot array of length N+K
            BCOEF  - coefficient array of length N
            N      - length of coefficient array
            K      - order of B-spline, K .GE. 1
            ID     - order of the spline derivative, 0 .LE. ID .LE. K-1
                     ID=0 gives the spline function
            X1,X2  - end points of quadrature interval in
                     T(K) .LE. X .LE. T(N+1)
            TOL    - desired accuracy for the quadrature, suggest
                     10.*DTOL .LT. TOL .LE. .1 where DTOL is the maximum
                     of 1.0D-18 and double precision unit roundoff for
                     the machine = D1MACH(4)

          Output     QUAD,WORK are double precision
            QUAD   - integral of BF(X) on (X1,X2)
            IERR   - a status code
                     IERR=1  normal return
                          2  some quadrature on (X1,X2) does not meet
                             the requested tolerance.
            WORK   - work vector of length 3*K

      Error Conditions
          Improper input is a fatal error
          Some quadrature fails to meet the requested tolerance

***REFERENCES  D. E. Amos, Quadrature subroutines for splines and
```

```
                 B-splines, Report SAND79-1825, Sandia Laboratories,
                 December 1979.
***ROUTINES CALLED  D1MACH, DBSGQ8, DINTRV, XERMSG
***REVISION HISTORY  (YYMMDD)
   800901  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)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DBHIN

```
      SUBROUTINE DBHIN (N, NELT, IA, JA, A, ISYM, SOLN, RHS, IUNIT, JOB)
***BEGIN PROLOGUE  DBHIN
***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      DOUBLE 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
     DOUBLE PRECISION A(NELT), SOLN(N), RHS(N)

     CALL DBHIN( 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      Double Precision 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      Double Precision 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      Double Precision 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
```

DBI

```
      DOUBLE PRECISION FUNCTION DBI (X)
***BEGIN PROLOGUE  DBI
***PURPOSE  Evaluate the Bairy function (the Airy function of the
            second kind).
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10D
***TYPE      DOUBLE PRECISION (BI-S, DBI-D)
***KEYWORDS  BAIRY FUNCTION, FNLIB, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBI(X) calculates the double precision Airy function of the
 second kind for double precision argument X.

 Series for BIF        on the interval -1.00000E+00 to  1.00000E+00
                                        with weighted error   1.45E-32
                                         log weighted error  31.84
                                significant figures required  30.85
                                     decimal places required  32.40

 Series for BIG        on the interval -1.00000E+00 to  1.00000E+00
                                        with weighted error   1.29E-33
                                         log weighted error  32.89
                                significant figures required  31.48
                                     decimal places required  33.45

 Series for BIF2       on the interval  1.00000E+00 to  8.00000E+00
                                        with weighted error   6.08E-32
                                         log weighted error  31.22
                         approx significant figures required  30.8
                                     decimal places required  31.80

 Series for BIG2       on the interval  1.00000E+00 to  8.00000E+00
                                        with weighted error   4.91E-33
                                         log weighted error  32.31
                         approx significant figures required  31.6
                                     decimal places required  32.90

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, D9AIMP, DBIE, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   END PROLOGUE
```

DBIE

```
      DOUBLE PRECISION FUNCTION DBIE (X)
***BEGIN PROLOGUE  DBIE
***PURPOSE  Calculate the Bairy function for a negative argument and an
            exponentially scaled Bairy function for a non-negative
            argument.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10D
***TYPE      DOUBLE PRECISION (BIE-S, DBIE-D)
***KEYWORDS  BAIRY FUNCTION, EXPONENTIALLY SCALED, FNLIB,
             SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBIE(X) calculates the double precision Airy function of the
 second kind or the double precision exponentially scaled Airy
 function of the second kind, depending on the value of the
 double precision argument X.

 Evaluate BI(X) for X .LE. 0.0  and  BI(X)*EXP(-ZETA)  where
 ZETA = 2/3 * X**(3/2)  for X .GE. 0.0


 Series for BIF        on the interval -1.00000E+00 to  1.00000E+00
                                        with weighted error   1.45E-32
                                         log weighted error  31.84
                                significant figures required  30.85
                                     decimal places required  32.40


 Series for BIG        on the interval -1.00000E+00 to  1.00000E+00
                                        with weighted error   1.29E-33
                                         log weighted error  32.89
                                significant figures required  31.48
                                     decimal places required  33.45


 Series for BIF2       on the interval  1.00000E+00 to  8.00000E+00
                                        with weighted error   6.08E-32
                                         log weighted error  31.22
                         approx significant figures required  30.8
                                     decimal places required  31.80


 Series for BIG2       on the interval  1.00000E+00 to  8.00000E+00
                                        with weighted error   4.91E-33
                                         log weighted error  32.31
                         approx significant figures required  31.6
                                     decimal places required  32.90


 Series for BIP1       on the interval  1.25000E-01 to  3.53553E-01
                                        with weighted error   1.06E-32
                                         log weighted error  31.98
                                significant figures required  30.61
                                     decimal places required  32.81
```

```
 Series for BIP2       on the interval  0.          to  1.25000E-01
                                      with weighted error   4.04E-33
                                       log weighted error  32.39
                               significant figures required  31.15
                                    decimal places required  33.37

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, D9AIMP, DCSEVL, INITDS
***REVISION HISTORY  (YYMMDD)
   770701  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
```

DBINOM

```
      DOUBLE PRECISION FUNCTION DBINOM (N, M)
***BEGIN PROLOGUE  DBINOM
***PURPOSE  Compute the binomial coefficients.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C1
***TYPE      DOUBLE PRECISION (BINOM-S, DBINOM-D)
***KEYWORDS  BINOMIAL COEFFICIENTS, FNLIB, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBINOM(N,M) calculates the double precision binomial coefficient
 for integer arguments N and M.  The result is (N!)/((M!)(N-M)!).

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, D9LGMC, DLNREL, XERMSG
***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)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   END PROLOGUE
```

DBINT4

```
      SUBROUTINE DBINT4 (X, Y, NDATA, IBCL, IBCR, FBCL, FBCR, KNTOPT, T,
     +   BCOEF, N, K, W)
***BEGIN PROLOGUE  DBINT4
***PURPOSE  Compute the B-representation of a cubic spline
            which interpolates given data.
***LIBRARY   SLATEC
***CATEGORY  E1A
***TYPE      DOUBLE PRECISION (BINT4-S, DBINT4-D)
***KEYWORDS  B-SPLINE, CUBIC SPLINES, DATA FITTING, INTERPOLATION
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Abstract    **** a double precision routine ****

          DBINT4 computes the B representation (T,BCOEF,N,K) of a
          cubic spline (K=4) which interpolates data (X(I),Y(I)),
          I=1,NDATA.  Parameters IBCL, IBCR, FBCL, FBCR allow the
          specification of the spline first or second derivative at
          both X(1) and X(NDATA).  When this data is not specified
          by the problem, it is common practice to use a natural
          spline by setting second derivatives at X(1) and X(NDATA)
          to zero (IBCL=IBCR=2,FBCL=FBCR=0.0).  The spline is defined
          on T(4) .LE. X .LE. T(N+1) with (ordered) interior knots at
          X(I) values where N=NDATA+2.  The knots T(1),T(2),T(3) lie to
          the left of T(4)=X(1) and the knots T(N+2), T(N+3), T(N+4)
          lie to the right of T(N+1)=X(NDATA) in increasing order.  If
          no extrapolation outside (X(1),X(NDATA)) is anticipated, the
          knots T(1)=T(2)=T(3)=T(4)=X(1) and T(N+2)=T(N+3)=T(N+4)=
          T(N+1)=X(NDATA) can be specified by KNTOPT=1.  KNTOPT=2
          selects a knot placement for T(1), T(2), T(3) to make the
          first 7 knots symmetric about T(4)=X(1) and similarly for
          T(N+2), T(N+3), T(N+4) about T(N+1)=X(NDATA).  KNTOPT=3
          allows the user to make his own selection, in increasing
          order, for T(1), T(2), T(3) to the left of X(1) and T(N+2),
          T(N+3), T(N+4) to the right of X(NDATA) in the work array
          W(1) through W(6).  In any case, the interpolation on
          T(4) .LE. X .LE. T(N+1) by using function DBVALU is unique
          for given boundary conditions.

      Description of Arguments

          Input      X,Y,FBCL,FBCR,W are double precision
            X      - X vector of abscissae of length NDATA, distinct
                     and in increasing order
            Y      - Y vector of ordinates of length NDATA
            NDATA  - number of data points, NDATA .GE. 2
            IBCL   - selection parameter for left boundary condition
                     IBCL = 1 constrain the first derivative at
                              X(1) to FBCL
                          = 2 constrain the second derivative at
                              X(1) to FBCL
            IBCR   - selection parameter for right boundary condition
                     IBCR = 1 constrain first derivative at
                              X(NDATA) to FBCR
                     IBCR = 2 constrain second derivative at
                              X(NDATA) to FBCR
```

```
            FBCL   - left boundary values governed by IBCL
            FBCR   - right boundary values governed by IBCR
            KNTOPT - knot selection parameter
                     KNTOPT = 1 sets knot multiplicity at T(4) and
                                T(N+1) to 4
                            = 2 sets a symmetric placement of knots
                                about T(4) and T(N+1)
                            = 3 sets T(I)=W(I) and T(N+1+I)=W(3+I),I=1,3
                                where W(I),I=1,6 is supplied by the user
            W      - work array of dimension at least 5*(NDATA+2)
                     If KNTOPT=3, then W(1),W(2),W(3) are knot values to
                     the left of X(1) and W(4),W(5),W(6) are knot
                     values to the right of X(NDATA) in increasing
                     order to be supplied by the user

          Output     T,BCOEF are double precision
            T      - knot array of length N+4
            BCOEF  - B spline coefficient array of length N
            N      - number of coefficients, N=NDATA+2
            K      - order of spline, K=4

      Error Conditions
          Improper  input is a fatal error
          Singular system of equations is a fatal error

***REFERENCES  D. E. Amos, Computation with splines and B-splines,
                 Report SAND78-1968, Sandia Laboratories, March 1979.
               Carl de Boor, Package for calculating with B-splines,
                 SIAM Journal on Numerical Analysis 14, 3 (June 1977),
                 pp. 441-472.
               Carl de Boor, A Practical Guide to Splines, Applied
                 Mathematics Series 27, Springer-Verlag, New York,
                 1978.
***ROUTINES CALLED  D1MACH, DBNFAC, DBNSLV, DBSPVD, XERMSG
***REVISION HISTORY  (YYMMDD)
   800901  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)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DBINTK

```
      SUBROUTINE DBINTK (X, Y, T, N, K, BCOEF, Q, WORK)
***BEGIN PROLOGUE  DBINTK
***PURPOSE  Compute the B-representation of a spline which interpolates
            given data.
***LIBRARY   SLATEC
***CATEGORY  E1A
***TYPE      DOUBLE PRECISION (BINTK-S, DBINTK-D)
***KEYWORDS  B-SPLINE, DATA FITTING, INTERPOLATION
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Written by Carl de Boor and modified by D. E. Amos

      Abstract    **** a double precision routine ****

          DBINTK is the SPLINT routine of the reference.

          DBINTK produces the B-spline coefficients, BCOEF, of the
          B-spline of order K with knots T(I), I=1,...,N+K, which
          takes on the value Y(I) at X(I), I=1,...,N.  The spline or
          any of its derivatives can be evaluated by calls to DBVALU.

          The I-th equation of the linear system A*BCOEF = B for the
          coefficients of the interpolant enforces interpolation at
          X(I), I=1,...,N.  Hence, B(I) = Y(I), for all I, and A is
          a band matrix with 2K-1 bands if A is invertible.  The matrix
          A is generated row by row and stored, diagonal by diagonal,
          in the rows of Q, with the main diagonal going into row K.
          The banded system is then solved by a call to DBNFAC (which
          constructs the triangular factorization for A and stores it
          again in Q), followed by a call to DBNSLV (which then
          obtains the solution BCOEF by substitution).  DBNFAC does no
          pivoting, since the total positivity of the matrix A makes
          this unnecessary.  The linear system to be solved is
          (theoretically) invertible if and only if
                  T(I) .LT. X(I) .LT. T(I+K),        for all I.
          Equality is permitted on the left for I=1 and on the right
          for I=N when K knots are used at X(1) or X(N).  Otherwise,
          violation of this condition is certain to lead to an error.

      Description of Arguments

          Input       X,Y,T are double precision
            X       - vector of length N containing data point abscissa
                      in strictly increasing order.
            Y       - corresponding vector of length N containing data
                      point ordinates.
            T       - knot vector of length N+K
                      Since T(1),..,T(K) .LE. X(1) and T(N+1),..,T(N+K)
                      .GE. X(N), this leaves only N-K knots (not nec-
                      essarily X(I) values) interior to (X(1),X(N))
            N       - number of data points, N .GE. K
            K       - order of the spline, K .GE. 1

          Output      BCOEF,Q,WORK are double precision
            BCOEF   - a vector of length N containing the B-spline
```

```
                    coefficients
          Q       - a work vector of length (2*K-1)*N, containing
                    the triangular factorization of the coefficient
                    matrix of the linear system being solved.  The
                    coefficients for the interpolant of an
                    additional data set (X(I),YY(I)), I=1,...,N
                    with the same abscissa can be obtained by loading
                    YY into BCOEF and then executing
                        CALL DBNSLV (Q,2K-1,N,K-1,K-1,BCOEF)
          WORK    - work vector of length 2*K

     Error Conditions
         Improper input is a fatal error
         Singular system of equations is a fatal error

***REFERENCES  D. E. Amos, Computation with splines and B-splines,
                 Report SAND78-1968, Sandia Laboratories, March 1979.
               Carl de Boor, Package for calculating with B-splines,
                 SIAM Journal on Numerical Analysis 14, 3 (June 1977),
                 pp. 441-472.
               Carl de Boor, A Practical Guide to Splines, Applied
                 Mathematics Series 27, Springer-Verlag, New York,
                 1978.
***ROUTINES CALLED  DBNFAC, DBNSLV, DBSPVN, XERMSG
***REVISION HISTORY  (YYMMDD)
   800901  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)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DBNDAC

```
      SUBROUTINE DBNDAC (G, MDG, NB, IP, IR, MT, JT)
***BEGIN PROLOGUE  DBNDAC
***PURPOSE  Compute the LU factorization of a  banded matrices using
            sequential accumulation of rows of the data matrix.
            Exactly one right-hand side vector is permitted.
***LIBRARY   SLATEC
***CATEGORY  D9
***TYPE      DOUBLE PRECISION (BNDACC-S, DBNDAC-D)
***KEYWORDS  BANDED MATRIX, CURVE FITTING, LEAST SQUARES
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
***DESCRIPTION

     These subroutines solve the least squares problem Ax = b for
     banded matrices A using sequential accumulation of rows of the
     data matrix.  Exactly one right-hand side vector is permitted.

     These subroutines are intended for the type of least squares
     systems that arise in applications such as curve or surface
     fitting of data.  The least squares equations are accumulated and
     processed using only part of the data.  This requires a certain
     user interaction during the solution of Ax = b.

     Specifically, suppose the data matrix (A B) is row partitioned
     into Q submatrices.  Let (E F) be the T-th one of these
     submatrices where E = (0 C 0).  Here the dimension of E is MT by N
     and the dimension of C is MT by NB.  The value of NB is the
     bandwidth of A.  The dimensions of the leading block of zeros in E
     are MT by JT-1.

     The user of the subroutine DBNDAC provides MT,JT,C and F for
     T=1,...,Q.  Not all of this data must be supplied at once.

     Following the processing of the various blocks (E F), the matrix
     (A B) has been transformed to the form (R D) where R is upper
     triangular and banded with bandwidth NB.  The least squares
     system Rx = d is then easily solved using back substitution by
     executing the statement CALL DBNDSL(1,...). The sequence of
     values for JT must be nondecreasing.  This may require some
     preliminary interchanges of rows and columns of the matrix A.

     The primary reason for these subroutines is that the total
     processing can take place in a working array of dimension MU by
     NB+1.  An acceptable value for MU is

                        MU = MAX(MT + N + 1),

     where N is the number of unknowns.

     Here the maximum is taken over all values of MT for T=1,...,Q.
     Notice that MT can be taken to be a small as one, showing that
     MU can be as small as N+2.  The subprogram DBNDAC processes the
     rows more efficiently if MU is large enough so that each new
     block (C F) has a distinct value of JT.

     The four principle parts of these algorithms are obtained by the
```

following call statements

CALL DBNDAC(...) Introduce new blocks of data.

CALL DBNDSL(1,...)Compute solution vector and length of residual vector.

CALL DBNDSL(2,...)Given any row vector H solve YR = H for the row vector Y.

CALL DBNDSL(3,...)Given any column vector W solve RZ = W for the column vector Z.

The dots in the above call statements indicate additional arguments that will be specified in the following paragraphs.

The user must dimension the array appearing in the call list.. G(MDG,NB+1)

Description of calling sequence for DBNDAC..

The entire set of parameters for DBNDAC are

Input.. All Type REAL variables are DOUBLE PRECISION

G(*,*) The working array into which the user will place the MT by NB+1 block (C F) in rows IR through IR+MT-1, columns 1 through NB+1. See descriptions of IR and MT below.

MDG The number of rows in the working array G(*,*). The value of MDG should be .GE. MU. The value of MU is defined in the abstract of these subprograms.

NB The bandwidth of the data matrix A.

IP Set by the user to the value 1 before the first call to DBNDAC. Its subsequent value is controlled by DBNDAC to set up for the next call to DBNDAC.

IR Index of the row of G(*,*) where the user is to place the new block of data (C F). Set by the user to the value 1 before the first call to DBNDAC. Its subsequent value is controlled by DBNDAC. A value of IR .GT. MDG is considered an error.

MT,JT Set by the user to indicate respectively the number of new rows of data in the block and the index of the first nonzero column in that set of rows (E F) = (0 C 0 F) being processed.

Output.. All Type REAL variables are DOUBLE PRECISION

G(*,*) The working array which will contain the processed rows of that part of the data matrix which has been passed to DBNDAC.

IP,IR	The values of these arguments are advanced by DBNDAC to be ready for storing and processing a new block of data in G(*,*).

Description of calling sequence for DBNDSL..

The user must dimension the arrays appearing in the call list..

G(MDG,NB+1), X(N)

The entire set of parameters for DBNDSL are

Input.. All Type REAL variables are DOUBLE PRECISION

MODE	Set by the user to one of the values 1, 2, or 3. These values respectively indicate that the solution of AX = B, YR = H or RZ = W is required.
G(*,*),MDG, NB,IP,IR	These arguments all have the same meaning and contents as following the last call to DBNDAC.
X(*)	With mode=2 or 3 this array contains, respectively, the right-side vectors H or W of the systems YR = H or RZ = W.
N	The number of variables in the solution vector. If any of the N diagonal terms are zero the subroutine DBNDSL prints an appropriate message. This condition is considered an error.

Output.. All Type REAL variables are DOUBLE PRECISION

X(*)	This array contains the solution vectors X, Y or Z of the systems AX = B, YR = H or RZ = W depending on the value of MODE=1, 2 or 3.
RNORM	If MODE=1 RNORM is the Euclidean length of the residual vector AX-B. When MODE=2 or 3 RNORM is set to zero.

Remarks..

To obtain the upper triangular matrix and transformed right-hand side vector D so that the super diagonals of R form the columns of G(*,*), execute the following Fortran statements.

```
      NBP1=NB+1

      DO 10 J=1, NBP1

   10 G(IR,J) = 0.E0

      MT=1

      JT=N+1

      CALL DBNDAC(G,MDG,NB,IP,IR,MT,JT)
```

```
***REFERENCES  C. L. Lawson and R. J. Hanson, Solving Least Squares
                 Problems, Prentice-Hall, Inc., 1974, Chapter 27.
***ROUTINES CALLED  DH12, XERMSG
***REVISION HISTORY  (YYMMDD)
   790101  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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
```

DBNDSL

```
      SUBROUTINE DBNDSL (MODE, G, MDG, NB, IP, IR, X, N, RNORM)
***BEGIN PROLOGUE  DBNDSL
***PURPOSE  Solve the least squares problem for a banded matrix using
            sequential accumulation of rows of the data matrix.
            Exactly one right-hand side vector is permitted.
***LIBRARY   SLATEC
***CATEGORY  D9
***TYPE      DOUBLE PRECISION (BNDSOL-S, DBNDSL-D)
***KEYWORDS  BANDED MATRIX, CURVE FITTING, LEAST SQUARES
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
***DESCRIPTION
```

These subroutines solve the least squares problem Ax = b for banded matrices A using sequential accumulation of rows of the data matrix. Exactly one right-hand side vector is permitted.

These subroutines are intended for the type of least squares systems that arise in applications such as curve or surface fitting of data. The least squares equations are accumulated and processed using only part of the data. This requires a certain user interaction during the solution of Ax = b.

Specifically, suppose the data matrix (A B) is row partitioned into Q submatrices. Let (E F) be the T-th one of these submatrices where E = (0 C 0). Here the dimension of E is MT by N and the dimension of C is MT by NB. The value of NB is the bandwidth of A. The dimensions of the leading block of zeros in E are MT by JT-1.

The user of the subroutine DBNDAC provides MT,JT,C and F for T=1,...,Q. Not all of this data must be supplied at once.

Following the processing of the various blocks (E F), the matrix (A B) has been transformed to the form (R D) where R is upper triangular and banded with bandwidth NB. The least squares system Rx = d is then easily solved using back substitution by executing the statement CALL DBNDSL(1,...). The sequence of values for JT must be nondecreasing. This may require some preliminary interchanges of rows and columns of the matrix A.

The primary reason for these subroutines is that the total processing can take place in a working array of dimension MU by NB+1. An acceptable value for MU is

$$MU = MAX(MT + N + 1),$$

where N is the number of unknowns.

Here the maximum is taken over all values of MT for T=1,...,Q. Notice that MT can be taken to be a small as one, showing that MU can be as small as N+2. The subprogram DBNDAC processes the rows more efficiently if MU is large enough so that each new block (C F) has a distinct value of JT.

The four principle parts of these algorithms are obtained by the

following call statements

CALL DBNDAC(...) Introduce new blocks of data.

CALL DBNDSL(1,...)Compute solution vector and length of residual vector.

CALL DBNDSL(2,...)Given any row vector H solve YR = H for the row vector Y.

CALL DBNDSL(3,...)Given any column vector W solve RZ = W for the column vector Z.

The dots in the above call statements indicate additional arguments that will be specified in the following paragraphs.

The user must dimension the array appearing in the call list.. G(MDG,NB+1)

Description of calling sequence for DBNDAC..

The entire set of parameters for DBNDAC are

Input.. All Type REAL variables are DOUBLE PRECISION

G(*,*) The working array into which the user will place the MT by NB+1 block (C F) in rows IR through IR+MT-1, columns 1 through NB+1. See descriptions of IR and MT below.

MDG The number of rows in the working array G(*,*). The value of MDG should be .GE. MU. The value of MU is defined in the abstract of these subprograms.

NB The bandwidth of the data matrix A.

IP Set by the user to the value 1 before the first call to DBNDAC. Its subsequent value is controlled by DBNDAC to set up for the next call to DBNDAC.

IR Index of the row of G(*,*) where the user is the user to the value 1 before the first call to DBNDAC. Its subsequent value is controlled by DBNDAC. A value of IR .GT. MDG is considered an error.

MT,JT Set by the user to indicate respectively the number of new rows of data in the block and the index of the first nonzero column in that set of rows (E F) = (0 C 0 F) being processed.

Output.. All Type REAL variables are DOUBLE PRECISION

G(*,*) The working array which will contain the processed rows of that part of the data matrix which has been passed to DBNDAC.

IP,IR The values of these arguments are advanced by DBNDAC to be ready for storing and processing

a new block of data in G(*,*).

Description of calling sequence for DBNDSL..

The user must dimension the arrays appearing in the call list..

G(MDG,NB+1), X(N)

The entire set of parameters for DBNDSL are

Input..

MODE	Set by the user to one of the values 1, 2, or 3. These values respectively indicate that the solution of AX = B, YR = H or RZ = W is required.
G(*,*),MDG, NB,IP,IR	These arguments all have the same meaning and contents as following the last call to DBNDAC.
X(*)	With mode=2 or 3 this array contains, respectively, the right-side vectors H or W of the systems YR = H or RZ = W.
N	The number of variables in the solution vector. If any of the N diagonal terms are zero the subroutine DBNDSL prints an appropriate message. This condition is considered an error.

Output..

X(*)	This array contains the solution vectors X, Y or Z of the systems AX = B, YR = H or RZ = W depending on the value of MODE=1, 2 or 3.
RNORM	If MODE=1 RNORM is the Euclidean length of the residual vector AX-B. When MODE=2 or 3 RNORM is set to zero.

Remarks..

To obtain the upper triangular matrix and transformed right-hand side vector D so that the super diagonals of R form the columns of G(*,*), execute the following Fortran statements.

```
      NBP1=NB+1

      DO 10 J=1, NBP1

   10 G(IR,J) = 0.E0

      MT=1

      JT=N+1

      CALL DBNDAC(G,MDG,NB,IP,IR,MT,JT)
```

***REFERENCES C. L. Lawson and R. J. Hanson, Solving Least Squares

```
                 Problems, Prentice-Hall, Inc., 1974, Chapter 27.
***ROUTINES CALLED  XERMSG
***REVISION HISTORY  (YYMMDD)
   790101  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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
```

DBOCLS

```
      SUBROUTINE DBOCLS (W, MDW, MCON, MROWS, NCOLS, BL, BU, IND, IOPT,
     +   X, RNORMC, RNORM, MODE, RW, IW)
***BEGIN PROLOGUE  DBOCLS
***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      DOUBLE PRECISION (SBOCLS-S, DBOCLS-D)
***KEYWORDS  BOUNDS, CONSTRAINTS, INEQUALITY, LEAST SQUARES, LINEAR
***AUTHOR  Hanson, R. J., (SNLA)
***DESCRIPTION

   **** All INPUT and OUTPUT real variables are DOUBLE PRECISION ****

      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 DBOCLS( ). 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 DBOCLS(), 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 DBOCLS( ) 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 DBOCLS( )
 . 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 DBOCLS( ) 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 DBOCLS( )
```

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 DBOCLS( )
```

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 DBOCLS( )
```

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 DBOCLS( )
```

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 DBOCLS( )
```

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 DBOCLS( )
```

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 DBOCLS( ) 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 DBOCLS( )
```

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 DBOCLS( ). 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 DBOCLS( )
```

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 DBOCLS( )
```

Error Messages for DBOCLS()
----- -------- --- --------

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

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

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

WARNING IN...
```

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

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

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

 WARNING IN...
 DBOCLS(). 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...
 DBOCLS(). 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  D1MACH, DASUM, DBOLS, DCOPY, DDOT, DNRM2, DSCAL,
                    XERMSG
***REVISION HISTORY  (YYMMDD)
   821220  DATE WRITTEN
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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 DBOLS.  (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DBOLS

```
      SUBROUTINE DBOLS (W, MDW, MROWS, NCOLS, BL, BU, IND, IOPT, X,
     +   RNORM, MODE, RW, IW)
***BEGIN PROLOGUE  DBOLS
***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      DOUBLE PRECISION (SBOLS-S, DBOLS-D)
***KEYWORDS  BOUNDS, CONSTRAINTS, INEQUALITY, LEAST SQUARES, LINEAR
***AUTHOR  Hanson, R. J., (SNLA)
***DESCRIPTION

   **** All INPUT and OUTPUT real variables are DOUBLE PRECISION ****

     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 DBOLSM( ). 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 DBOLSM( ).
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 DBOLS( ). 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 DBOLSM(). See the section ERROR MESSAGES for DBOLSM() in the documentation for DBOLSM().

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 DBOLS( ). 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 DBOLS( ) 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 DBOLS()
. 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 DBOLS() 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 DBOLS()
```

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 DBOLS()
```

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 DBOLS()
```

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 DBOLSM(). If LP is the processing pointer for IOPT(*),

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

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 DBOLS()
```

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 DBOLS()
   ----- -------- --- -------

WARNING IN...
DBOLS(). 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...
DBOLS(). 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...
DBOLS(). 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...
DBOLS(). 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...
DBOLS(). 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...
DBOLS(). 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...
DBOLS(). 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...
DBOLS().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...
 DBOLS(). 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...
 DBOLS().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...
 DBOLS(). 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...
 DBOLS(). 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...
 DBOLS() 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...
 DBOLS() 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  DBOLSM, DCOPY, DNRM2, DROT, DROTG, IDAMAX, XERMSG
***REVISION HISTORY  (YYMMDD)
   821220  DATE WRITTEN
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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
```

DBSI0E

```
      DOUBLE PRECISION FUNCTION DBSI0E (X)
***BEGIN PROLOGUE  DBSI0E
***PURPOSE  Compute the exponentially scaled modified (hyperbolic)
            Bessel function of the first kind of order zero.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10B1
***TYPE      DOUBLE PRECISION (BESI0E-S, DBSI0E-D)
***KEYWORDS  EXPONENTIALLY SCALED, FIRST KIND, FNLIB,
             HYPERBOLIC BESSEL FUNCTION, MODIFIED BESSEL FUNCTION,
             ORDER ZERO, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBSI0E(X) calculates the double precision exponentially scaled
 modified (hyperbolic) Bessel function of the first kind of order
 zero for double precision argument X.  The result is the Bessel
 function I0(X) multiplied by EXP(-ABS(X)).

 Series for BI0        on the interval  0.          to  9.00000E+00
                                        with weighted error   9.51E-34
                                         log weighted error  33.02
                                 significant figures required  33.31
                                      decimal places required  33.65

 Series for AI0        on the interval  1.25000E-01 to  3.33333E-01
                                        with weighted error   2.74E-32
                                         log weighted error  31.56
                                 significant figures required  30.15
                                      decimal places required  32.39

 Series for AI02       on the interval  0.          to  1.25000E-01
                                        with weighted error   1.97E-32
                                         log weighted error  31.71
                                 significant figures required  30.15
                                      decimal places required  32.63

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DCSEVL, INITDS
***REVISION HISTORY  (YYMMDD)
   770701  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
```

DBSI1E

```
      DOUBLE PRECISION FUNCTION DBSI1E (X)
***BEGIN PROLOGUE  DBSI1E
***PURPOSE  Compute the exponentially scaled modified (hyperbolic)
            Bessel function of the first kind of order one.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10B1
***TYPE      DOUBLE PRECISION (BESI1E-S, DBSI1E-D)
***KEYWORDS  EXPONENTIALLY SCALED, FIRST KIND, FNLIB,
             HYPERBOLIC BESSEL FUNCTION, MODIFIED BESSEL FUNCTION,
             ORDER ONE, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBSI1E(X) calculates the double precision exponentially scaled
 modified (hyperbolic) Bessel function of the first kind of order
 one for double precision argument X.  The result is I1(X)
 multiplied by EXP(-ABS(X)).

 Series for BI1        on the interval  0.          to  9.00000E+00
                                        with weighted error   1.44E-32
                                         log weighted error  31.84
                                 significant figures required  31.45
                                      decimal places required  32.46

 Series for AI1        on the interval  1.25000E-01 to  3.33333E-01
                                        with weighted error   2.81E-32
                                         log weighted error  31.55
                                 significant figures required  29.93
                                      decimal places required  32.38

 Series for AI12       on the interval  0.          to  1.25000E-01
                                        with weighted error   1.83E-32
                                         log weighted error  31.74
                                 significant figures required  29.97
                                      decimal places required  32.66

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   END PROLOGUE
```

DBSK0E

```
      DOUBLE PRECISION FUNCTION DBSK0E (X)
***BEGIN PROLOGUE  DBSK0E
***PURPOSE  Compute the exponentially scaled modified (hyperbolic)
            Bessel function of the third kind of order zero.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10B1
***TYPE      DOUBLE PRECISION (BESK0E-S, DBSK0E-D)
***KEYWORDS  EXPONENTIALLY SCALED, FNLIB, HYPERBOLIC BESSEL FUNCTION,
             MODIFIED BESSEL FUNCTION, ORDER ZERO, SPECIAL FUNCTIONS,
             THIRD KIND
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBSK0E(X) computes the double precision exponentially scaled
 modified (hyperbolic) Bessel function of the third kind of
 order zero for positive double precision argument X.

 Series for BK0        on the interval  0.          to  4.00000E+00
                                        with weighted error   3.08E-33
                                         log weighted error  32.51
                                 significant figures required  32.05
                                      decimal places required  33.11

 Series for AK0        on the interval  1.25000E-01 to  5.00000E-01
                                        with weighted error   2.85E-32
                                         log weighted error  31.54
                                 significant figures required  30.19
                                      decimal places required  32.33

 Series for AK02       on the interval  0.          to  1.25000E-01
                                        with weighted error   2.30E-32
                                         log weighted error  31.64
                                 significant figures required  29.68
                                      decimal places required  32.40

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DBESI0, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   END PROLOGUE
```

DBSK1E

```
      DOUBLE PRECISION FUNCTION DBSK1E (X)
***BEGIN PROLOGUE  DBSK1E
***PURPOSE  Compute the exponentially scaled modified (hyperbolic)
            Bessel function of the third kind of order one.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10B1
***TYPE      DOUBLE PRECISION (BESK1E-S, DBSK1E-D)
***KEYWORDS  EXPONENTIALLY SCALED, FNLIB, HYPERBOLIC BESSEL FUNCTION,
             MODIFIED BESSEL FUNCTION, ORDER ONE, SPECIAL FUNCTIONS,
             THIRD KIND
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBSK1E(S) computes the double precision exponentially scaled
 modified (hyperbolic) Bessel function of the third kind of order
 one for positive double precision argument X.

 Series for BK1        on the interval  0.          to  4.00000E+00
                                        with weighted error   9.16E-32
                                         log weighted error  31.04
                                 significant figures required  30.61
                                      decimal places required  31.64

 Series for AK1        on the interval  1.25000E-01 to  5.00000E-01
                                        with weighted error   3.07E-32
                                         log weighted error  31.51
                                 significant figures required  30.71
                                      decimal places required  32.30

 Series for AK12       on the interval  0.          to  1.25000E-01
                                        with weighted error   2.41E-32
                                         log weighted error  31.62
                                 significant figures required  30.25
                                      decimal places required  32.38

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DBESI1, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   END PROLOGUE
```

DBSKES

```
      SUBROUTINE DBSKES (XNU, X, NIN, BKE)
***BEGIN PROLOGUE  DBSKES
***PURPOSE  Compute a sequence of exponentially scaled modified Bessel
            functions of the third kind of fractional order.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C10B3
***TYPE      DOUBLE PRECISION (BESKES-S, DBSKES-D)
***KEYWORDS  EXPONENTIALLY SCALED, FNLIB, FRACTIONAL ORDER,
             MODIFIED BESSEL FUNCTION, SEQUENCE OF BESSEL FUNCTIONS,
             SPECIAL FUNCTIONS, THIRD KIND
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DBSKES(XNU,X,NIN,BKE) computes a double precision sequence
 of exponentially scaled modified Bessel functions
 of the third kind of order XNU + I at X, where X .GT. 0,
 XNU lies in (-1,1), and I = 0, 1, ... , NIN - 1, if NIN is positive
 and I = 0, -1, ... , NIN + 1, if NIN is negative.  On return, the
 vector BKE(.) contains the results at X for order starting at XNU.
 XNU, X, and BKE are double precision.  NIN is integer.

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, D9KNUS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770601  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  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)
   END PROLOGUE
```

DBSKIN

```
      SUBROUTINE DBSKIN (X, N, KODE, M, Y, NZ, IERR)
***BEGIN PROLOGUE  DBSKIN
***PURPOSE  Compute repeated integrals of the K-zero Bessel function.
***LIBRARY   SLATEC
***CATEGORY  C10F
***TYPE      DOUBLE PRECISION (BSKIN-S, DBSKIN-D)
***KEYWORDS  BICKLEY FUNCTIONS, EXPONENTIAL INTEGRAL,
             INTEGRALS OF BESSEL FUNCTIONS, K-ZERO BESSEL FUNCTION
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

         The following definitions are used in DBSKIN:

     Definition 1
         KI(0,X) = K-zero Bessel function.

     Definition 2
         KI(N,X) = Bickley Function
                 =  integral from X to infinity of KI(N-1,t)dt
                     for X .ge. 0 and N = 1,2,...
```

```
    DBSKIN computes a sequence of Bickley functions (repeated integrals
    of the K0 Bessel function); i.e. for fixed X and N and for K=1,...,
    DBSKIN computes the sequence

                      Y(K) =         KI(N+K-1,X) for KODE=1
          or
                      Y(K) = EXP(X)*KI(N+K-1,X) for KODE=2,

         for N.ge.0 and X.ge.0 (N and X cannot be zero simultaneously).

      INPUT      X is DOUBLE PRECISION
        X      - Argument, X .ge. 0.0D0
        N      - Order of first member of the sequence N .ge. 0
        KODE   - Selection parameter
             KODE = 1 returns Y(K)=        KI(N+K-1,X), K=1,M
                  = 2 returns Y(K)=EXP(X)*KI(N+K-1,X), K=1,M
        M      - Number of members in the sequence, M.ge.1

       OUTPUT     Y is a DOUBLE PRECISION VECTOR
         Y      - A vector of dimension at least M containing the
                  sequence selected by KODE.
         NZ     - Underflow flag
                  NZ = 0 means computation completed
                     = 1 means an exponential underflow occurred on
                         KODE=1.  Y(K)=0.0D0, K=1,...,M is returned
                         KODE=1 AND Y(K)=0.0E0, K=1,...,M IS RETURNED
         IERR   - Error flag
                    IERR=0, Normal return, computation completed
                    IERR=1, Input error,   no computation
                    IERR=2, Error,         no computation
                            Algorithm termination condition not met

         The nominal computational accuracy is the maximum of unit
         roundoff (=D1MACH(4)) and 1.0D-18 since critical constants
         are given to only 18 digits.
```

```
         BSKIN is the single precision version of DBSKIN.

 *Long Description:

         Numerical recurrence on

      (L-1)*KI(L,X) = X(KI(L-3,X) - KI(L-1,X)) + (L-2)*KI(L-2,X)

         is stable where recurrence is carried forward or backward
         away from INT(X+0.5).  The power series for indices 0,1 and 2
         on 0.le.X.le.2 starts a stable recurrence for indices
         greater than 2.  If N is sufficiently large (N.gt.NLIM), the
         uniform asymptotic expansion for N to INFINITY is more
         economical.  On X.gt.2 the recursion is started by evaluating
         the uniform expansion for the three members whose indices are
         closest to INT(X+0.5) within the set N,...,N+M-1.  Forward
         recurrence, backward recurrence or both complete the
         sequence depending on the relation of INT(X+0.5) to the
         indices N,...,N+M-1.

***REFERENCES  D. E. Amos, Uniform asymptotic expansions for
                 exponential integrals E(N,X) and Bickley functions
                 KI(N,X), ACM Transactions on Mathematical Software,
                 1983.
               D. E. Amos, A portable Fortran subroutine for the
                 Bickley functions KI(N,X), Algorithm 609, ACM
                 Transactions on Mathematical Software, 1983.
***ROUTINES CALLED  D1MACH, DBKIAS, DBKISR, DEXINT, DGAMRN, I1MACH
***REVISION HISTORY  (YYMMDD)
   820601  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   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
```

DBSPDR

```
      SUBROUTINE DBSPDR (T, A, N, K, NDERIV, AD)
***BEGIN PROLOGUE  DBSPDR
***PURPOSE  Use the B-representation to construct a divided difference
            table preparatory to a (right) derivative calculation.
***LIBRARY   SLATEC
***CATEGORY  E3, K6
***TYPE      DOUBLE PRECISION (BSPDR-S, DBSPDR-D)
***KEYWORDS  B-SPLINE, DATA FITTING, DIFFERENTIATION OF SPLINES,
             INTERPOLATION
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Written by Carl de Boor and modified by D. E. Amos

      Abstract     **** a double precision routine ****
          DBSPDR is the BSPLDR routine of the reference.

          DBSPDR uses the B-representation (T,A,N,K) to construct a
          divided difference table ADIF preparatory to a (right)
          derivative calculation in DBSPEV.  The lower triangular matrix
          ADIF is stored in vector AD by columns.  The arrays are
          related by

            ADIF(I,J) = AD(I-J+1 + (2*N-J+2)*(J-1)/2)

          I = J,N   ,   J=1,NDERIV.

      Description of Arguments

          Input      T,A are double precision
           T       - knot vector of length N+K
           A       - B-spline coefficient vector of length N
           N       - number of B-spline coefficients
                     N = sum of knot multiplicities-K
           K       - order of the spline, K .GE. 1
           NDERIV  - number of derivatives, 1 .LE. NDERIV .LE. K.
                     NDERIV=1 gives the zero-th derivative =
                     function value

          Output     AD is double precision
           AD      - table of differences in a vector of length
                     (2*N-NDERIV+1)*NDERIV/2 for input to DBSPEV

      Error Conditions
          Improper input is a fatal error

***REFERENCES  Carl de Boor, Package for calculating with B-splines,
                 SIAM Journal on Numerical Analysis 14, 3 (June 1977),
                 pp. 441-472.
***ROUTINES CALLED  XERMSG
***REVISION HISTORY  (YYMMDD)
   800901  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  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
```

DBSPEV

```
      SUBROUTINE DBSPEV (T, AD, N, K, NDERIV, X, INEV, SVALUE, WORK)
***BEGIN PROLOGUE  DBSPEV
***PURPOSE  Calculate the value of the spline and its derivatives from
            the B-representation.
***LIBRARY   SLATEC
***CATEGORY  E3, K6
***TYPE      DOUBLE PRECISION (BSPEV-S, DBSPEV-D)
***KEYWORDS  B-SPLINE, DATA FITTING, INTERPOLATION, SPLINES
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Written by Carl de Boor and modified by D. E. Amos

      Abstract    **** a double precision routine ****
          DBSPEV is the BSPLEV routine of the reference.

          DBSPEV calculates the value of the spline and its derivatives
          at X from the B-representation (T,A,N,K) and returns them in
          SVALUE(I),I=1,NDERIV, T(K) .LE. X .LE. T(N+1).  AD(I) can be
          the B-spline coefficients A(I), I=1,N) if NDERIV=1.  Otherwise
          AD must be computed before hand by a call to DBSPDR (T,A,N,K,
          NDERIV,AD).  If X=T(I),I=K,N), right limiting values are
          obtained.

          To compute left derivatives or left limiting values at a
          knot T(I), replace N by I-1 and set X=T(I), I=K+1,N+1.

          DBSPEV calls DINTRV, DBSPVN

      Description of Arguments

          Input      T,AD,X, are double precision
           T       - knot vector of length N+K
           AD      - vector of length (2*N-NDERIV+1)*NDERIV/2 containing
                     the difference table from DBSPDR.
           N       - number of B-spline coefficients
                     N = sum of knot multiplicities-K
           K       - order of the B-spline, K .GE. 1
           NDERIV  - number of derivatives, 1 .LE. NDERIV .LE. K.
                     NDERIV=1 gives the zero-th derivative =
                     function value
           X       - argument, T(K) .LE. X .LE. T(N+1)
           INEV    - an initialization parameter which must be set
                     to 1 the first time DBSPEV is called.

          Output     SVALUE,WORK are double precision
           INEV    - INEV contains information for efficient process-
                     ing after the initial call and INEV must not
                     be changed by the user.  Distinct splines require
                     distinct INEV parameters.
           SVALUE  - vector of length NDERIV containing the spline
                     value in SVALUE(1) and the NDERIV-1 derivatives
                     in the remaining components.
           WORK    - work vector of length 3*K

      Error Conditions
```

```
         Improper input is a fatal error.

***REFERENCES  Carl de Boor, Package for calculating with B-splines,
                 SIAM Journal on Numerical Analysis 14, 3 (June 1977),
                 pp. 441-472.
***ROUTINES CALLED  DBSPVN, DINTRV, XERMSG
***REVISION HISTORY  (YYMMDD)
   800901  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
```

DBSPPP

```
      SUBROUTINE DBSPPP (T, A, N, K, LDC, C, XI, LXI, WORK)
***BEGIN PROLOGUE  DBSPPP
***PURPOSE  Convert the B-representation of a B-spline to the piecewise
            polynomial (PP) form.
***LIBRARY   SLATEC
***CATEGORY  E3, K6
***TYPE      DOUBLE PRECISION (BSPPP-S, DBSPPP-D)
***KEYWORDS  B-SPLINE, PIECEWISE POLYNOMIAL
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Written by Carl de Boor and modified by D. E. Amos

      Abstract    **** a double precision routine ****
          DBSPPP is the BSPLPP routine of the reference.

          DBSPPP converts the B-representation (T,A,N,K) to the
          piecewise polynomial (PP) form (C,XI,LXI,K) for use with
          DPPVAL.  Here XI(*), the break point array of length LXI, is
          the knot array T(*) with multiplicities removed.  The columns
          of the matrix C(I,J) contain the right Taylor derivatives
          for the polynomial expansion about XI(J) for the intervals
          XI(J) .LE. X .LE. XI(J+1), I=1,K, J=1,LXI.  Function DPPVAL
          makes this evaluation at a specified point X in
          XI(1) .LE. X .LE. XI(LXI+1)

      Description of Arguments

          Input      T,A are double precision
           T       - knot vector of length N+K
           A       - B-spline coefficient vector of length N
           N       - number of B-spline coefficients
                     N = sum of knot multiplicities-K
           K       - order of the B-spline, K .GE. 1
           LDC     - leading dimension of C, LDC .GE. K

          Output     C,XI,WORK are double precision
           C       - matrix of dimension at least (K,LXI) containing
                     right derivatives at break points
           XI      - XI break point vector of length LXI+1
           LXI     - number of break points, LXI .LE. N-K+1
           WORK    - work vector of length K*(N+3)

      Error Conditions
          Improper input is a fatal error

***REFERENCES  Carl de Boor, Package for calculating with B-splines,
                 SIAM Journal on Numerical Analysis 14, 3 (June 1977),
                 pp. 441-472.
***ROUTINES CALLED  DBSPDR, DBSPEV, XERMSG
***REVISION HISTORY  (YYMMDD)
   800901  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
```

DBSPVD

```
      SUBROUTINE DBSPVD (T, K, NDERIV, X, ILEFT, LDVNIK, VNIKX, WORK)
***BEGIN PROLOGUE  DBSPVD
***PURPOSE  Calculate the value and all derivatives of order less than
            NDERIV of all basis functions which do not vanish at X.
***LIBRARY   SLATEC
***CATEGORY  E3, K6
***TYPE      DOUBLE PRECISION (BSPVD-S, DBSPVD-D)
***KEYWORDS  DIFFERENTIATION OF B-SPLINE, EVALUATION OF B-SPLINE
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Written by Carl de Boor and modified by D. E. Amos

      Abstract    **** a double precision routine ****

          DBSPVD is the BSPLVD routine of the reference.

          DBSPVD calculates the value and all derivatives of order
          less than NDERIV of all basis functions which do not
          (possibly) vanish at X.  ILEFT is input such that
          T(ILEFT) .LE. X .LT. T(ILEFT+1).  A call to INTRV(T,N+1,X,
          ILO,ILEFT,MFLAG) will produce the proper ILEFT.  The output of
          DBSPVD is a matrix VNIKX(I,J) of dimension at least (K,NDERIV)
          whose columns contain the K nonzero basis functions and
          their NDERIV-1 right derivatives at X, I=1,K, J=1,NDERIV.
          These basis functions have indices ILEFT-K+I, I=1,K,
          K .LE. ILEFT .LE. N.  The nonzero part of the I-th basis
          function lies in (T(I),T(I+K)), I=1,N).

          If X=T(ILEFT+1) then VNIKX contains left limiting values
          (left derivatives) at T(ILEFT+1).  In particular, ILEFT = N
          produces left limiting values at the right end point
          X=T(N+1).  To obtain left limiting values at T(I), I=K+1,N+1,
          set X= next lower distinct knot, call INTRV to get ILEFT,
          set X=T(I), and then call DBSPVD.

      Description of Arguments
          Input      T,X are double precision
           T       - knot vector of length N+K, where
                     N = number of B-spline basis functions
                     N = sum of knot multiplicities-K
           K       - order of the B-spline, K .GE. 1
           NDERIV  - number of derivatives = NDERIV-1,
                     1 .LE. NDERIV .LE. K
           X       - argument of basis functions,
                     T(K) .LE. X .LE. T(N+1)
           ILEFT   - largest integer such that
                     T(ILEFT) .LE. X .LT.  T(ILEFT+1)
           LDVNIK  - leading dimension of matrix VNIKX

          Output     VNIKX,WORK are double precision
           VNIKX   - matrix of dimension at least (K,NDERIV) contain-
                     ing the nonzero basis functions at X and their
                     derivatives columnwise.
           WORK    - a work vector of length (K+1)*(K+2)/2
```

```
     Error Conditions
         Improper input is a fatal error

***REFERENCES  Carl de Boor, Package for calculating with B-splines,
                 SIAM Journal on Numerical Analysis 14, 3 (June 1977),
                 pp. 441-472.
***ROUTINES CALLED  DBSPVN, XERMSG
***REVISION HISTORY  (YYMMDD)
   800901  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
```

DBSPVN

```
      SUBROUTINE DBSPVN (T, JHIGH, K, INDEX, X, ILEFT, VNIKX, WORK,
     +   IWORK)
***BEGIN PROLOGUE  DBSPVN
***PURPOSE  Calculate the value of all (possibly) nonzero basis
            functions at X.
***LIBRARY   SLATEC
***CATEGORY  E3, K6
***TYPE      DOUBLE PRECISION (BSPVN-S, DBSPVN-D)
***KEYWORDS  EVALUATION OF B-SPLINE
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Written by Carl de Boor and modified by D. E. Amos

      Abstract    **** a double precision routine ****
          DBSPVN is the BSPLVN routine of the reference.

          DBSPVN calculates the value of all (possibly) nonzero basis
          functions at X of order MAX(JHIGH,(J+1)*(INDEX-1)), where T(K)
          .LE. X .LE. T(N+1) and J=IWORK is set inside the routine on
          the first call when INDEX=1.  ILEFT is such that T(ILEFT) .LE.
          X .LT. T(ILEFT+1).  A call to DINTRV(T,N+1,X,ILO,ILEFT,MFLAG)
          produces the proper ILEFT.  DBSPVN calculates using the basic
          algorithm needed in DBSPVD.  If only basis functions are
          desired, setting JHIGH=K and INDEX=1 can be faster than
          calling DBSPVD, but extra coding is required for derivatives
          (INDEX=2) and DBSPVD is set up for this purpose.

          Left limiting values are set up as described in DBSPVD.

      Description of Arguments

          Input      T,X are double precision
           T       - knot vector of length N+K, where
                     N = number of B-spline basis functions
                     N = sum of knot multiplicities-K
           JHIGH   - order of B-spline, 1 .LE. JHIGH .LE. K
           K       - highest possible order
           INDEX   - INDEX = 1 gives basis functions of order JHIGH
                           = 2 denotes previous entry with work, IWORK
                               values saved for subsequent calls to
                               DBSPVN.
           X       - argument of basis functions,
                     T(K) .LE. X .LE. T(N+1)
           ILEFT   - largest integer such that
                     T(ILEFT) .LE. X .LT.  T(ILEFT+1)

          Output     VNIKX, WORK are double precision
           VNIKX   - vector of length K for spline values.
           WORK    - a work vector of length 2*K
           IWORK   - a work parameter.  Both WORK and IWORK contain
                     information necessary to continue for INDEX = 2.
                     When INDEX = 1 exclusively, these are scratch
                     variables and can be used for other purposes.

      Error Conditions
```

```
         Improper input is a fatal error.

***REFERENCES  Carl de Boor, Package for calculating with B-splines,
                 SIAM Journal on Numerical Analysis 14, 3 (June 1977),
                 pp. 441-472.
***ROUTINES CALLED  XERMSG
***REVISION HISTORY  (YYMMDD)
   800901  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
```

DBSQAD

```
      SUBROUTINE DBSQAD (T, BCOEF, N, K, X1, X2, BQUAD, WORK)
***BEGIN PROLOGUE  DBSQAD
***PURPOSE  Compute the integral of a K-th order B-spline using the
            B-representation.
***LIBRARY   SLATEC
***CATEGORY  H2A2A1, E3, K6
***TYPE      DOUBLE PRECISION (BSQAD-S, DBSQAD-D)
***KEYWORDS  INTEGRAL OF B-SPLINES, QUADRATURE
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Abstract    **** a double precision routine ****

          DBSQAD computes the integral on (X1,X2) of a K-th order
          B-spline using the B-representation (T,BCOEF,N,K).  Orders
          K as high as 20 are permitted by applying a 2, 6, or 10
          point Gauss formula on subintervals of (X1,X2) which are
          formed by included (distinct) knots.

          If orders K greater than 20 are needed, use DBFQAD with
          F(X) = 1.

          The maximum number of significant digits obtainable in
          DBSQAD is the smaller of 18 and the number of digits
          carried in double precision arithmetic.

      Description of Arguments
          Input      T,BCOEF,X1,X2 are double precision
            T      - knot array of length N+K
            BCOEF  - B-spline coefficient array of length N
            N      - length of coefficient array
            K      - order of B-spline, 1 .LE. K .LE. 20
            X1,X2  - end points of quadrature interval in
                     T(K) .LE. X .LE. T(N+1)

          Output     BQUAD,WORK are double precision
            BQUAD  - integral of the B-spline over (X1,X2)
            WORK   - work vector of length 3*K

      Error Conditions
          Improper input is a fatal error

***REFERENCES  D. E. Amos, Quadrature subroutines for splines and
                 B-splines, Report SAND79-1825, Sandia Laboratories,
                 December 1979.
***ROUTINES CALLED  DBVALU, DINTRV, XERMSG
***REVISION HISTORY  (YYMMDD)
   800901  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)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DBVALU

```
      DOUBLE PRECISION FUNCTION DBVALU (T, A, N, K, IDERIV, X, INBV,
     +   WORK)
***BEGIN PROLOGUE  DBVALU
***PURPOSE  Evaluate the B-representation of a B-spline at X for the
            function value or any of its derivatives.
***LIBRARY   SLATEC
***CATEGORY  E3, K6
***TYPE      DOUBLE PRECISION (BVALU-S, DBVALU-D)
***KEYWORDS  DIFFERENTIATION OF B-SPLINE, EVALUATION OF B-SPLINE
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Written by Carl de Boor and modified by D. E. Amos

      Abstract   **** a double precision routine ****
          DBVALU is the BVALUE function of the reference.

          DBVALU evaluates the B-representation (T,A,N,K) of a B-spline
          at X for the function value on IDERIV=0 or any of its
          derivatives on IDERIV=1,2,...,K-1.  Right limiting values
          (right derivatives) are returned except at the right end
          point X=T(N+1) where left limiting values are computed.  The
          spline is defined on T(K) .LE. X .LE. T(N+1).  DBVALU returns
          a fatal error message when X is outside of this interval.

          To compute left derivatives or left limiting values at a
          knot T(I), replace N by I-1 and set X=T(I), I=K+1,N+1.

          DBVALU calls DINTRV

      Description of Arguments

          Input      T,A,X are double precision
           T       - knot vector of length N+K
           A       - B-spline coefficient vector of length N
           N       - number of B-spline coefficients
                     N = sum of knot multiplicities-K
           K       - order of the B-spline, K .GE. 1
           IDERIV  - order of the derivative, 0 .LE. IDERIV .LE. K-1
                     IDERIV = 0 returns the B-spline value
           X       - argument, T(K) .LE. X .LE. T(N+1)
           INBV    - an initialization parameter which must be set
                     to 1 the first time DBVALU is called.

          Output     WORK,DBVALU are double precision
           INBV    - INBV contains information for efficient process-
                     ing after the initial call and INBV must not
                     be changed by the user.  Distinct splines require
                     distinct INBV parameters.
           WORK    - work vector of length 3*K.
           DBVALU  - value of the IDERIV-th derivative at X

      Error Conditions
          An improper input is a fatal error

***REFERENCES  Carl de Boor, Package for calculating with B-splines,
```

```
                 SIAM Journal on Numerical Analysis 14, 3 (June 1977),
                 pp. 441-472.
***ROUTINES CALLED  DINTRV, XERMSG
***REVISION HISTORY  (YYMMDD)
   800901  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  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
```

DBVSUP

```
      SUBROUTINE DBVSUP (Y, NROWY, NCOMP, XPTS, NXPTS, A, NROWA, ALPHA,
         NIC, B, NROWB, BETA, NFC, IGOFX, RE, AE, IFLAG, WORK, NDW,
     +   IWORK, NDIW, NEQIVP)
***BEGIN PROLOGUE  DBVSUP
***PURPOSE  Solve a linear two-point boundary value problem using
            superposition coupled with an orthonormalization procedure
            and a variable-step integration scheme.
***LIBRARY   SLATEC
***CATEGORY  I1B1
***TYPE      DOUBLE PRECISION (BVSUP-S, DBVSUP-D)
***KEYWORDS  ORTHONORMALIZATION, SHOOTING,
             TWO-POINT BOUNDARY VALUE PROBLEM
***AUTHOR  Scott, M. R., (SNLA)
           Watts, H. A., (SNLA)
***DESCRIPTION

 **********************************************************************

     Subroutine DBVSUP solves a linear two-point boundary-value problem
     of the form
                        DY/DX = MATRIX(X,U)*Y(X) + G(X,U)
                 A*Y(XINITIAL) = ALPHA ,  B*Y(XFINAL) = BETA

     coupled with the solution of the initial value problem

                        DU/DX = F(X,U)
                      U(XINITIAL) = ETA

 **********************************************************************
     ABSTRACT
        The method of solution uses superposition coupled with an
     orthonormalization procedure and a variable-step integration
     scheme.  Each time the superposition solutions start to
     lose their numerical linear independence, the vectors are
     reorthonormalized before integration proceeds.  The underlying
     principle of the algorithm is then to piece together the
     intermediate (orthogonalized) solutions, defined on the various
     subintervals, to obtain the desired solutions.

 **********************************************************************
     INPUT to DBVSUP
 **********************************************************************

     NROWY = actual row dimension of Y in calling program.
             NROWY must be .GE. NCOMP

     NCOMP = number of components per solution vector.
             NCOMP is equal to number of original differential
             equations.  NCOMP = NIC + NFC.

     XPTS = desired output points for solution. They must be monotonic.
            XINITIAL = XPTS(1)
            XFINAL = XPTS(NXPTS)

     NXPTS = number of output points.
```

```
    A(NROWA,NCOMP) = boundary condition matrix at XINITIAL
                     must be contained in (NIC,NCOMP) sub-matrix.

    NROWA = actual row dimension of A in calling program,
            NROWA must be .GE. NIC.

    ALPHA(NIC+NEQIVP) = boundary conditions at XINITIAL.
                        If NEQIVP .GT. 0 (see below), the boundary
                        conditions at XINITIAL for the initial value
                        equations must be stored starting in
                        position (NIC + 1) of ALPHA.
                        Thus,  ALPHA(NIC+K) = ETA(K).

    NIC = number of boundary conditions at XINITIAL.

    B(NROWB,NCOMP) = boundary condition matrix at XFINAL.
                     Must be contained in (NFC,NCOMP) sub-matrix.

    NROWB = actual row dimension of B in calling program,
            NROWB must be .GE. NFC.

    BETA(NFC) = boundary conditions at XFINAL.

    NFC = number of boundary conditions at XFINAL.

    IGOFX =0 -- The inhomogeneous term G(X) is identically zero.
          =1 -- The inhomogeneous term G(X) is not identically zero.
                (if IGOFX=1, then Subroutine DGVEC (or DUVEC) must be
                 supplied).

    RE = relative error tolerance used by the integrator.
         (see one of the integrators)

    AE = absolute error tolerance used by the integrator.
         (see one of the integrators)
**NOTE-  RE and AE should not both be zero.

    IFLAG = a status parameter used principally for output.
            However, for efficient solution of problems which
            are originally defined as COMPLEX*16 valued (but
            converted to double precision systems to use this code),
            the user must set IFLAG=13 on input. See the comment
            below for more information on solving such problems.

    WORK(NDW) = floating point array used for internal storage.

    NDW = actual dimension of work array allocated by user.
          An estimate for NDW can be computed from the following
           NDW = 130 + NCOMP**2 * (6 + NXPTS/2 + expected number of
                                     orthonormalizations/8)
          For the disk or tape storage mode,
           NDW = 6 * NCOMP**2 + 10 * NCOMP + 130
 However, when the ADAMS integrator is to be used, the estimates are
           NDW = 130 + NCOMP**2 * (13 + NXPTS/2 + expected number of
                                     orthonormalizations/8)
   and     NDW = 13 * NCOMP**2 + 22 * NCOMP + 130   , respectively.

    IWORK(NDIW) = integer array used for internal storage.

    NDIW = actual dimension of IWORK array allocated by user.
```

An estimate for NDIW can be computed from the following
NDIW = 68 + NCOMP * (1 + expected number of orthonormalizations)

**NOTE -- the amount of storage required is problem dependent and may be difficult to predict in advance. Experience has shown that for most problems 20 or fewer orthonormalizations should suffice. If the problem cannot be completed with the allotted storage, then a message will be printed which estimates the amount of storage necessary. In any case, the user can examine the IWORK array for the actual storage requirements, as described in the output information below.

NEQIVP = number of auxiliary initial value equations being added to the boundary value problem.

**NOTE -- Occasionally the coefficients matrix and/or G may be functions which depend on the independent variable X and on U, the solution of an auxiliary initial value problem. In order to avoid the difficulties associated with interpolation, the auxiliary equations may be solved simultaneously with the given boundary value problem. This initial value problem may be linear or nonlinear. See SAND77-1328 for an example.

The user must supply subroutines DFMAT, DGVEC, DUIVP and DUVEC, when needed (they must be so named), to evaluate the derivatives as follows

A. DFMAT must be supplied.

```
SUBROUTINE DFMAT(X,Y,YP)
X = independent variable (input to DFMAT)
Y = dependent variable vector (input to DFMAT)
YP = DY/DX = derivative vector (output from DFMAT)
```

Compute the derivatives for the homogeneous problem
YP(I) = DY(I)/DX = MATRIX(X) * Y(I) , I = 1,...,NCOMP

When (NEQIVP .GT. 0) and matrix is dependent on U as well as on X, the following common statement must be included in DFMAT

```
COMMON /DMLIVP/ NOFST
```

for convenience, the U vector is stored at the bottom of the Y array. Thus, during any call to DFMAT, U(I) is referenced by Y(NOFST + I).

Subroutine DBVDER calls DFMAT NFC times to evaluate the homogeneous equations and, if necessary, it calls DFMAT once in evaluating the particular solution. since X remains unchanged in this sequence of calls it is possible to realize considerable computational savings for complicated and expensive evaluations of the matrix entries. To do this the user merely passes a variable, say XS, via common where XS is defined in the main program to be any value except the initial X. Then the non-constant elements of matrix(x) appearing in the differential equations need only be computed if X is unequal to XS, whereupon XS is reset to X.

B. If NEQIVP .GT. 0 , DUIVP must also be supplied.

```
SUBROUTINE DUIVP(X,U,UP)
X = independent variable (input to DUIVP)
U = dependent variable vector (input to DUIVP)
UP = DU/DX = derivative vector (output from DUIVP)
```

Compute the derivatives for the auxiliary initial value eqs
UP(I) = DU(I)/DX, I = 1,...,NEQIVP.

Subroutine DBVDER calls DUIVP once to evaluate the derivatives for the auxiliary initial value equations.

C. If NEQIVP = 0 and IGOFX = 1 , DGVEC must be supplied.

```
SUBROUTINE DGVEC(X,G)
X = independent variable (input to DGVEC)
G = vector of inhomogeneous terms G(X) (output from
DGVEC)
```

Compute the inhomogeneous terms G(X)
G(I) = G(X) values for I = 1,...,NCOMP.

Subroutine DBVDER calls DGVEC in evaluating the particular solution provided G(X) is not identically zero. Thus, when IGOFX=0, the user need not write a DGVEC subroutine. Also, the user does not have to bother with the computational savings scheme for DGVEC as this is automatically achieved via the DBVDER subroutine.

D. If NEQIVP .GT. 0 and IGOFX = 1 , DUVEC must be supplied.

```
SUBROUTINE DUVEC(X,U,G)
X = independent variable (input to DUVEC)
U = dependent variable vector from the auxiliary initial
    value problem    (input to DUVEC)
G = array of inhomogeneous terms G(X,U)(output from DUVEC)
```

Compute the inhomogeneous terms G(X,U)
G(I) = G(X,U) values for I = 1,...,NCOMP.

Subroutine DBVDER calls DUVEC in evaluating the particular solution provided G(X,U) is not identically zero. Thus, when IGOFX=0, the user need not write a DUVEC subroutine.

The following is optional input to DBVSUP to give user more flexibility in use of code. See SAND75-0198, SAND77-1328, SAND77-1690, SAND78-0522, and SAND78-1501 for more information.

****CAUTION -- The user must zero out IWORK(1),...,IWORK(15) prior to calling DBVSUP. These locations define optional input and must be zero unless set to special values by the user as described below.

IWORK(1) -- number of orthonormalization points.
A value need be set only if IWORK(11) = 1

```
    IWORK(9) -- integrator and orthonormalization parameter
                (default value is 1)
                1 = RUNGE-KUTTA-FEHLBERG code using GRAM-SCHMIDT test.
                2 = ADAMS code using GRAM-SCHMIDT test.

    IWORK(11) -- orthonormalization points parameter
                 (default value is 0)
                 0 - orthonormalization points not pre-assigned.
                 1 - orthonormalization points pre-assigned in
                     the first IWORK(1) positions of work.

    IWORK(12) -- storage parameter
                 (default value is 0)
                 0 - all storage in core.
                 LUN - homogeneous and inhomogeneous solutions at
                     output points and orthonormalization information
                     are stored on disk.  The logical unit number to
                     be used for disk I/O (NTAPE) is set to IWORK(12).

    WORK(1),... -- pre-assigned orthonormalization points, stored
                   monotonically, corresponding to the direction
                   of integration.


                  ******************************************************
                  *** COMPLEX*16 VALUED PROBLEM ***
                  ******************************************************
**NOTE***
      Suppose the original boundary value problem is NC equations
    of the form
                    DW/DX = MAT(X,U)*W(X) + H(X,U)
                  R*W(XINITIAL)=GAMMA , S*W(XFINAL)=DELTA
    where all variables are COMPLEX*16 valued. The DBVSUP code can be
    used by converting to a double precision system of size 2*NC. To
    solve the larger dimensioned problem efficiently, the user must
    initialize IFLAG=13 on input and order the vector components
    according to Y(1)=DOUBLE PRECISION(W(1)),...,Y(NC)=DOUBLE
    PRECISION(W(NC)),Y(NC+1)=IMAG(W(1)),...., Y(2*NC)=IMAG(W(NC)).
    Then define
                        ...............................................
                        . DOUBLE PRECISION(MAT)    -IMAG(MAT) .
           MATRIX  =    .                             .
                        . IMAG(MAT)     DOUBLE PRECISION(MAT) .
                        ...............................................

    The matrices A,B and vectors G,ALPHA,BETA must be defined
    similarly. Further details can be found in SAND78-1501.


**********************************************************************
    OUTPUT from DBVSUP
**********************************************************************

    Y(NROWY,NXPTS) = solution at specified output points.

    IFLAG Output Values
           =-5 algorithm ,for obtaining starting vectors for the
               special COMPLEX*16 problem structure, was unable to
```

```
            obtain the initial vectors satisfying the necessary
            independence criteria.
        =-4 rank of boundary condition matrix A is less than NIC,
            as determined by DLSSUD.
        =-2 invalid input parameters.
        =-1 insufficient number of storage locations allocated for
            WORK or IWORK.

        =0 indicates successful solution.

        =1 a computed solution is returned but uniqueness of the
           solution of the boundary-value problem is questionable.
           For an eigenvalue problem, this should be treated as a
           successful execution since this is the expected mode
           of return.
        =2 a computed solution is returned but the existence of the
           solution to the boundary-value problem is questionable.
        =3 a nontrivial solution approximation is returned although
           the boundary condition matrix B*Y(XFINAL) is found to be
           nonsingular (to the desired accuracy level) while the
           right hand side vector is zero. To eliminate this type
           of return, the accuracy of the eigenvalue parameter
           must be improved.
        ***NOTE-We attempt to diagnose the correct problem behavior
           and report possible difficulties by the appropriate
           error flag.  However, the user should probably resolve
           the problem using smaller error tolerances and/or
           perturbations in the boundary conditions or other
           parameters. This will often reveal the correct
           interpretation for the problem posed.

        =13 maximum number of orthonormalizations attained before
            reaching XFINAL.
        =20-flag from integrator (DDERKF or DDEABM) values can
            range from 21 to 25.
        =30 solution vectors form a dependent set.

WORK(1),...,WORK(IWORK(1)) = orthonormalization points
                             determined by DBVPOR.

IWORK(1) = number of orthonormalizations performed by DBVPOR.

IWORK(2) = maximum number of orthonormalizations allowed as
           calculated from storage allocated by user.

IWORK(3),IWORK(4),IWORK(5),IWORK(6)   give information about
           actual storage requirements for WORK and IWORK
           arrays.  In particular,
                  required storage for  work array is
   IWORK(3) + IWORK(4)*(expected number of orthonormalizations)

                  required storage for IWORK array is
   IWORK(5) + IWORK(6)*(expected number of orthonormalizations)

IWORK(8) = final value of exponent parameter used in tolerance
           test for orthonormalization.

IWORK(16) = number of independent vectors returned from DMGSBV.
           It is only of interest when IFLAG=30 is obtained.
```

```
     IWORK(17) = numerically estimated rank of the boundary
                 condition matrix defined from B*Y(XFINAL)

 ***********************************************************************

     Necessary machine constants are defined in the Function
     Routine D1MACH. The user must make sure that the values
     set in D1MACH are relevant to the computer being used.

 ***********************************************************************
 ***********************************************************************

***REFERENCES  M. R. Scott and H. A. Watts, SUPORT - a computer code
                for two-point boundary-value problems via
                orthonormalization, SIAM Journal of Numerical
                Analysis 14, (1977), pp. 40-70.
              B. L. Darlow, M. R. Scott and H. A. Watts, Modifications
                of SUPORT, a linear boundary value problem solver
                Part I - pre-assigning orthonormalization points,
                auxiliary initial value problem, disk or tape storage,
                Report SAND77-1328, Sandia Laboratories, Albuquerque,
                New Mexico, 1977.
              B. L. Darlow, M. R. Scott and H. A. Watts, Modifications
                of SUPORT, a linear boundary value problem solver
                Part II - inclusion of an Adams integrator, Report
                SAND77-1690, Sandia Laboratories, Albuquerque,
                New Mexico, 1977.
              M. E. Lord and H. A. Watts, Modifications of SUPORT,
                a linear boundary value problem solver Part III -
                orthonormalization improvements, Report SAND78-0522,
                Sandia Laboratories, Albuquerque, New Mexico, 1978.
              H. A. Watts, M. R. Scott and M. E. Lord, Computational
                solution of complex*16 valued boundary problems,
                Report SAND78-1501, Sandia Laboratories,
                Albuquerque, New Mexico, 1978.
***ROUTINES CALLED  DEXBVP, DMACON, XERMSG
***COMMON BLOCKS    DML15T, DML17B, DML18J, DML5MC, DML8SZ
***REVISION HISTORY  (YYMMDD)
   750601  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890921  Realigned order of variables in certain COMMON blocks.
           (WRB)
   890921  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900510  Convert XERRWV calls to XERMSG calls, remove some extraneous
           comments.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DCBRT

```
      DOUBLE PRECISION FUNCTION DCBRT (X)
***BEGIN PROLOGUE  DCBRT
***PURPOSE  Compute the cube root.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C2
***TYPE      DOUBLE PRECISION (CBRT-S, DCBRT-D, CCBRT-C)
***KEYWORDS  CUBE ROOT, ELEMENTARY FUNCTIONS, FNLIB, ROOTS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DCBRT(X) calculates the double precision cube root for
 double precision argument X.

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, D9PAK, D9UPAK
***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
```

DCDOT

```
      SUBROUTINE DCDOT (N, FM, CX, INCX, CY, INCY, DCR, DCI)
***BEGIN PROLOGUE  DCDOT
***PURPOSE  Compute the inner product of two vectors with extended
            precision accumulation and result.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A4
***TYPE      COMPLEX (DSDOT-D, DCDOT-C)
***KEYWORDS  BLAS, COMPLEX VECTORS, DOT PRODUCT, INNER PRODUCT,
             LINEAR ALGEBRA, VECTOR
***AUTHOR  (UNKNOWN)
***DESCRIPTION

    Compute the dot product of 2 complex vectors, CX and CY, e.g.
    CX DOT CY, or, CXconjugate DOT CY.  The real and imaginary
    parts of CX and CY are converted to double precision, the dot
    product accumulation is done in double precision and the output
    is given as 2 double precision numbers, corresponding to the real
    and imaginary part of the result.
     Input
      N:  Number of complex components of CX and CY.
      FM: =+1.0   compute CX DOT CY.
          =-1.0   compute CXconjugate DOT CY.
      CX(N):
      CY(N):  Complex arrays of length N.
      INCX:(Integer)   Spacing of elements of CX to use
      INCY:(Integer)   Spacing of elements of CY to use.
     Output
      DCR:(Double Precision) Real part of dot product.
      DCI:(Double Precision) Imaginary part of dot product.

***REFERENCES  (NONE)
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   790101  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   890831  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   END PROLOGUE
```

DCG

```
      SUBROUTINE DCG(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  DCG
***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      DOUBLE 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)
     DOUBLE PRECISION B(N), X(N), A(NELT), TOL, ERR, R(N), Z(N)
     DOUBLE PRECISION P(N), DZ(N), RWORK(USER DEFINED)
     EXTERNAL MATVEC, MSOLVE

     CALL DCG(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       Double Precision B(N).
         Right-hand side vector.
 X      :INOUT    Double Precision 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       Double Precision 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 double precision 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 /DSLBLK/ 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 /DSLBLK/ 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 Double Precision.
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 Double Precision.
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*D1MACH(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     Double Precision R(N).
Z      :WORK     Double Precision Z(N).
P      :WORK     Double Precision P(N).
DZ     :WORK     Double Precision DZ(N).
        Double Precision arrays used for workspace.
RWORK  :WORK     Double Precision RWORK(USER DEFINED).
        Double Precision 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 DSDCG and DSICCG 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 double precision 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, and 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.

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 DSDCG, DSICCG

***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 D1MACH, DAXPY, DCOPY, DDOT, ISDCG

***REVISION HISTORY (YYMMDD)

890404 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)
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 DSLBLK.  (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
```

DCGN

```
      SUBROUTINE DCGN(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  DCGN
***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      DOUBLE 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)
     DOUBLE PRECISION B(N), X(N), A(NELT), TOL, ERR, R(N), Z(N)
     DOUBLE PRECISION P(N), ATP(N), ATZ(N), DZ(N), ATDZ(N)
     DOUBLE PRECISION RWORK(USER DEFINED)
     EXTERNAL MATVEC, MTTVEC, MSOLVE

     CALL DCGN(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       Double Precision B(N).
         Right-hand side vector.
 X      :INOUT    Double Precision 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       Double Precision 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 double precision 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 /DSLBLK/ 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 /DSLBLK/ 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    Double Precision.
        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      Double Precision.
        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*D1MACH(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     Double Precision R(N).
Z      :WORK     Double Precision Z(N).
P      :WORK     Double Precision P(N).
ATP    :WORK     Double Precision ATP(N).
ATZ    :WORK     Double Precision ATZ(N).
DZ     :WORK     Double Precision DZ(N).
ATDZ   :WORK     Double Precision ATDZ(N).
        Double Precision arrays used for workspace.
RWORK  :WORK     Double Precision RWORK(USER DEFINED).
        Double Precision 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, where  x  = A'y.
      In PCG method the iteration count is determined by condition
                                 -1
      number of the  matrix (M  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 DCGN 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 double precision 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, and 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.

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  DSDCGN, DSLUCN, ISDCGN
***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  D1MACH, DAXPY, DCOPY, DDOT, ISDCGN
***REVISION HISTORY  (YYMMDD)
   890404  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)
   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 DSLBLK.  (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
```

DCGS

```
      SUBROUTINE DCGS(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  DCGS
***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      DOUBLE 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)
      DOUBLE PRECISION B(N), X(N), A(NELT), TOL, ERR, R(N), R0(N), P(N)
      DOUBLE PRECISION Q(N), U(N), V1(N), V2(N), RWORK(USER DEFINED)
      EXTERNAL MATVEC, MSOLVE

      CALL DCGS(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       Double Precision B(N).
         Right-hand side vector.
 X      :INOUT    Double Precision 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       Double Precision 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  double precision 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 /DSLBLK/ 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    Double Precision.
        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      Double Precision.
        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*D1MACH(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     Double Precision R(N).
R0     :WORK     Double Precision R0(N).
P      :WORK     Double Precision P(N).
Q      :WORK     Double Precision Q(N).
U      :WORK     Double Precision U(N).
V1     :WORK     Double Precision V1(N).
V2     :WORK     Double Precision V2(N).
        Double Precision arrays used for workspace.
RWORK  :WORK     Double Precision RWORK(USER DEFINED).
        Double Precision 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 DSDBCG and DSLUCS 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 double precision 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, and 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.

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 DSDCGS, DSLUCS

***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 D1MACH, DAXPY, DDOT, ISDCGS

***REVISION HISTORY (YYMMDD)

```
890404  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)
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 DSLBLK.  (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
```

DCHDC

```
      SUBROUTINE DCHDC (A, LDA, P, WORK, JPVT, JOB, INFO)
***BEGIN PROLOGUE  DCHDC
***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      DOUBLE 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

      DCHDC 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      DOUBLE PRECISION(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   DOUBLE PRECISION.
                 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  DAXPY, DSWAP
***REVISION HISTORY  (YYMMDD)
   790319  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
```

DCHDD

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

     DCHDD 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, DCHDD
     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).  DCHDD will simultaneously downdate
     several triplets (Z,Y,RHO) along with R.
     For a less terse description of what DCHDD 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 double precision.

     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      DOUBLE PRECISION(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       DOUBLE PRECISION(P).
                X contains the row vector that is to
                be removed from R.  X is not altered by DCHDD.

        Z       DOUBLE PRECISION(LDZ,N)Z), 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       DOUBLE PRECISION(NZ).
                Y contains the scalars for the downdating
                of the vectors Z.  Y is not altered by DCHDD.

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

    On Return

        R
        Z       contain the downdated quantities.
        RHO

        C       DOUBLE PRECISION(P).
                C contains the cosines of the transforming
                rotations.

        S       DOUBLE PRECISION(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 RHO's 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  DDOT, DNRM2
***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
```

DCHEX

```
      SUBROUTINE DCHEX (R, LDR, P, K, L, Z, LDZ, NZ, C, S, JOB)
***BEGIN PROLOGUE  DCHEX
***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      DOUBLE 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

     DCHEX 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), DCHEX 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 DCHEX 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 double precision.  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      DOUBLE PRECISION(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      DOUBLE PRECISION(LDZ,N)Z), 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      DOUBLE PRECISION(P).
       C contains the cosines of the transforming rotations.

S      DOUBLE PRECISION(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  DROTG
***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
```

DCHFDV

```
      SUBROUTINE DCHFDV (X1, X2, F1, F2, D1, D2, NE, XE, FE, DE, NEXT,
     +   IERR)
***BEGIN PROLOGUE  DCHFDV
***PURPOSE  Evaluate a cubic polynomial given in Hermite form and its
            first derivative at an array of points.  While designed for
            use by DPCHFD, it may be useful directly as an evaluator
            for a piecewise cubic Hermite function in applications,
            such as graphing, where the interval is known in advance.
            If only function values are required, use DCHFEV instead.
***LIBRARY   SLATEC (PCHIP)
***CATEGORY  E3, H1
***TYPE      DOUBLE PRECISION (CHFDV-S, DCHFDV-D)
***KEYWORDS  CUBIC HERMITE DIFFERENTIATION, CUBIC HERMITE EVALUATION,
             CUBIC POLYNOMIAL EVALUATION, PCHIP
***AUTHOR  Fritsch, F. N., (LLNL)
             Lawrence Livermore National Laboratory
             P.O. Box 808  (L-316)
             Livermore, CA  94550
             FTS 532-4275, (510) 422-4275
***DESCRIPTION

        DCHFDV:  Cubic Hermite Function and Derivative Evaluator

     Evaluates the cubic polynomial determined by function values
     F1,F2 and derivatives D1,D2 on interval (X1,X2), together with
     its first derivative, at the points  XE(J), J=1(1)NE.

     If only function values are required, use DCHFEV, instead.

 ----------------------------------------------------------------------

  Calling sequence:

        INTEGER  NE, NEXT(2), IERR
        DOUBLE PRECISION  X1, X2, F1, F2, D1, D2, XE(NE), FE(NE),
                          DE(NE)

        CALL  DCHFDV (X1,X2, F1,F2, D1,D2, NE, XE, FE, DE, NEXT, IERR)

   Parameters:

     X1,X2 -- (input) endpoints of interval of definition of cubic.
           (Error return if  X1.EQ.X2 .)

     F1,F2 -- (input) values of function at X1 and X2, respectively.

     D1,D2 -- (input) values of derivative at X1 and X2, respectively.

     NE -- (input) number of evaluation points.  (Error return if
           NE.LT.1 .)

     XE -- (input) real*8 array of points at which the functions are to
           be evaluated.  If any of the XE are outside the interval
           [X1,X2], a warning error is returned in NEXT.

     FE -- (output) real*8 array of values of the cubic function
```

```
            defined by  X1,X2, F1,F2, D1,D2  at the points  XE.

      DE -- (output) real*8 array of values of the first derivative of
            the same function at the points  XE.

      NEXT -- (output) integer array indicating number of extrapolation
            points:
             NEXT(1) = number of evaluation points to left of interval.
             NEXT(2) = number of evaluation points to right of interval.

      IERR -- (output) error flag.
            Normal return:
               IERR = 0  (no errors).
            "Recoverable" errors:
               IERR = -1  if NE.LT.1 .
               IERR = -2  if X1.EQ.X2 .
                 (Output arrays have not been changed in either case.)

***REFERENCES  (NONE)
***ROUTINES CALLED  XERMSG
***REVISION HISTORY  (YYMMDD)
   811019  DATE WRITTEN
   820803  Minor cosmetic changes for release 1.
   870707  Corrected XERROR calls for d.p. names(s).
   870813  Minor cosmetic changes.
   890411  Added SAVE statements (Vers. 3.2).
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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)
   END PROLOGUE
```

DCHFEV

```
      SUBROUTINE DCHFEV (X1, X2, F1, F2, D1, D2, NE, XE, FE, NEXT, IERR)
***BEGIN PROLOGUE  DCHFEV
***PURPOSE  Evaluate a cubic polynomial given in Hermite form at an
            array of points.  While designed for use by DPCHFE, it may
            be useful directly as an evaluator for a piecewise cubic
            Hermite function in applications, such as graphing, where
            the interval is known in advance.
***LIBRARY   SLATEC (PCHIP)
***CATEGORY  E3
***TYPE      DOUBLE PRECISION (CHFEV-S, DCHFEV-D)
***KEYWORDS  CUBIC HERMITE EVALUATION, CUBIC POLYNOMIAL EVALUATION,
             PCHIP
***AUTHOR  Fritsch, F. N., (LLNL)
             Lawrence Livermore National Laboratory
             P.O. Box 808  (L-316)
             Livermore, CA  94550
             FTS 532-4275, (510) 422-4275
***DESCRIPTION

          DCHFEV:  Cubic Hermite Function EValuator

     Evaluates the cubic polynomial determined by function values
     F1,F2 and derivatives D1,D2 on interval (X1,X2) at the points
     XE(J), J=1(1)NE.

 ----------------------------------------------------------------------

  Calling sequence:

        INTEGER  NE, NEXT(2), IERR
        DOUBLE PRECISION  X1, X2, F1, F2, D1, D2, XE(NE), FE(NE)

        CALL  DCHFEV (X1,X2, F1,F2, D1,D2, NE, XE, FE, NEXT, IERR)

   Parameters:

     X1,X2 -- (input) endpoints of interval of definition of cubic.
           (Error return if  X1.EQ.X2 .)

     F1,F2 -- (input) values of function at X1 and X2, respectively.

     D1,D2 -- (input) values of derivative at X1 and X2, respectively.

     NE -- (input) number of evaluation points.  (Error return if
           NE.LT.1 .)

     XE -- (input) real*8 array of points at which the function is to
           be evaluated.  If any of the XE are outside the interval
           [X1,X2], a warning error is returned in NEXT.

     FE -- (output) real*8 array of values of the cubic function
           defined by  X1,X2, F1,F2, D1,D2  at the points  XE.

     NEXT -- (output) integer array indicating number of extrapolation
           points:
            NEXT(1) = number of evaluation points to left of interval.
```

```
            NEXT(2) = number of evaluation points to right of interval.

     IERR -- (output) error flag.
           Normal return:
              IERR = 0  (no errors).
           "Recoverable" errors:
              IERR = -1  if NE.LT.1 .
              IERR = -2  if X1.EQ.X2 .
                (The FE-array has not been changed in either case.)

***REFERENCES  (NONE)
***ROUTINES CALLED  XERMSG
***REVISION HISTORY  (YYMMDD)
   811019  DATE WRITTEN
   820803  Minor cosmetic changes for release 1.
   870813  Corrected XERROR calls for d.p. names(s).
   890206  Corrected XERROR calls.
   890411  Added SAVE statements (Vers. 3.2).
   890531  Changed all specific intrinsics to generic.  (WRB)
   890703  Corrected category record.  (WRB)
   890831  Modified array declarations.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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)
   END PROLOGUE
```

DCHU

```
      DOUBLE PRECISION FUNCTION DCHU (A, B, X)
***BEGIN PROLOGUE  DCHU
***PURPOSE  Compute the logarithmic confluent hypergeometric function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C11
***TYPE      DOUBLE PRECISION (CHU-S, DCHU-D)
***KEYWORDS  FNLIB, LOGARITHMIC CONFLUENT HYPERGEOMETRIC FUNCTION,
             SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DCHU(A,B,X) calculates the double precision logarithmic confluent
 hypergeometric function U(A,B,X) for double precision arguments
 A, B, and X.

 This routine is not valid when 1+A-B is close to zero if X is small.

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, D9CHU, DEXPRL, DGAMMA, DGAMR, DPOCH,
                    DPOCH1, XERMSG
***REVISION HISTORY  (YYMMDD)
   770801  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)
   900727  Added EXTERNAL statement.  (WRB)
   END PROLOGUE
```

DCHUD

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

      DCHUD 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, DCHUD 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).  DCHUD will simultaneously update
      several triplets (Z,Y,RHO).
      For a less terse description of what DCHUD 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 double precision.

      On Entry

          R      DOUBLE PRECISION(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      DOUBLE PRECISION(P).
                 X contains the row to be added to R.  X is
                 not altered by DCHUD.

          Z      DOUBLE PRECISION(LDZ,N)Z), 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      DOUBLE PRECISION(NZ).
                 Y contains the scalars for updating the vectors
                 Z.  Y is not altered by DCHUD.

          RHO    DOUBLE PRECISION(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      DOUBLE PRECISION(P).
                 C contains the cosines of the transforming
                 rotations.

          S      DOUBLE PRECISION(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  DROTG
***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
```

DCKDER

```
      SUBROUTINE DCKDER (M, N, X, FVEC, FJAC, LDFJAC, XP, FVECP, MODE,
     +   ERR)
***BEGIN PROLOGUE  DCKDER
***PURPOSE  Check the gradients of M nonlinear functions in N
            variables, evaluated at a point X, for consistency
            with the functions themselves.
***LIBRARY   SLATEC
***CATEGORY  F3, G4C
***TYPE      DOUBLE PRECISION (CHKDER-S, DCKDER-D)
***KEYWORDS  GRADIENTS, JACOBIAN, MINPACK, NONLINEAR
***AUTHOR  Hiebert, K. L. (SNLA)
***DESCRIPTION

   This subroutine is a companion routine to DNSQ and DNSQE. It may
   be used to check the coding of the Jacobian calculation.

     SUBROUTINE DCKDER

     This subroutine checks the gradients of M nonlinear functions
     in N variables, evaluated at a point X, for consistency with
     the functions themselves. The user must call DCKDER twice,
     first with MODE = 1 and then with MODE = 2.

     MODE = 1. On input, X must contain the point of evaluation.
               On output, XP is set to a neighboring point.

     MODE = 2. On input, FVEC must contain the functions and the
                         rows of FJAC must contain the gradients
                         of the respective functions each evaluated
                         at X, and FVECP must contain the functions
                         evaluated at XP.
               On output, ERR contains measures of correctness of
                          the respective gradients.

     The subroutine does not perform reliably if cancellation or
     rounding errors cause a severe loss of significance in the
     evaluation of a function. Therefore, none of the components
     of X should be unusually small (in particular, zero) or any
     other value which may cause loss of significance.

     The SUBROUTINE statement is

       SUBROUTINE DCKDER(M,N,X,FVEC,FJAC,LDFJAC,XP,FVECP,MODE,ERR)

     where

       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.

       X is an input array of length N.

       FVEC is an array of length M. On input when MODE = 2,
         FVEC must contain the functions evaluated at X.
```

```
        FJAC is an M by N array. On input when MODE = 2,
          the rows of FJAC must contain the gradients of
          the respective functions evaluated at X.

        LDFJAC is a positive integer input parameter not less than M
          which specifies the leading dimension of the array FJAC.

        XP is an array of length N. On output when MODE = 1,
          XP is set to a neighboring point of X.

        FVECP is an array of length M. On input when MODE = 2,
          FVECP must contain the functions evaluated at XP.

        MODE is an integer input variable set to 1 on the first call
          and 2 on the second. Other values of MODE are equivalent
          to MODE = 1.

        ERR is an array of length M. On output when MODE = 2,
          ERR contains measures of correctness of the respective
          gradients. If there is no severe loss of significance,
          then if ERR(I) is 1.0 the I-th gradient is correct,
          while if ERR(I) is 0.0 the I-th gradient is incorrect.
          For values of ERR between 0.0 and 1.0, the categorization
          is less certain. In general, a value of ERR(I) greater
          than 0.5 indicates that the I-th gradient is probably
          correct, while a value of ERR(I) less than 0.5 indicates
          that the I-th gradient is probably incorrect.

***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  D1MACH
***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)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DCOPY

```
      SUBROUTINE DCOPY (N, DX, INCX, DY, INCY)
***BEGIN PROLOGUE  DCOPY
***PURPOSE  Copy a vector.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A5
***TYPE      DOUBLE 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)
       DX  double precision vector with N elements
     INCX  storage spacing between elements of DX
       DY  double precision vector with N elements
     INCY  storage spacing between elements of DY

     --Output--
       DY  copy of vector DX (unchanged if N .LE. 0)

     Copy double precision DX to double precision DY.
     For I = 0 to N-1, copy DX(LX+I*INCX) to DY(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
```

DCOPYM

```
      SUBROUTINE DCOPYM (N, DX, INCX, DY, INCY)
***BEGIN PROLOGUE  DCOPYM
***PURPOSE  Copy the negative of a vector to a vector.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A5
***TYPE      DOUBLE 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)
      DX   Double precision vector with N elements
    INCX   Storage spacing between elements of DX
      DY*  Double precision negative copy of DX
    INCY   Storage spacing between elements of DY

      ***  Note that DY = -DX  ***

     Copy negative of d.p. DX to d.p. DY.  For I=0 to N-1,
     copy  -DX(LX+I*INCX) to DY(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
   861211  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
```

DCOSDG

```
      DOUBLE PRECISION FUNCTION DCOSDG (X)
***BEGIN PROLOGUE  DCOSDG
***PURPOSE  Compute the cosine of an argument in degrees.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C4A
***TYPE      DOUBLE PRECISION (COSDG-S, DCOSDG-D)
***KEYWORDS  COSINE, DEGREES, ELEMENTARY FUNCTIONS, FNLIB,
             TRIGONOMETRIC
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DCOSDG(X) calculates the double precision trigonometric cosine
 for double precision argument X in units of 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
```

DCOT

```
      DOUBLE PRECISION FUNCTION DCOT (X)
***BEGIN PROLOGUE  DCOT
***PURPOSE  Compute the cotangent.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C4A
***TYPE      DOUBLE PRECISION (COT-S, DCOT-D, CCOT-C)
***KEYWORDS  COTANGENT, ELEMENTARY FUNCTIONS, FNLIB, TRIGONOMETRIC
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DCOT(X) calculates the double precision trigonometric cotangent
 for double precision argument X.  X is in units of radians.

 Series for COT        on the interval  0.          to  6.25000E-02
                                       with weighted error   5.52E-34
                                        log weighted error  33.26
                                 significant figures required  32.34
                                      decimal places required  33.85

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DCSEVL, INITDS, XERMSG
***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)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   920618  Removed space from variable names.  (RWC, WRB)
   END PROLOGUE
```

DCOV

```
      SUBROUTINE DCOV (FCN, IOPT, M, N, X, FVEC, R, LDR, INFO, WA1, WA2,
     +   WA3, WA4)
***BEGIN PROLOGUE  DCOV
***PURPOSE  Calculate the covariance matrix for a nonlinear data
            fitting problem.  It is intended to be used after a
            successful return from either DNLS1 or DNLS1E.
***LIBRARY   SLATEC
***CATEGORY  K1B1
***TYPE      DOUBLE PRECISION (SCOV-S, DCOV-D)
***KEYWORDS  COVARIANCE MATRIX, NONLINEAR DATA FITTING,
             NONLINEAR LEAST SQUARES
***AUTHOR  Hiebert, K. L., (SNLA)
***DESCRIPTION

  1. Purpose.

     DCOV calculates the covariance matrix for a nonlinear data
     fitting problem.  It is intended to be used after a
     successful return from either DNLS1 or DNLS1E. DCOV
     and DNLS1 (and DNLS1E) have compatible parameters.  The
     required external subroutine, FCN, is the same
     for all three codes, DCOV, DNLS1, and DNLS1E.

  2. Subroutine and Type Statements.

     SUBROUTINE DCOV(FCN,IOPT,M,N,X,FVEC,R,LDR,INFO,
                     WA1,WA2,WA3,WA4)
     INTEGER IOPT,M,N,LDR,INFO
     DOUBLE PRECISION X(N),FVEC(M),R(LDR,N),WA1(N),WA2(N),WA3(N),WA4(M)
     EXTERNAL FCN

  3. Parameters. All TYPE REAL parameters are DOUBLE PRECISION

      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 in DNLS1 or DNLS1E,
         then FCN must do the printing.  See the explanation of NPRINT
         in DNLS1 or DNLS1E.  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
         DOUBLE PRECISION X(N),FVEC(M)
         ----------
         FJAC and LDFJAC may be ignored       , if IOPT=1.
         DOUBLE PRECISION FJAC(LDFJAC,N)      , if IOPT=2.
         DOUBLE PRECISION FJAC(N)             , if IOPT=3.
         ----------
           If IFLAG=0, the values in X and FVEC are available
           for printing in DNLS1 or DNLS1E.
           IFLAG will never be zero when FCN is called by DCOV.
           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.  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 DCOV. 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 DNLS1 (or DNLS1E). 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,WA2 are work arrays of length N.
 and WA3

     WA4 is a work array of length M.

***REFERENCES  (NONE)
***ROUTINES CALLED  DENORM, DFDJC3, DQRFAC, DWUPDT, XERMSG
***REVISION HISTORY  (YYMMDD)
   810522  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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
```

DCPPLT

```
      SUBROUTINE DCPPLT (N, NELT, IA, JA, A, ISYM, IUNIT)
***BEGIN PROLOGUE  DCPPLT
***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      DOUBLE 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
     DOUBLE PRECISION A(NELT)

     CALL DCPPLT( 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       Double Precision 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 double precision 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)

```
921026  Corrected D to E in output format.  (FNF)
930701  Updated CATEGORY section.  (FNF, WRB)
END PROLOGUE
```

DCSEVL

```
      DOUBLE PRECISION FUNCTION DCSEVL (X, CS, N)
***BEGIN PROLOGUE  DCSEVL
***PURPOSE  Evaluate a Chebyshev series.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C3A2
***TYPE      DOUBLE PRECISION (CSEVL-S, DCSEVL-D)
***KEYWORDS  CHEBYSHEV SERIES, FNLIB, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

  Evaluate the N-term Chebyshev series CS at X.  Adapted from
  a method presented in the paper by Broucke referenced below.

       Input Arguments --
  X    value at which the series is to be evaluated.
  CS   array of N terms of a Chebyshev series.  In evaluating
       CS, only half the first coefficient is summed.
  N    number of terms in array CS.

***REFERENCES  R. Broucke, Ten subroutines for the manipulation of
                 Chebyshev series, Algorithm 446, Communications of
                 the A.C.M. 16, (1973) pp. 254-256.
               L. Fox and I. B. Parker, Chebyshev Polynomials in
                 Numerical Analysis, Oxford University Press, 1968,
                 page 56.
***ROUTINES CALLED  D1MACH, XERMSG
***REVISION HISTORY  (YYMMDD)
   770401  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)
   900329  Prologued revised extensively and code rewritten to allow
           X to be slightly outside interval (-1,+1).  (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DCV

```
      SUBROUTINE DCV (XVAL, NDATA, NCONST, NORD, NBKPT,
     +   BKPT, W)
***BEGIN PROLOGUE  DCV
***PURPOSE  Evaluate the variance function of the curve obtained
            by the constrained B-spline fitting subprogram DFC.
***LIBRARY   SLATEC
***CATEGORY  L7A3
***TYPE      DOUBLE PRECISION (CV-S, DCV-D)
***KEYWORDS  ANALYSIS OF COVARIANCE, B-SPLINE,
             CONSTRAINED LEAST SQUARES, CURVE FITTING
***AUTHOR  Hanson, R. J., (SNLA)
***DESCRIPTION

      DCV( ) is a companion function subprogram for DFC( ).  The
      documentation for DFC( ) has complete usage instructions.

      DCV( ) is used to evaluate the variance function of the curve
      obtained by the constrained B-spline fitting subprogram, DFC( ).
      The variance function defines the square of the probable error
      of the fitted curve at any point, XVAL.  One can use the square
      root of this variance function to determine a probable error band
      around the fitted curve.

      DCV( ) is used after a call to DFC( ).  MODE, an input variable to
      DFC( ), is used to indicate if the variance function is desired.
      In order to use DCV( ), MODE must equal 2 or 4 on input to DFC( ).
      MODE is also used as an output flag from DFC( ).  Check to make
      sure that MODE = 0 after calling DFC( ), indicating a successful
      constrained curve fit.  The array SDDATA, as input to DFC( ), must
      also be defined with the standard deviation or uncertainty of the
      Y values to use DCV( ).

      To evaluate the variance function after calling DFC( ) as stated
      above, use DCV( ) as shown here

           VAR=DCV(XVAL,NDATA,NCONST,NORD,NBKPT,BKPT,W)

      The variance function is given by

       VAR=(transpose of B(XVAL))*C*B(XVAL)/DBLE(MAX(NDATA-N,1))

      where N = NBKPT - NORD.

      The vector B(XVAL) is the B-spline basis function values at
      X=XVAL.  The covariance matrix, C, of the solution coefficients
      accounts only for the least squares equations and the explicitly
      stated equality constraints.  This fact must be considered when
      interpreting the variance function from a data fitting problem
      that has inequality constraints on the fitted curve.

      All the variables in the calling sequence for DCV( ) are used in
      DFC( ) except the variable XVAL.  Do not change the values of
      these variables between the call to DFC( ) and the use of DCV( ).

      The following is a brief description of the variables
```

```
     XVAL    The point where the variance is desired, a double
             precision variable.

     NDATA   The number of discrete (X,Y) pairs for which DFC( )
             calculated a piece-wise polynomial curve.

     NCONST  The number of conditions that constrained the B-spline in
             DFC( ).

     NORD    The order of the B-spline used in DFC( ).
             The value of NORD must satisfy 1 < NORD < 20 .

             (The order of the spline is one more than the degree of
             the piece-wise polynomial defined on each interval.  This
             is consistent with the B-spline package convention.  For
             example, NORD=4 when we are using piece-wise cubics.)

     NBKPT   The number of knots in the array BKPT(*).
             The value of NBKPT must satisfy NBKPT .GE. 2*NORD.

     BKPT(*) The double precision array of knots.  Normally the problem
             data interval will be included between the limits
             BKPT(NORD) and BKPT(NBKPT-NORD+1).  The additional end
             knots BKPT(I),I=1,...,NORD-1 and I=NBKPT-NORD+2,...,NBKPT,
             are required by DFC( ) to compute the functions used to
             fit the data.

     W(*)    Double precision work array as used in DFC( ).  See DFC( )
             for the required length of W(*).  The contents of W(*)
             must not be modified by the user if the variance function
             is desired.

***REFERENCES  R. J. Hanson, Constrained least squares curve fitting
                 to discrete data using B-splines, a users guide,
                 Report SAND78-1291, Sandia Laboratories, December
                 1978.
***ROUTINES CALLED  DDOT, DFSPVN
***REVISION HISTORY  (YYMMDD)
   780801  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DDASSL

```
      SUBROUTINE DDASSL (RES, NEQ, T, Y, YPRIME, TOUT, INFO, RTOL, ATOL,
     *   IDID, RWORK, LRW, IWORK, LIW, RPAR, IPAR, JAC)
***BEGIN PROLOGUE  DDASSL
***PURPOSE  This code solves a system of differential/algebraic
            equations of the form G(T,Y,YPRIME) = 0.
***LIBRARY   SLATEC (DASSL)
***CATEGORY  I1A2
***TYPE      DOUBLE 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
      DOUBLE PRECISION T, Y(NEQ), YPRIME(NEQ), TOUT, RTOL, ATOL,
     *   RWORK(LRW), RPAR

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


 *Arguments:
  (In the following, all real arrays should be type DOUBLE PRECISION.)

  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 DDASSL are:
T, Y(*), YPRIME(*), INFO(1), RTOL, ATOL,
IDID, RWORK(*) AND IWORK(*)

*Description

Subroutine DDASSL 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 DDASSL.
2. Output after any return from DDASSL.
3. What to do to continue the integration.
4. Error messages.

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

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 DDASSL.
        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 DDASSL will try to solve the problem
        without getting IRES = -1. If IRES = -2, DDASSL
        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 DDASSL. 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 DDASSL 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) - DDASSL will perform much better if the matrix of
        partial derivatives, DG/DY + CJ*DG/DYPRIME,
        (here CJ is a scalar determined by DDASSL)
        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) --DDASSL 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 DDASSL 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 DDASSL and the RES subroutine (and the JAC subroutine). They are not altered by DDASSL. 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 DDASSL. 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 DDASSL. 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:

DDASSL uses a weighted norm DDANRM to measure the size of vectors such as the estimated error in each step. A FUNCTION subprogram

```
  DOUBLE PRECISION FUNCTION DDANRM(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 DDANRM routine has been included with DDASSL which computes the weighted root-mean-square norm given by

```
  DDANRM=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 DDANRM. This should, however, be attempted only after careful thought and consideration.

-------- OUTPUT -- AFTER ANY RETURN FROM DDASSL ----------

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 -- DDASSL 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 -- DDASSL 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 DDASSL. 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 DDASSL. 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.
               DDASSL cannot solve the problem as stated.
               It is possible that the redundant equations
               could be removed, and then DDASSL could
               solve the problem. It is also possible
               that a solution to your problem either
               does not exist or is not unique.

IDID = -9, DDASSL 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, DDASSL 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, DDASSL 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 DDASSL 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  D1MACH, DDAINI, DDANRM, DDASTP, DDATRP, DDAWTS,
                    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 DDASSL was replaced with a PARAMETER
           statement.  Also the statement S = 100.D0 was removed
           from the top of the Newton iteration in DDASTP.
           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 or DBLE.  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 DSQRT with SQRT everywhere.
              Replaced DABS with ABS everywhere.
              Replaced DMIN1 with MIN everywhere.
              Replaced MIN0 with MIN everywhere.
              Replaced DMAX1 with MAX everywhere.
              Replaced MAX0 with MAX everywhere.
              Replaced DSIGN with SIGN 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 DDASSL
                   and DDAINI routines.  DDAINI 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 DDASTP.
           900904  Three modifications were made to fix a bug (inside
                   DDASSL) 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 DDASTP to step
                       exactly to TSTOP under certain situations
                       (i.e. when H returned from DDASTP would otherwise
                       take TN beyond TSTOP).

                    2) Inside the DDASTP 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
```

DDAWS

```
      DOUBLE PRECISION FUNCTION DDAWS (X)
***BEGIN PROLOGUE  DDAWS
***PURPOSE  Compute Dawson's function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C8C
***TYPE      DOUBLE PRECISION (DAWS-S, DDAWS-D)
***KEYWORDS  DAWSON'S FUNCTION, FNLIB, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DDAWS(X) calculates the double precision Dawson's integral
 for double precision argument X.

 Series for DAW        on the interval  0.          to  1.00000E+00
                                        with weighted error   8.95E-32
                                         log weighted error  31.05
                                significant figures required  30.41
                                     decimal places required  31.71

 Series for DAW2       on the interval  0.          to  1.60000E+01
                                        with weighted error   1.61E-32
                                         log weighted error  31.79
                                significant figures required  31.40
                                     decimal places required  32.62

 Series for DAWA       on the interval  0.          to  6.25000E-02
                                        with weighted error   1.97E-32
                                         log weighted error  31.71
                                significant figures required  29.79
                                     decimal places required  32.64

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   780401  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)
   920618  Removed space from variable names.  (RWC, WRB)
   END PROLOGUE
```

DDEABM

```
      SUBROUTINE DDEABM (DF, NEQ, T, Y, TOUT, INFO, RTOL, ATOL, IDID,
     +   RWORK, LRW, IWORK, LIW, RPAR, IPAR)
***BEGIN PROLOGUE  DDEABM
***PURPOSE  Solve an initial value problem in ordinary differential
            equations using an Adams-Bashforth method.
***LIBRARY   SLATEC (DEPAC)
***CATEGORY  I1A1B
***TYPE      DOUBLE PRECISION (DEABM-S, DDEABM-D)
***KEYWORDS  ADAMS-BASHFORTH METHOD, DEPAC, INITIAL VALUE PROBLEMS,
             ODE, ORDINARY DIFFERENTIAL EQUATIONS, PREDICTOR-CORRECTOR
***AUTHOR  Shampine, L. F., (SNLA)
           Watts, H. A., (SNLA)
***DESCRIPTION

   This is the Adams code in the package of differential equation
   solvers DEPAC, consisting of the codes DDERKF, DDEABM, and DDEBDF.
   Design of the package was by L. F. Shampine and H. A. Watts.
   It is documented in
        SAND79-2374 , DEPAC - Design of a User Oriented Package of ODE
                              Solvers.
   DDEABM is a driver for a modification of the code ODE written by
             L. F. Shampine and M. K. Gordon
             Sandia Laboratories
             Albuquerque, New Mexico 87185

 ***********************************************************************
 * ABSTRACT *
 ************

   Subroutine DDEABM uses the Adams-Bashforth-Moulton
   Predictor-Corrector formulas of orders one through twelve to
   integrate a system of NEQ first order ordinary differential
   equations of the form
                         DU/DX = DF(X,U)
   when the vector Y(*) of initial values for U(*) at X=T is given.
   The subroutine integrates from T to TOUT. It is easy to continue the
   integration to get results at additional TOUT.  This is the interval
   mode of operation.  It is also easy for the routine to return with
   the solution at each intermediate step on the way to TOUT.  This is
   the intermediate-output mode of operation.

   DDEABM uses subprograms DDES, DSTEPS, DINTP, DHSTRT, DHVNRM,
   D1MACH, and the error handling routine XERMSG.  The only machine
   dependent parameters to be assigned appear in D1MACH.

 ***********************************************************************
 * Description of The Arguments To DDEABM (An Overview) *
 ***********************************************************************

   The Parameters are

      DF -- This is the name of a subroutine which you provide to
             define the differential equations.

      NEQ -- This is the number of (first order) differential
             equations to be integrated.
```

T -- This is a DOUBLE PRECISION value of the independent variable.

Y(*) -- This DOUBLE PRECISION array contains the solution components at T.

TOUT -- This is a DOUBLE PRECISION point at which a solution is desired.

INFO(*) -- The basic task of the code is to integrate the differential equations 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.

RTOL, ATOL -- These DOUBLE PRECISION 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.

IDID -- 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(*), LRW -- RWORK(*) is a DOUBLE PRECISION work array of length LRW which provides the code with needed storage space.

IWORK(*), LIW -- IWORK(*) is an INTEGER work array of length LIW which provides the code with needed storage space and an across call flag.

RPAR, IPAR -- These are DOUBLE PRECISION and INTEGER parameter arrays which you can use for communication between your calling program and the DF subroutine.

Quantities which are used as input items are
NEQ, T, Y(*), TOUT, INFO(*),
RTOL, ATOL, RWORK(1), LRW and LIW.

Quantities which may be altered by the code are
T, Y(*), INFO(1), RTOL, ATOL,
IDID, RWORK(*) and IWORK(*).

```
*********************************************************************
* INPUT -- What To Do On The First Call To DDEABM *
*********************************************************************
```

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.

DF -- Provide a subroutine of the form

```
DF(X,U,UPRIME,RPAR,IPAR)
```

to define the system of first order differential equations

which is to be solved. For the given values of X and the vector U(*)=(U(1),U(2),...,U(NEQ)) , the subroutine must evaluate the NEQ components of the system of differential equations DU/DX=DF(X,U) and store the derivatives in the array UPRIME(*), that is, UPRIME(I) = * DU(I)/DX * for equations I=1,...,NEQ.

Subroutine DF must NOT alter X or U(*). You must declare the name df in an external statement in your program that calls DDEABM. You must dimension U and UPRIME in DF.

RPAR and IPAR are DOUBLE PRECISION and INTEGER parameter arrays which you can use for communication between your calling program and subroutine DF. They are not used or altered by DDEABM. 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 DF 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. You must use a program variable for T because the code changes its value.

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

TOUT -- Set it to the first point at which a solution is desired. You can take TOUT = T, in which case the code will evaluate the derivative of the solution at T and return. 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 following each intermediate step (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 to accommodate other members of
       DEPAC or possible future extensions, though DDEABM uses
       only the first four 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 ****
```

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 program 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 Euclidean 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. Roughly speaking, they produce a solution Y(T) which satisfies the differential equations with a residual R(T), DY(T)/DT = DF(T,Y(T)) + R(T) , and, almost always, R(T) is bounded by the error tolerances. 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.D0 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 threshold ATOL.

Proper selection of the absolute error control parameters ATOL requires you to have some idea of the scale of the solution components. To acquire this information may mean that you will have to solve the problem more than once. In the absence of scale information, you should ask for some relative accuracy in all the components (by setting RTOL values non-zero) and perhaps impose extremely small absolute error tolerances to protect against the danger of a solution component becoming zero.

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 DOUBLE PRECISION work array of length LRW in your calling program.

```
RWORK(1) -- If you have set INFO(4)=0, you can ignore this
       optional input parameter.  Otherwise you must define a
       stopping point TSTOP by setting   RWORK(1) = TSTOP.
       (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.)

LRW -- Set it to the declared length of the RWORK array.
       You must have  LRW .GE. 130+21*NEQ

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. 51

RPAR, IPAR -- These are parameter arrays, of DOUBLE PRECISION and
       INTEGER type, respectively.  You can use them for
       communication between your program that calls DDEABM and
       the  DF subroutine.  They are not used or altered by
       DDEABM.  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 DF as arrays of appropriate length.

***********************************************************************
* OUTPUT -- After Any Return From DDEABM *
***********************************************************************

  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.
       You may also be interested in the approximate derivative
       of the solution at T.  It is contained in
       RWORK(21),...,RWORK(20+NEQ).

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 TOUT was successfully
                   completed (T=TOUT) by stepping exactly to TOUT.

       IDID = 3 -- The integration to TOUT was successfully
                   completed (T=TOUT) by stepping past TOUT.
                   Y(*) is obtained by interpolation.
```

```
                 *** Task Interrupted ***
          Reported by negative values of IDID

     IDID = -1 -- A large amount of work has been expended.
                  (500 steps attempted)

     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 = -4 -- The problem appears to be stiff.

     IDID = -5,-6,-7,..,-32  -- Not applicable for this code
                  but used by other members of DEPAC or possible
                  future extensions.

                 *** 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(11)--which contains the step size H to be
                  attempted on the next step.

       RWORK(12)--if the tolerances have been increased by the
                  code (IDID = -2) , they were multiplied by the
                  value in RWORK(12).

       RWORK(13)--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(20+I)--Which contains the approximate derivative
                  of the solution component Y(I).  In DDEABM, it
                  is obtained by calling subroutine DF to
                  evaluate the differential equation using T and
                  Y(*) when IDID=1 or 2, and by interpolation
                  when IDID=3.
```

```
*********************************************************************
* 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(*), RWORK(*), IWORK(*) or the differential equation in subroutine DF. 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.

The parameter INFO(1) is used by the code to indicate the beginning of a new problem and to indicate whether integration is to be continued. You must input the value INFO(1) = 0 when starting a new problem. You must input the value INFO(1) = 1 if you wish to continue after an interrupted task. Do not set INFO(1) = 0 on a continuation call unless you want the code to restart at the current T.

```
                    *** 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 reset INFO(1) = 1
 If
```

```
            IDID = -1, the code has attempted 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, the problem appears to be stiff.  It is very
                    inefficient to solve such problems with DDEABM.
                    The code DDEBDF in DEPAC handles this task
                    efficiently.  If you are absolutely sure you want
                    to continue with DDEABM, set INFO(1)=1 and call
                    the code again.

            IDID = -5,-6,-7,..,-32  --- cannot occur with this code
                    but used by other members of DEPAC or possible
                    future extensions.

                        *** 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.

***********************************************************************
*Long Description:

***********************************************************************
*             DEPAC Package Overview           *
***********************************************************************

....   You have a choice of three differential equation solvers from
....   DEPAC. The following brief descriptions are meant to aid you in
....   choosing the most appropriate code for your problem.

....   DDERKF is a fifth order Runge-Kutta code. It is the simplest of
....   the three choices, both algorithmically and in the use of the
....   code. DDERKF is primarily designed to solve non-stiff and
....   mildly stiff differential equations when derivative evaluations
....   are not expensive. It should generally not be used to get high
....   accuracy results nor answers at a great many specific points.
....   Because DDERKF has very low overhead costs, it will usually
....   result in the least expensive integration when solving
....   problems requiring a modest amount of accuracy and having
....   equations that are not costly to evaluate. DDERKF attempts to
....   discover when it is not suitable for the task posed.
```

```
 ....   DDEABM is a variable order (one through twelve) Adams code.
 ....   Its complexity lies somewhere between that of DDERKF and
 ....   DDEBDF.  DDEABM is primarily designed to solve non-stiff and
 ....   mildly stiff differential equations when derivative evaluations
 ....   are expensive, high accuracy results are needed or answers at
 ....   many specific points are required. DDEABM attempts to discover
 ....   when it is not suitable for the task posed.

 ....   DDEBDF is a variable order (one through five) backward
 ....   differentiation formula code. it is the most complicated of
 ....   the three choices. DDEBDF is primarily designed to solve stiff
 ....   differential equations at crude to moderate tolerances.
 ....   If the problem is very stiff at all, DDERKF and DDEABM will be
 ....   quite inefficient compared to DDEBDF. However, DDEBDF will be
 ....   inefficient compared to DDERKF and DDEABM on non-stiff problems
 ....   because it uses much more storage, has a much larger overhead,
 ....   and the low order formulas will not give high accuracies
 ....   efficiently.

 ....   The concept of stiffness cannot be described in a few words.
 ....   If you do not know the problem to be stiff, try either DDERKF
 ....   or DDEABM. Both of these codes will inform you of stiffness
 ....   when the cost of solving such problems becomes important.

 ***********************************************************************

***REFERENCES  L. F. Shampine and H. A. Watts, DEPAC - design of a user
                 oriented package of ODE solvers, Report SAND79-2374,
                 Sandia Laboratories, 1979.
***ROUTINES CALLED  DDES, XERMSG
***REVISION HISTORY  (YYMMDD)
   820301  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891024  Changed references from DVNORM to DHVNRM.  (WRB)
   891024  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
```

DDEBDF

```
      SUBROUTINE DDEBDF (DF, NEQ, T, Y, TOUT, INFO, RTOL, ATOL, IDID,
     +   RWORK, LRW, IWORK, LIW, RPAR, IPAR, DJAC)
***BEGIN PROLOGUE  DDEBDF
***PURPOSE  Solve an initial value problem in ordinary differential
            equations using backward differentiation formulas.  It is
            intended primarily for stiff problems.
***LIBRARY   SLATEC (DEPAC)
***CATEGORY  I1A2
***TYPE      DOUBLE PRECISION (DEBDF-S, DDEBDF-D)
***KEYWORDS  BACKWARD DIFFERENTIATION FORMULAS, DEPAC,
             INITIAL VALUE PROBLEMS, ODE,
             ORDINARY DIFFERENTIAL EQUATIONS, STIFF
***AUTHOR  Shampine, L. F., (SNLA)
           Watts, H. A., (SNLA)
***DESCRIPTION

   This is the backward differentiation code in the package of
   differential equation solvers DEPAC, consisting of the codes
   DDERKF, DDEABM, and DDEBDF.  Design of the package was by
   L. F. Shampine and H. A. Watts.  It is documented in
        SAND-79-2374 , DEPAC - Design of a User Oriented Package of ODE
                                Solvers.
   DDEBDF is a driver for a modification of the code LSODE written by
             A. C. Hindmarsh
             Lawrence Livermore Laboratory
             Livermore, California 94550

 ***********************************************************************
 **              DEPAC PACKAGE OVERVIEW           **
 ***********************************************************************

        You have a choice of three differential equation solvers from
        DEPAC.  The following brief descriptions are meant to aid you
        in choosing the most appropriate code for your problem.

        DDERKF is a fifth order Runge-Kutta code. It is the simplest of
        the three choices, both algorithmically and in the use of the
        code. DDERKF is primarily designed to solve non-stiff and mild-
        ly stiff differential equations when derivative evaluations are
        not expensive.  It should generally not be used to get high
        accuracy results nor answers at a great many specific points.
        Because DDERKF has very low overhead costs, it will usually
        result in the least expensive integration when solving
        problems requiring a modest amount of accuracy and having
        equations that are not costly to evaluate.  DDERKF attempts to
        discover when it is not suitable for the task posed.

        DDEABM is a variable order (one through twelve) Adams code. Its
        complexity lies somewhere between that of DDERKF and DDEBDF.
        DDEABM is primarily designed to solve non-stiff and mildly
        stiff differential equations when derivative evaluations are
        expensive, high accuracy results are needed or answers at
        many specific points are required.  DDEABM attempts to discover
        when it is not suitable for the task posed.

        DDEBDF is a variable order (one through five) backward
```

differentiation formula code. It is the most complicated of the three choices. DDEBDF is primarily designed to solve stiff differential equations at crude to moderate tolerances.
If the problem is very stiff at all, DDERKF and DDEABM will be quite inefficient compared to DDEBDF. However, DDEBDF will be inefficient compared to DDERKF and DDEABM on non-stiff problems because it uses much more storage, has a much larger overhead, and the low order formulas will not give high accuracies efficiently.

The concept of stiffness cannot be described in a few words. If you do not know the problem to be stiff, try either DDERKF or DDEABM. Both of these codes will inform you of stiffness when the cost of solving such problems becomes important.

****
** ABSTRACT **
****

Subroutine DDEBDF uses the backward differentiation formulas of orders one through five to integrate a system of NEQ first order ordinary differential equations of the form

DU/DX = DF(X,U)

when the vector Y(*) of initial values for U(*) at X=T is given. The subroutine integrates from T to TOUT. It is easy to continue the integration to get results at additional TOUT. This is the interval mode of operation. It is also easy for the routine to return with the solution at each intermediate step on the way to TOUT. This is the intermediate-output mode of operation.

****
* Description of The Arguments To DDEBDF (An Overview) *
****

The Parameters are:

DF -- This is the name of a subroutine which you provide to define the differential equations.

NEQ -- This is the number of (first order) differential equations to be integrated.

T -- This is a DOUBLE PRECISION value of the independent variable.

Y(*) -- This DOUBLE PRECISION array contains the solution components at T.

TOUT -- This is a DOUBLE PRECISION point at which a solution is desired.

INFO(*) -- The basic task of the code is to integrate the differential equations 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.

RTOL, ATOL -- These DOUBLE PRECISION 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.

IDID -- 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(*), LRW -- RWORK(*) is a DOUBLE PRECISION work array of length LRW which provides the code with needed storage space.

IWORK(*), LIW -- IWORK(*) is an INTEGER work array of length LIW which provides the code with needed storage space and an across call flag.

RPAR, IPAR -- These are DOUBLE PRECISION and INTEGER parameter arrays which you can use for communication between your calling program and the DF subroutine (and the DJAC subroutine).

DJAC -- This is the name of a subroutine which you may choose to provide for defining the Jacobian matrix of partial derivatives DF/DU.

Quantities which are used as input items are
NEQ, T, Y(*), TOUT, INFO(*),
RTOL, ATOL, RWORK(1), LRW,
IWORK(1), IWORK(2), and LIW.

Quantities which may be altered by the code are
T, Y(*), INFO(1), RTOL, ATOL,
IDID, RWORK(*) and IWORK(*).

```
**********************************************************************
* INPUT -- What To Do On The First Call To DDEBDF *
**********************************************************************
```

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.

DF -- Provide a subroutine of the form
DF(X,U,UPRIME,RPAR,IPAR)
to define the system of first order differential equations which is to be solved. For the given values of X and the vector U(*)=(U(1),U(2),...,U(NEQ)) , the subroutine must evaluate the NEQ components of the system of differential equations DU/DX=DF(X,U) and store the derivatives in the array UPRIME(*), that is, UPRIME(I) = * DU(I)/DX * for equations I=1,...,NEQ.

Subroutine DF must not alter X or U(*). You must declare the name DF in an external statement in your program that calls DDEBDF. You must dimension U and UPRIME in DF.

RPAR and IPAR are DOUBLE PRECISION and INTEGER parameter arrays which you can use for communication between your

calling program and subroutine DF. They are not used or altered by DDEBDF. 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 DF 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. You must use a program variable for T because the code changes its value.

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

TOUT -- Set it to the first point at which a solution is desired. You can take TOUT = T, in which case the code will evaluate the derivative of the solution at T and return. 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 following each intermediate step (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 to accommodate other members of DEPAC or possible future extensions, though DDEBDF uses only the first six 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 stiff problems it is necessary to use the
       Jacobian matrix of partial derivatives of the system
       of differential equations.  If you do not provide a
       subroutine to evaluate it analytically (see the
       description of the item DJAC 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 DJAC.  Sometimes numerical differencing
       is cheaper than evaluating derivatives in DJAC and
       sometimes it is not - this depends on your problem.
```

```
          If your problem is linear, i.e. has the form
          DU/DX = DF(X,U) = J(X)*U + G(X)   for some matrix J(X)
          and vector G(X), the Jacobian matrix  DF/DU = J(X).
          Since you must provide a subroutine to evaluate DF(X,U)
          analytically, it is little extra trouble to provide
          subroutine DJAC for evaluating J(X) analytically.
          Furthermore, in such cases, numerical differencing is
          much more expensive than analytic evaluation.

      **** 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 DJAC for evaluating the
                   Jacobian matrix ****

  INFO(6) -- DDEBDF will perform much better if the Jacobian
          matrix is banded and the code is told this.  In this
          case, the storage needed will be greatly reduced,
          numerical differencing will be performed more cheaply,
          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 Jacobian,
          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 Jacobian matrix has a banded structure and
          you want to provide subroutine DJAC to compute the
          partial derivatives, then you must be careful to store
          the elements of the Jacobian matrix in the special form
          indicated in the description of DJAC.

      **** Do you want to solve the problem using a full
           (dense) Jacobian 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 ****

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
       program 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. Roughly speaking, they produce a solution Y(T) which satisfies the differential equations with a residual R(T), DY(T)/DT = DF(T,Y(T)) + R(T) , and, almost always, R(T) is bounded by the error tolerances. 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 threshold ATOL.

Proper selection of the absolute error control parameters ATOL requires you to have some idea of the scale of the solution components. To acquire this information may mean that you will have to solve the problem more than once. In the absence of scale information, you should ask for some relative accuracy in all the components (by setting RTOL values non-zero) and perhaps impose extremely small absolute error tolerances to protect against the danger of a solution component becoming zero.

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 DOUBLE PRECISION work array of length LRW in your calling program.

RWORK(1) -- If you have set INFO(4)=0, you can ignore this optional input parameter. Otherwise you must define a stopping point TSTOP by setting RWORK(1) = TSTOP. (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.)

LRW -- Set it to the declared length of the RWORK array. You must have
```
     LRW .GE. 250+10*NEQ+NEQ**2
```

for the full (dense) Jacobian case (when INFO(6)=0), or
LRW .GE. 250+10*NEQ+(2*ML+MU+1)*NEQ
for the banded Jacobian case (when INFO(6)=1).

IWORK(*) -- Dimension this INTEGER work array of length LIW in your calling program.

IWORK(1), IWORK(2) -- If you have set INFO(6)=0, you can ignore these optional input parameters. Otherwise you must define the half-bandwidths ML (lower) and MU (upper) of the Jacobian matrix by setting IWORK(1) = ML and IWORK(2) = MU. (The code will work with a full matrix of NEQ**2 elements unless it is told that the problem has a banded Jacobian, in which case the code will work with a matrix containing at most (2*ML+MU+1)*NEQ elements.)

LIW -- Set it to the declared length of the IWORK array. You must have LIW .GE. 56+NEQ.

RPAR, IPAR -- These are parameter arrays, of DOUBLE PRECISION and INTEGER type, respectively. You can use them for communication between your program that calls DDEBDF and the DF subroutine (and the DJAC subroutine). They are not used or altered by DDEBDF. 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 DF (and in DJAC) as arrays of appropriate length.

DJAC -- If you have set INFO(5)=0, you can ignore this parameter by treating it as a dummy argument. (For some compilers you may have to write a dummy subroutine named DJAC in order to avoid problems associated with missing external routine names.) Otherwise, you must provide a subroutine of the form

```
DJAC(X,U,PD,NROWPD,RPAR,IPAR)
```

to define the Jacobian matrix of partial derivatives DF/DU of the system of differential equations DU/DX = DF(X,U). For the given values of X and the vector U(*)=(U(1),U(2),...,U(NEQ)), the subroutine must evaluate the non-zero partial derivatives DF(I)/DU(J) for each differential equation I=1,...,NEQ and each solution component J=1,...,NEQ , and store these values in the matrix PD. The elements of PD are set to zero before each call to DJAC so only non-zero elements need to be defined.

Subroutine DJAC must not alter X, U(*), or NROWPD. You must declare the name DJAC in an external statement in your program that calls DDEBDF. NROWPD is the row dimension of the PD matrix and is assigned by the code. Therefore you must dimension PD in DJAC according to

```
DIMENSION PD(NROWPD,1)
```

You must also dimension U in DJAC.

The way you must store the elements into the PD matrix depends on the structure of the Jacobian which you indicated by INFO(6).

*** INFO(6)=0 -- Full (Dense) Jacobian ***
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) = * DF(I)/DU(J) *
            *** INFO(6)=1 -- Banded Jacobian with ML Lower and MU
                Upper Diagonal Bands (refer to INFO(6) description of
                ML and MU) ***
                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) = * DF(I)/DU(J) *

            RPAR and IPAR are DOUBLE PRECISION and INTEGER parameter
            arrays which you can use for communication between your
            calling program and your Jacobian subroutine DJAC. They
            are not altered by DDEBDF. 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 DJAC as arrays of
            appropriate length.

**********************************************************************
* OUTPUT -- After any return from DDEBDF *
**********************************************************************

  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.
            You may also be interested in the approximate derivative
            of the solution at T.  It is contained in
            RWORK(21),...,RWORK(20+NEQ).

     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 TOUT was successfully
                        completed (T=TOUT) by stepping exactly to TOUT.

            IDID = 3 -- The integration to TOUT was successfully
                        completed (T=TOUT) by stepping past TOUT.
                        Y(*) is obtained by interpolation.

                          *** Task Interrupted ***
                     Reported by negative values of IDID

            IDID = -1 -- A large amount of work has been expended.
                        (500 steps attempted)
```

```
        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 = -4,-5  -- Not applicable for this code but used
                     by other members of DEPAC.

        IDID = -6 -- DDEBDF had repeated convergence test failures
                     on the last attempted step.

        IDID = -7 -- DDEBDF had repeated error test failures on
                     the last attempted step.

        IDID = -8,..,-32  -- Not applicable for this code but
                     used by other members of DEPAC or possible
                     future extensions.

                       *** 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(11)--which contains the step size H to be
                    attempted on the next step.

        RWORK(12)--If the tolerances have been increased by the
                    code (IDID = -2) , they were multiplied by the
                    value in RWORK(12).

        RWORK(13)--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(20+I)--which contains the approximate derivative
                    of the solution component Y(I).  In DDEBDF, it
                    is never obtained by calling subroutine DF to
                    evaluate the differential equation using T and
                    Y(*), except at the initial point of
```

integration.

```
*********************************************************************
** 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(*), RWORK(*), IWORK(*) or the differential equation in subroutine DF. 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.

The parameter INFO(1) is used by the code to indicate the beginning of a new problem and to indicate whether integration is to be continued. You must input the value INFO(1) = 0 when starting a new problem. You must input the value INFO(1) = 1 if you wish to continue after an interrupted task. Do not set INFO(1) = 0 on a continuation call unless you want the code to restart at the current T.

*** 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 reset INFO(1) = 1

If

IDID = -1, the code has attempted 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 but used by other members of DEPAC.

IDID = -6, repeated convergence test failures occurred on the last attempted step in DDEBDF. An inaccurate Jacobian may be the problem. If you are absolutely certain you want to continue, restart the integration at the current T by setting INFO(1)=0 and call the code again.

IDID = -7, repeated error test failures occurred on the last attempted step in DDEBDF. A singularity in the solution may be present. You should re-examine the problem being solved. If you are absolutely certain you want to continue, restart the integration at the current T by setting INFO(1)=0 and call the code again.

IDID = -8,..,-32 --- cannot occur with this code but used by other members of DDEPAC or possible future extensions.

*** 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.

****

***** Warning *****

If DDEBDF is to be used in an overlay situation, you must save and restore certain items used internally by DDEBDF (values in the common block DDEBD1). This can be accomplished as follows.

```
     To save the necessary values upon return from DDEBDF, simply call
        DSVCO(RWORK(22+NEQ),IWORK(21+NEQ)).

     To restore the necessary values before the next call to DDEBDF,
     simply call    DRSCO(RWORK(22+NEQ),IWORK(21+NEQ)).

***REFERENCES  L. F. Shampine and H. A. Watts, DEPAC - design of a user
                 oriented package of ODE solvers, Report SAND79-2374,
                 Sandia Laboratories, 1979.
***ROUTINES CALLED  DLSOD, XERMSG
***COMMON BLOCKS    DDEBD1
***REVISION HISTORY  (YYMMDD)
   820301  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   891024  Changed references from DVNORM to DHVNRM.  (WRB)
   891024  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   900510  Convert XERRWV calls to XERMSG calls, make Prologue comments
           consistent with DEBDF.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DDERKF

```
      SUBROUTINE DDERKF (DF, NEQ, T, Y, TOUT, INFO, RTOL, ATOL, IDID,
     +   RWORK, LRW, IWORK, LIW, RPAR, IPAR)
***BEGIN PROLOGUE  DDERKF
***PURPOSE  Solve an initial value problem in ordinary differential
            equations using a Runge-Kutta-Fehlberg scheme.
***LIBRARY   SLATEC (DEPAC)
***CATEGORY  I1A1A
***TYPE      DOUBLE PRECISION (DERKF-S, DDERKF-D)
***KEYWORDS  DEPAC, INITIAL VALUE PROBLEMS, ODE,
             ORDINARY DIFFERENTIAL EQUATIONS, RKF,
             RUNGE-KUTTA-FEHLBERG METHODS
***AUTHOR  Watts, H. A., (SNLA)
           Shampine, L. F., (SNLA)
***DESCRIPTION

   This is the Runge-Kutta code in the package of differential equation
   solvers DEPAC, consisting of the codes DDERKF, DDEABM, and DDEBDF.
   Design of the package was by L. F. Shampine and H. A. Watts.
   It is documented in
        SAND-79-2374 , DEPAC - Design of a User Oriented Package of ODE
                                Solvers.
   DDERKF is a driver for a modification of the code RKF45 written by
             H. A. Watts and L. F. Shampine
             Sandia Laboratories
             Albuquerque, New Mexico 87185

 ***********************************************************************
 **             DDEPAC PACKAGE OVERVIEW           **
 ***********************************************************************

        You have a choice of three differential equation solvers from
        DDEPAC.  The following brief descriptions are meant to aid you
        in choosing the most appropriate code for your problem.

        DDERKF is a fifth order Runge-Kutta code. It is the simplest of
        the three choices, both algorithmically and in the use of the
        code. DDERKF is primarily designed to solve non-stiff and mild-
        ly stiff differential equations when derivative evaluations are
        not expensive.  It should generally not be used to get high
        accuracy results nor answers at a great many specific points.
        Because DDERKF has very low overhead costs, it will usually
        result in the least expensive integration when solving
        problems requiring a modest amount of accuracy and having
        equations that are not costly to evaluate.  DDERKF attempts to
        discover when it is not suitable for the task posed.

        DDEABM is a variable order (one through twelve) Adams code. Its
        complexity lies somewhere between that of DDERKF and DDEBDF.
        DDEABM is primarily designed to solve non-stiff and mildly
        stiff differential equations when derivative evaluations are
        expensive, high accuracy results are needed or answers at
        many specific points are required.  DDEABM attempts to discover
        when it is not suitable for the task posed.

        DDEBDF is a variable order (one through five) backward
        differentiation formula code.  It is the most complicated of
```

the three choices. DDEBDF is primarily designed to solve stiff differential equations at crude to moderate tolerances.
If the problem is very stiff at all, DDERKF and DDEABM will be quite inefficient compared to DDEBDF. However, DDEBDF will be inefficient compared to DDERKF and DDEABM on non-stiff problems because it uses much more storage, has a much larger overhead, and the low order formulas will not give high accuracies efficiently.

The concept of stiffness cannot be described in a few words. If you do not know the problem to be stiff, try either DDERKF or DDEABM. Both of these codes will inform you of stiffness when the cost of solving such problems becomes important.

****
** ABSTRACT **
****

Subroutine DDERKF uses a Runge-Kutta-Fehlberg (4,5) method to integrate a system of NEQ first order ordinary differential equations of the form

DU/DX = DF(X,U)

when the vector Y(*) of initial values for U(*) at X=T is given. The subroutine integrates from T to TOUT. It is easy to continue the integration to get results at additional TOUT. This is the interval mode of operation. It is also easy for the routine to return with the solution at each intermediate step on the way to TOUT. This is the intermediate-output mode of operation.

DDERKF uses subprograms DRKFS, DFEHL, DHSTRT, DHVNRM, D1MACH, and the error handling routine XERMSG. The only machine dependent parameters to be assigned appear in D1MACH.

****
** DESCRIPTION OF THE ARGUMENTS TO DDERKF (AN OVERVIEW) **
****

The Parameters are:

DF -- This is the name of a subroutine which you provide to define the differential equations.

NEQ -- This is the number of (first order) differential equations to be integrated.

T -- This is a DOUBLE PRECISION value of the independent variable.

Y(*) -- This DOUBLE PRECISION array contains the solution components at T.

TOUT -- This is a DOUBLE PRECISION point at which a solution is desired.

INFO(*) -- The basic task of the code is to integrate the differential equations 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.

RTOL, ATOL -- These DOUBLE PRECISION 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.

IDID -- 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(*), LRW -- RWORK(*) is a DOUBLE PRECISION work array of length LRW which provides the code with needed storage space.

IWORK(*), LIW -- IWORK(*) is an INTEGER work array of length LIW which provides the code with needed storage space and an across call flag.

RPAR, IPAR -- These are DOUBLE PRECISION and INTEGER parameter arrays which you can use for communication between your calling program and the DF subroutine.

Quantities which are used as input items are
NEQ, T, Y(*), TOUT, INFO(*),
RTOL, ATOL, LRW and LIW.

Quantities which may be altered by the code are
T, Y(*), INFO(1), RTOL, ATOL,
IDID, RWORK(*) and IWORK(*).

```
*********************************************************************
** INPUT -- What to do On The First Call To DDERKF **
*********************************************************************
```

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.

DF -- Provide a subroutine of the form

DF(X,U,UPRIME,RPAR,IPAR)

to define the system of first order differential equations which is to be solved. For the given values of X and the vector U(*)=(U(1),U(2),...,U(NEQ)) , the subroutine must evaluate the NEQ components of the system of differential equations DU/DX=DF(X,U) and store the derivatives in the array UPRIME(*), that is, UPRIME(I) = * DU(I)/DX * for equations I=1,...,NEQ.

Subroutine DF must not alter X or U(*). You must declare the name DF in an external statement in your program that calls DDERKF. You must dimension U and UPRIME in DF.

RPAR and IPAR are DOUBLE PRECISION and INTEGER parameter arrays which you can use for communication between your calling program and subroutine DF. They are not used or altered by DDERKF. 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 DF 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. You must use a program variable for T because the code changes its value.

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

TOUT -- Set it to the first point at which a solution is desired. You can take TOUT = T, in which case the code will evaluate the derivative of the solution at T and return. 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 following each intermediate step (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. Since DDERKF will never step past a TOUT point, you need only make sure that no TOUT lies beyond TSTOP.

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 to accommodate other members of DEPAC or possible future extensions, though DDERKF uses only the first three 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).
       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 is
       INEFFICIENT.  The code DDEABM in DEPAC handles this task
       more 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 ****

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
      program 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 maximum norm is used to measure
      the size of vectors, and the error test uses the average
      of the magnitude of the solution at the beginning and end
      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.  Roughly speaking, they produce a solution Y(T)
      which satisfies the differential equations with a
      residual R(T),    DY(T)/DT = DF(T,Y(T)) + R(T)   ,
      and, almost always, R(T) is bounded by the error
      tolerances.  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. yields 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 threshold ATOL.

Proper selection of the absolute error control parameters ATOL requires you to have some idea of the scale of the solution components. To acquire this information may mean that you will have to solve the problem more than once. In the absence of scale information, you should ask for some relative accuracy in all the components (by setting RTOL values non-zero) and perhaps impose extremely small absolute error tolerances to protect against the danger of a solution component becoming zero.

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. If you want relative accuracies smaller than about 10**(-8), you should not ordinarily use DDERKF. The code DDEABM in DEPAC obtains stringent accuracies more efficiently.

RWORK(*) -- Dimension this DOUBLE PRECISION 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. 33+7*NEQ

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. 34

RPAR, IPAR -- These are parameter arrays, of DOUBLE PRECISION and INTEGER type, respectively. You can use them for communication between your program that calls DDERKF and the DF subroutine. They are not used or altered by DDERKF. 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 DF as arrays of appropriate length.

```
**********************************************************************
** OUTPUT -- After any return from DDERKF **
**********************************************************************
```

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.
       You may also be interested in the approximate derivative
       of the solution at T.  It is contained in
       RWORK(21),...,RWORK(20+NEQ).

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 TOUT was successfully
                   completed (T=TOUT) by stepping exactly to TOUT.

                     *** Task Interrupted ***
              Reported by negative values of IDID

       IDID = -1 -- A large amount of work has been expended.
                   (500 steps attempted)

       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 = -4 -- The problem appears to be stiff.

       IDID = -5 -- DDERKF is being used very inefficiently
                   because the natural step size is being
                   restricted by too frequent output.

       IDID = -6,-7,..,-32  -- Not applicable for this code but
                   used by other members of DEPAC or possible
                   future extensions.

                     *** 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(11)--which contains the step size H to be
                        attempted on the next step.

             RWORK(12)--If the tolerances have been increased by the
                        code (IDID = -2) , they were multiplied by the
                        value in RWORK(12).

             RWORK(20+I)--which contains the approximate derivative
                        of the solution component Y(I).  In DDERKF, it
                        is always obtained by calling subroutine DF to
                        evaluate the differential equation using T and
                        Y(*).

***********************************************************************
** 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 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(*), RWORK(*), IWORK(*) or
       the differential equation in subroutine DF. 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.

       The parameter INFO(1) is used by the code to indicate the
       beginning of a new problem and to indicate whether integration
       is to be continued.  You must input the value  INFO(1) = 0
       when starting a new problem.  You must input the value
       INFO(1) = 1  if you wish to continue after an interrupted task.
       Do not set  INFO(1) = 0  on a continuation call unless you
       want the code to restart at the current T.

                         *** Following a Completed Task ***
        If
            IDID = 1, call the code again to continue the integration
                    another step in the direction of TOUT.
```

```
    IDID = 2, 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 reset INFO(1) = 1
If
    IDID = -1, the code has attempted 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, the problem appears to be stiff.  It is very
            inefficient to solve such problems with DDERKF.
            The code DDEBDF in DEPAC handles this task
            efficiently.  If you are absolutely sure you want
            to continue with DDERKF, set INFO(1)=1 and call
            the code again.

    IDID = -5, you are using DDERKF very inefficiently by
            choosing output points TOUT so close together that
            the step size is repeatedly forced to be rather
            smaller than necessary.  If you are willing to
            accept solutions at the steps chosen by the code,
            a good way to proceed is to use the intermediate
            output mode (setting INFO(3)=1).  If you must have
            solutions at so many specific TOUT points, the
            code DDEABM in DEPAC handles this task
            efficiently.  If you want to continue with DDERKF,
            set INFO(1)=1 and call the code again.

    IDID = -6,-7,..,-32  --- cannot occur with this code but
            used by other members of DEPAC or possible future
            extensions.

                *** 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.
```

****

```
*Long Description:

**********************************************************************
**              DEPAC Package Overview           **
**********************************************************************

....   You have a choice of three differential equation solvers from
....   DEPAC. The following brief descriptions are meant to aid you in
....   choosing the most appropriate code for your problem.

....   DDERKF is a fifth order Runge-Kutta code. It is the simplest of
....   the three choices, both algorithmically and in the use of the
....   code. DDERKF is primarily designed to solve non-stiff and
....   mildly stiff differential equations when derivative evaluations
....   are not expensive. It should generally not be used to get high
....   accuracy results nor answers at a great many specific points.
....   Because DDERKF has very low overhead costs, it will usually
....   result in the least expensive integration when solving
....   problems requiring a modest amount of accuracy and having
....   equations that are not costly to evaluate. DDERKF attempts to
....   discover when it is not suitable for the task posed.

....   DDEABM is a variable order (one through twelve) Adams code.
....   Its complexity lies somewhere between that of DDERKF and
....   DDEBDF.  DDEABM is primarily designed to solve non-stiff and
....   mildly stiff differential equations when derivative evaluations
....   are expensive, high accuracy results are needed or answers at
....   many specific points are required. DDEABM attempts to discover
....   when it is not suitable for the task posed.

....   DDEBDF is a variable order (one through five) backward
....   differentiation formula code. it is the most complicated of
....   the three choices. DDEBDF is primarily designed to solve stiff
....   differential equations at crude to moderate tolerances.
....   If the problem is very stiff at all, DDERKF and DDEABM will be
....   quite inefficient compared to DDEBDF. However, DDEBDF will be
....   inefficient compared to DDERKF and DDEABM on non-stiff problems
....   because it uses much more storage, has a much larger overhead,
....   and the low order formulas will not give high accuracies
....   efficiently.

....   The concept of stiffness cannot be described in a few words.
....   If you do not know the problem to be stiff, try either DDERKF
....   or DDEABM. Both of these codes will inform you of stiffness
....   when the cost of solving such problems becomes important.

*********************************************************************

***REFERENCES  L. F. Shampine and H. A. Watts, DEPAC - design of a user
                 oriented package of ODE solvers, Report SAND79-2374,
                 Sandia Laboratories, 1979.
               L. F. Shampine and H. A. Watts, Practical solution of
                 ordinary differential equations by Runge-Kutta
                 methods, Report SAND76-0585, Sandia Laboratories,
                 1976.
***ROUTINES CALLED  DRKFS, XERMSG
***REVISION HISTORY  (YYMMDD)
   820301  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   891024  Changed references from DVNORM to DHVNRM.  (WRB)
```

```
891024  REVISION DATE from Version 3.2
891214  Prologue converted to Version 4.0 format.  (BAB)
900510  Convert XERRWV calls to XERMSG calls, make Prologue comments
        consistent with DERKF.  (RWC)
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

DDOT

```
      DOUBLE PRECISION FUNCTION DDOT (N, DX, INCX, DY, INCY)
***BEGIN PROLOGUE  DDOT
***PURPOSE  Compute the inner product of two vectors.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A4
***TYPE      DOUBLE 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)
       DX  double precision vector with N elements
     INCX  storage spacing between elements of DX
       DY  double precision vector with N elements
     INCY  storage spacing between elements of DY

     --Output--
     DDOT  double precision dot product (zero if N .LE. 0)

     Returns the dot product of double precision DX and DY.
     DDOT = sum for I = 0 to N-1 of  DX(LX+I*INCX) * DY(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
```

DDRIV1

```
      SUBROUTINE DDRIV1 (N, T, Y, F, TOUT, MSTATE, EPS, LENW,
     8   IERFLG)
***BEGIN PROLOGUE  DDRIV1
***PURPOSE  The function of DDRIV1 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.  DDRIV1 uses double precision arithmetic.
***LIBRARY   SLATEC (SDRIVE)
***CATEGORY  I1A2, I1A1B
***TYPE      DOUBLE PRECISION (SDRIV1-S, DDRIV1-D, CDRIV1-C)
***KEYWORDS  DOUBLE PRECISION, GEAR'S METHOD, INITIAL VALUE PROBLEMS,
             ODE, ORDINARY DIFFERENTIAL EQUATIONS, SDRIVE, 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. DDRIV1 should be tried first for those routine problems with
         no more than 200 differential equations (DDRIV2 and DDRIV3
         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 DDRIV1 should not have to concern themselves
         with these details.

      B. DDRIV2 should be considered for those problems for which
         DDRIV1 is inadequate.  For example, DDRIV1 may have difficulty
         with problems having zero initial conditions and zero
         derivatives.  In this case DDRIV2, with an appropriate value
         of the parameter EWT, should perform more efficiently.  DDRIV2
         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. DDRIV3 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 .....

(REMEMBER--To run DDRIV1 correctly in double precision, ALL non-integer arguments in the call sequence, including arrays, MUST be declared double precision.)

The user should use parameter names in the call sequence of DDRIV1 for those quantities whose value may be altered by DDRIV1. 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)
      DOUBLE PRECISION 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 DDRIV1. 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 DDRIV1. However, if the user would like to abort the calculation, i.e., return control to the program which calls DDRIV1, he should set N to zero. DDRIV1 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 DDRIV1.

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 DDRIV1 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, DDRIV1 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 DDRIV1 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 DDRIV1 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 DDRIV1 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 DDRIV1 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 DDRIV1 increases as EPS decreases.

WORK
LENW = (Input)
WORK is an array of LENW double precision words used internally for temporary storage. The user must allocate space for this array in the calling program by a statement such as

```
DOUBLE PRECISION 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 DDRIV1.

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
      DOUBLE PRECISION ALFA, EPS, T, TOUT
C                                N is the number of equations
      PARAMETER(ALFA = 1.D0, N = 3, LENW = N*N + 11*N + 300)
      DOUBLE PRECISION WORK(LENW), Y(N+1)
C                                             Initial point
```

```
      T = 0.00001D0
C                                          Set initial conditions
      Y(1) = 10.D0
      Y(2) = 0.D0
      Y(3) = 10.D0
C                                                   Pass parameter
      Y(4) = ALFA
      TOUT = T
      MSTATE = 1
      EPS = .001D0
 10   CALL DDRIV1 (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.D0*TOUT
      IF (TOUT .LT. 50.D0) GO TO 10
      END

      SUBROUTINE F (N, T, Y, YDOT)
      DOUBLE PRECISION ALFA, T, Y(*), YDOT(*)
      ALFA = Y(N+1)
      YDOT(1) = 1.D0 + ALFA*(Y(2) - Y(1)) - Y(1)*Y(3)
      YDOT(2) = ALFA*(Y(1) - Y(2)) - Y(2)*Y(3)
      YDOT(3) = 1.D0 - 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 DDRIV3.

```
***REFERENCES  C. W. Gear, Numerical Initial Value Problems in
                 Ordinary Differential Equations, Prentice-Hall, 1971.
***ROUTINES CALLED  DDRIV3, XERMSG
***REVISION HISTORY  (YYMMDD)
   790601  DATE WRITTEN
   900329  Initial submission to SLATEC.
   END PROLOGUE
```

DDRIV2

```
      SUBROUTINE DDRIV2 (N, T, Y, F, TOUT, MSTATE, NROOT, EPS, EWT,
     8   MINT, WORK, LENW, IWORK, LENIW, G, IERFLG)
***BEGIN PROLOGUE  DDRIV2
***PURPOSE  The function of DDRIV2 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.  DDRIV2 uses double precision arithmetic.
***LIBRARY   SLATEC (SDRIVE)
***CATEGORY  I1A2, I1A1B
***TYPE      DOUBLE PRECISION (SDRIV2-S, DDRIV2-D, CDRIV2-C)
***KEYWORDS  DOUBLE PRECISION, GEAR'S METHOD, INITIAL VALUE PROBLEMS,
             ODE, ORDINARY DIFFERENTIAL EQUATIONS, SDRIVE, 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  ...................................................

        (REMEMBER--To run DDRIV2 correctly in double precision, ALL
        non-integer arguments in the call sequence, including
        arrays, MUST be declared double precision.)

    The user should use parameter names in the call sequence of DDRIV2
    for those quantities whose value may be altered by DDRIV2.  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)
                   DOUBLE PRECISION 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 DDRIV2. 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 DDRIV2. However, if the user would like to abort the calculation, i.e., return control to the program which calls DDRIV2, he should set N to zero. DDRIV2 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 DDRIV2.

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 DDRIV2 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, DDRIV2 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 DDRIV2 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 DDRIV2 again.
5. (Output) A root was found at a point less than TOUT. The user can continue the integration toward TOUT by

simply calling DDRIV2 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 DDRIV2 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 DDRIV2 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.) DDRIV2 attempts to find the value of T at which one of the equations changes sign. DDRIV2 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 DDRIV2 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 double precision words used
          internally for temporary storage.  The user must allocate
          space for this array in the calling program by a statement
          such as
                    DOUBLE PRECISION 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 DDRIV2.

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 DDRIV2.

G      = A double precision 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:
                DOUBLE PRECISION FUNCTION G (N, T, Y, IROOT)
                DOUBLE PRECISION 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 DDRIV2.  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  DDRIV2.  However, if the user would like
          to abort the calculation, i.e., return control to the
          program which calls DDRIV2, he should set N to zero.
          DDRIV2 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 DDRIV2.

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 DDRIV2 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 DDRIV2.

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 DDRIV2.

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
      DOUBLE PRECISION 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 DDRIV2 (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 DDRIV3, XERMSG
***REVISION HISTORY (YYMMDD)
790601 DATE WRITTEN
900329 Initial submission to SLATEC.
END PROLOGUE

DDRIV3

```
      SUBROUTINE DDRIV3 (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  DDRIV3
***PURPOSE  The function of DDRIV3 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.  DDRIV3
            uses double precision arithmetic.
***LIBRARY   SLATEC (SDRIVE)
***CATEGORY  I1A2, I1A1B
***TYPE      DOUBLE PRECISION (SDRIV3-S, DDRIV3-D, CDRIV3-C)
***KEYWORDS  DOUBLE PRECISION, GEAR'S METHOD, INITIAL VALUE PROBLEMS,
             ODE, ORDINARY DIFFERENTIAL EQUATIONS, SDRIVE, 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 DDRIV3 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, DDRIV3 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.
    DDRIV3 is to be called once for each output point of T.

  II.  PARAMETERS  ....................................................
       (REMEMBER--To run DDRIV3 correctly in double precision, ALL
       non-integer arguments in the call sequence, including
       arrays, MUST be declared double precision.)

    The user should use parameter names in the call sequence of DDRIV3
    for those quantities whose value may be altered by DDRIV3.  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)
      DOUBLE PRECISION 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 DDRIV3. 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 DDRIV3. However, if the user would like to abort the calculation, i.e., return control to the program which calls DDRIV3, he should set N to zero. DDRIV3 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 DDRIV3.

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, DDRIV3 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 DDRIV3 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 DDRIV3 again.

5 (Output) A root was found at a point less than TOUT. The user can continue the integration toward TOUT by simply calling DDRIV3 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 DDRIV3 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 DDRIV3 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 DDRIV3 will integrate past TOUT and interpolate the solution. This is the most efficient mode.

NTASK = 2 Means DDRIV3 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 DDRIV3 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.) DDRIV3 attempts to find the value of T at which one of the equations changes sign. DDRIV3 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 DDRIV3 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 double precision words used
         internally for temporary storage.  The user must allocate
         space for this array in the calling program by a statement
         such as
```

```
                    DOUBLE PRECISION 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
          DDRIV3.

     IMPL =   0            1                2             3
              -------------------------------------------------------
MITER =  0    (MXORD+4)*N   Not allowed      Not allowed   Not allowed
              + 2*NROOT
              + 250

        1,2   N*N +         2*N*N +          N*N +         N*(N + NDE)
              (MXORD+5)*N   (MXORD+5)*N      (MXORD+6)*N   + (MXORD+5)*N
              + 2*NROOT     + 2*NROOT        + 2*NROOT     + 2*NROOT
              + 250         + 250            + 250         + 250

         3    (MXORD+4)*N   (MXORD+4)*N      (MXORD+4)*N   (MXORD+4)*N
              + 2*NROOT     + 2*NROOT        + 2*NROOT     + 2*NROOT
              + 250         + 250            + 250         + 250

        4,5   (2*ML+MU+1)   2*(2*ML+MU+1)    (2*ML+MU+1)   (2*ML+MU+1)*
              *N +          *N +             *N +          (N+NDE) +
              (MXORD+5)*N   (MXORD+5)*N      (MXORD+6)*N   + (MXORD+5)*N
              + 2*NROOT     + 2*NROOT        + 2*NROOT     + 2*NROOT
              + 250         + 250            + 250         + 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 DDRIV3.

   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 DDRIV3, and we only ask the
          user to tell DDRIV3 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)
      DOUBLE PRECISION 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 DDRIV3. 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 DDRIV3. However, if the user would like to abort the calculation, i.e., return control to the program which calls DDRIV3, he should set N to zero. DDRIV3 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 DDRIV3.

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)
      DOUBLE PRECISION 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)
    DOUBLE PRECISION 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)
    DOUBLE PRECISION 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 DDRIV3. 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 DDRIV3. However, if the user would like to abort the calculation, i.e., return control to the program which calls DDRIV3, he should set N to zero. DDRIV3 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 DDRIV3.

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 DDRIV3.

G = A double precision 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:

```
      DOUBLE PRECISION FUNCTION G (N, T, Y, IROOT)
      DOUBLE PRECISION 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 DDRIV3. 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 DDRIV3. However, if the user would like to abort the calculation, i.e., return control to the program which calls DDRIV3, he should set N to zero. DDRIV3 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 DDRIV3.

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 DDRIV3 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 DDRIV3. 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)
      DOUBLE PRECISION 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 DDRIV3. However, if the user would like to abort the calculation, i.e., return control to the program which calls DDRIV3, he should set N to zero. DDRIV3 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 DDRIV3.

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:
DDNTP, DDZRO, DDSTP, DDNTL, DDPST, DDCOR, DDCST, DDPSC, and DDSCL;
DGEFA, DGESL, DGBFA, DGBSL, and DNRM2 (from LINPACK)
D1MACH (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 DDRIV3 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 DDRIV3.

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 DDRIV3. 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 DDRIV3. 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 DDRIV3 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.

```
      DOUBLE PRECISION DFDY(N,N), EPSJ, H, R, D1MACH,
     8     SAVE1(N), SAVE2(N), T, UROUND, Y(N), YJ, YWT(N)
      UROUND = D1MACH(4)
      EPSJ = SQRT(UROUND)
      DO 30 J = J1,J2
        R = EPSJ*MAX(ABS(YWT(J)), ABS(Y(J)))
        IF (R .EQ. 0.D0) 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 DDRIV3.

***REFERENCES C. W. Gear, Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall, 1971.

```
***ROUTINES CALLED  D1MACH, DDNTP, DDSTP, DDZRO, DGBFA, DGBSL, DGEFA,
                    DGESL, DNRM2, XERMSG
***REVISION HISTORY  (YYMMDD)
   790601  DATE WRITTEN
   900329  Initial submission to SLATEC.
   END PROLOGUE
```

DE1

```
      DOUBLE PRECISION FUNCTION DE1 (X)
***BEGIN PROLOGUE  DE1
***PURPOSE  Compute the exponential integral E1(X).
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C5
***TYPE      DOUBLE PRECISION (E1-S, DE1-D)
***KEYWORDS  E1 FUNCTION, EXPONENTIAL INTEGRAL, FNLIB,
             SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DE1 calculates the double precision exponential integral, E1(X), for
 positive double precision argument X and the Cauchy principal value
 for negative X.  If principal values are used everywhere, then, for
 all X,

    E1(X) = -Ei(-X)
 or
    Ei(X) = -E1(-X).


 Series for AE10       on the interval -3.12500E-02 to  0.
                                   with weighted error   4.62E-32
                                    log weighted error  31.34
                            significant figures required  29.70
                                 decimal places required  32.18


 Series for AE11       on the interval -1.25000E-01 to -3.12500E-02
                                   with weighted error   2.22E-32
                                    log weighted error  31.65
                            significant figures required  30.75
                                 decimal places required  32.54


 Series for AE12       on the interval -2.50000E-01 to -1.25000E-01
                                   with weighted error   5.19E-32
                                    log weighted error  31.28
                            significant figures required  30.82
                                 decimal places required  32.09


 Series for E11        on the interval -4.00000E+00 to -1.00000E+00
                                   with weighted error   8.49E-34
                                    log weighted error  33.07
                            significant figures required  34.13
                                 decimal places required  33.80


 Series for E12        on the interval -1.00000E+00 to  1.00000E+00
                                   with weighted error   8.08E-33
                                    log weighted error  32.09
                        approx significant figures required  30.4
                                 decimal places required  32.79
```

```
 Series for AE13       on the interval  2.50000E-01 to  1.00000E+00
                                       with weighted error   6.65E-32
                                        log weighted error  31.18
                               significant figures required  30.69
                                    decimal places required  32.03


 Series for AE14       on the interval  0.          to  2.50000E-01
                                       with weighted error   5.07E-32
                                        log weighted error  31.30
                               significant figures required  30.40
                                    decimal places required  32.20

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   891115  Modified prologue description.  (WRB)
   891115  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)
   920618  Removed space from variable names.  (RWC, WRB)
   END PROLOGUE
```

DEABM

```
      SUBROUTINE DEABM (F, NEQ, T, Y, TOUT, INFO, RTOL, ATOL, IDID,
     +   RWORK, LRW, IWORK, LIW, RPAR, IPAR)
***BEGIN PROLOGUE  DEABM
***PURPOSE  Solve an initial value problem in ordinary differential
            equations using an Adams-Bashforth method.
***LIBRARY   SLATEC (DEPAC)
***CATEGORY  I1A1B
***TYPE      SINGLE PRECISION (DEABM-S, DDEABM-D)
***KEYWORDS  ADAMS-BASHFORTH METHOD, DEPAC, INITIAL VALUE PROBLEMS,
             ODE, ORDINARY DIFFERENTIAL EQUATIONS, PREDICTOR-CORRECTOR
***AUTHOR  Shampine, L. F., (SNLA)
           Watts, H. A., (SNLA)
***DESCRIPTION

   This is the Adams code in the package of differential equation
   solvers DEPAC, consisting of the codes DERKF, DEABM, and DEBDF.
   Design of the package was by L. F. Shampine and H. A. Watts.
   It is documented in
        SAND79-2374 , DEPAC - Design of a User Oriented Package of ODE
                              Solvers.
   DEABM is a driver for a modification of the code ODE written by
             L. F. Shampine and M. K. Gordon
             Sandia Laboratories
             Albuquerque, New Mexico 87185

 ***********************************************************************
 **              DEPAC PACKAGE OVERVIEW           **
 ****************************************************

        You have a choice of three differential equation solvers from
        DEPAC.  The following brief descriptions are meant to aid you
        in choosing the most appropriate code for your problem.

        DERKF is a fifth order Runge-Kutta code.  It is the simplest of
        the three choices, both algorithmically and in the use of the
        code.  DERKF is primarily designed to solve non-stiff and mild-
        ly stiff differential equations when derivative evaluations are
        not expensive.  It should generally not be used to get high
        accuracy results nor answers at a great many specific points.
        Because DERKF has very low overhead costs, it will usually
        result in the least expensive integration when solving
        problems requiring a modest amount of accuracy and having
        equations that are not costly to evaluate.  DERKF attempts to
        discover when it is not suitable for the task posed.

        DEABM is a variable order (one through twelve) Adams code.
        Its complexity lies somewhere between that of DERKF and DEBDF.
        DEABM is primarily designed to solve non-stiff and mildly stiff
        differential equations when derivative evaluations are
        expensive, high accuracy results are needed or answers at
        many specific points are required.  DEABM attempts to discover
        when it is not suitable for the task posed.

        DEBDF is a variable order (one through five) backward
        differentiation formula code.  It is the most complicated of
        the three choices.  DEBDF is primarily designed to solve stiff
```

differential equations at crude to moderate tolerances. If the problem is very stiff at all, DERKF and DEABM will be quite inefficient compared to DEBDF. However, DEBDF will be inefficient compared to DERKF and DEABM on non-stiff problems because it uses much more storage, has a much larger overhead, and the low order formulas will not give high accuracies efficiently.

The concept of stiffness cannot be described in a few words. If you do not know the problem to be stiff, try either DERKF or DEABM. Both of these codes will inform you of stiffness when the cost of solving such problems becomes important.

****
** ABSTRACT **
****

Subroutine DEABM uses the Adams-Bashforth-Moulton predictor-corrector formulas of orders one through twelve to integrate a system of NEQ first order ordinary differential equations of the form

DU/DX = F(X,U)

when the vector Y(*) of initial values for U(*) at X=T is given. The subroutine integrates from T to TOUT. It is easy to continue the integration to get results at additional TOUT. This is the interval mode of operation. It is also easy for the routine to return with the solution at each intermediate step on the way to TOUT. This is the intermediate-output mode of operation.

DEABM uses subprograms DES, STEPS, SINTRP, HSTART, HVNRM, R1MACH and the error handling routine XERMSG. The only machine dependent parameters to be assigned appear in R1MACH.

****
** DESCRIPTION OF THE ARGUMENTS TO DEABM (AN OVERVIEW) **
****

The parameters are

F -- This is the name of a subroutine which you provide to define the differential equations.

NEQ -- This is the number of (first order) differential equations to be integrated.

T -- This is a value of the independent variable.

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

TOUT -- This is a point at which a solution is desired.

INFO(*) -- The basic task of the code is to integrate the differential equations 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.

RTOL, ATOL -- 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.

IDID -- 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(*), LRW -- RWORK(*) is a real work array of length LRW which provides the code with needed storage space.

IWORK(*), LIW -- IWORK(*) is an integer work array of length LIW which provides the code with needed storage space and an across call flag.

RPAR, IPAR -- These are real and integer parameter arrays which you can use for communication between your calling program and the F subroutine.

Quantities which are used as input items are
NEQ, T, Y(*), TOUT, INFO(*),
RTOL, ATOL, RWORK(1), LRW and LIW.

Quantities which may be altered by the code are
T, Y(*), INFO(1), RTOL, ATOL,
IDID, RWORK(*) and IWORK(*).

```
*********************************************************************
** INPUT -- WHAT TO DO ON THE FIRST CALL TO DEABM **
*****************************************************
```

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.

F -- Provide a subroutine of the form

```
F(X,U,UPRIME,RPAR,IPAR)
```

to define the system of first order differential equations which is to be solved. For the given values of X and the vector U(*)=(U(1),U(2),...,U(NEQ)) , the subroutine must evaluate the NEQ components of the system of differential equations DU/DX = F(X,U) and store the derivatives in array UPRIME(*), that is, UPRIME(I) = * DU(I)/DX * for equations I=1,...,NEQ.

Subroutine F must not alter X or U(*). You must declare the name F in an external statement in your program that calls DEABM. You must dimension U and UPRIME in F.

RPAR and IPAR are real and integer parameter arrays which you can use for communication between your calling program and subroutine F. They are not used or altered by DEABM. 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 F 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.
       You must use a program variable for T because the code
       changes its value.

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

TOUT -- Set it to the first point at which a solution
       is desired.  You can take TOUT = T, in which case the code
       will evaluate the derivative of the solution at T and
       return.  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 following
       each intermediate step (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 to accommodate other members of
       DEPAC or possible future extensions, though DEABM uses
       only the first four 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 ****

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
        program 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 Euclidean 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. Roughly speaking, they produce a solution Y(T) which satisfies the differential equations with a residual R(T), DY(T)/DT = F(T,Y(T)) + R(T) , and, almost always, R(T) is bounded by the error tolerances. 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=0.0 results in a pure relative error test on that component. Setting RTOL=0.0 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 threshold ATOL.

Proper selection of the absolute error control parameters ATOL requires you to have some idea of the scale of the solution components. To acquire this information may mean that you will have to solve the problem more than once. In the absence of scale information, you should ask for some relative accuracy in all the components (by setting RTOL values non-zero) and perhaps impose extremely small absolute error tolerances to protect against the danger of a solution component becoming zero.

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.

RWORK(1) -- If you have set INFO(4)=0, you can ignore this optional input parameter. Otherwise you must define a stopping point TSTOP by setting RWORK(1) = TSTOP. (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.)

LRW -- Set it to the declared length of the RWORK array. You must have LRW .GE. 130+21*NEQ

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. 51

RPAR, IPAR -- These are parameter arrays, of real and integer type, respectively. You can use them for communication between your program that calls DEABM and the F subroutine. They are not used or altered by DEABM. 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 F as
            arrays of appropriate length.

*********************************************************************
** OUTPUT -- AFTER ANY RETURN FROM DEABM **
*******************************************

  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.
            You may also be interested in the approximate derivative
            of the solution at T.  It is contained in
            RWORK(21),...,RWORK(20+NEQ).

     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 TOUT was successfully
                      completed (T=TOUT) by stepping exactly to TOUT.

            IDID = 3 -- The integration to TOUT was successfully
                      completed (T=TOUT) by stepping past TOUT.
                      Y(*) is obtained by interpolation.

                          *** Task Interrupted ***
                   reported by negative values of IDID

            IDID = -1 -- A large amount of work has been expended.
                      (500 steps attempted)

            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 = -4 -- The problem appears to be stiff.

            IDID = -5,-6,-7,..,-32  -- Not applicable for this code
                      but used by other members of DEPAC or possible
                      future extensions.

                          *** 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(11)--Which contains the step size H to be attempted on the next step.

RWORK(12)--If the tolerances have been increased by the code (IDID = -2) , they were multiplied by the value in RWORK(12).

RWORK(13)--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(20+I)--Which contains the approximate derivative of the solution component Y(I). In DEABM, it is obtained by calling subroutine F to evaluate the differential equation using T and Y(*) when IDID=1 or 2, and by interpolation when IDID=3.

```
*********************************************************************
** 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(*), RWORK(*), IWORK(*) or the differential equation in subroutine F. 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.

The parameter INFO(1) is used by the code to indicate the beginning of a new problem and to indicate whether integration is to be continued. You must input the value INFO(1) = 0 when starting a new problem. You must input the value INFO(1) = 1 if you wish to continue after an interrupted task. Do not set INFO(1) = 0 on a continuation call unless you want the code to restart at the current T.

*** 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 reset INFO(1) = 1

If

IDID = -1, the code has attempted 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, the problem appears to be stiff. It is very inefficient to solve such problems with DEABM. The code DEBDF in DEPAC handles this task efficiently.

```
                    If you are absolutely sure you want to continue
                    with DEABM, set INFO(1)=1 and call the code again.

            IDID = -5,-6,-7,..,-32  --- cannot occur with this code
                    but used by other members of DEPAC or possible
                    future extensions.

                        *** 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.

 ***********************************************************************

***REFERENCES  L. F. Shampine and H. A. Watts, DEPAC - design of a user
                oriented package of ODE solvers, Report SAND79-2374,
                Sandia Laboratories, 1979.
***ROUTINES CALLED  DES, XERMSG
***REVISION HISTORY  (YYMMDD)
   800501  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   891024  Changed references from VNORM to HVNRM.  (WRB)
   891024  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
```

DEBDF

```
      SUBROUTINE DEBDF (F, NEQ, T, Y, TOUT, INFO, RTOL, ATOL, IDID,
     +   RWORK, LRW, IWORK, LIW, RPAR, IPAR, JAC)
***BEGIN PROLOGUE  DEBDF
***PURPOSE  Solve an initial value problem in ordinary differential
            equations using backward differentiation formulas.  It is
            intended primarily for stiff problems.
***LIBRARY   SLATEC (DEPAC)
***CATEGORY  I1A2
***TYPE      SINGLE PRECISION (DEBDF-S, DDEBDF-D)
***KEYWORDS  BACKWARD DIFFERENTIATION FORMULAS, DEPAC,
             INITIAL VALUE PROBLEMS, ODE,
             ORDINARY DIFFERENTIAL EQUATIONS, STIFF
***AUTHOR  Shampine, L. F., (SNLA)
           Watts, H. A., (SNLA)
***DESCRIPTION

   This is the backward differentiation code in the package of
   differential equation solvers DEPAC, consisting of the codes
   DERKF, DEABM, and DEBDF.  Design of the package was by
   L. F. Shampine and H. A. Watts.  It is documented in
        SAND-79-2374 , DEPAC - Design of a User Oriented Package of ODE
                                Solvers.
   DEBDF is a driver for a modification of the code LSODE written by
             A. C. Hindmarsh
             Lawrence Livermore Laboratory
             Livermore, California 94550

 ***********************************************************************
 **              DEPAC PACKAGE OVERVIEW           **
 ***********************************************************************

        You have a choice of three differential equation solvers from
        DEPAC.  The following brief descriptions are meant to aid you
        in choosing the most appropriate code for your problem.

        DERKF is a fifth order Runge-Kutta code.  It is the simplest of
        the three choices, both algorithmically and in the use of the
        code.  DERKF is primarily designed to solve non-stiff and mild-
        ly stiff differential equations when derivative evaluations are
        not expensive.  It should generally not be used to get high
        accuracy results nor answers at a great many specific points.
        Because DERKF has very low overhead costs, it will usually
        result in the least expensive integration when solving
        problems requiring a modest amount of accuracy and having
        equations that are not costly to evaluate.  DERKF attempts to
        discover when it is not suitable for the task posed.

        DEABM is a variable order (one through twelve) Adams code.
        Its complexity lies somewhere between that of DERKF and DEBDF.
        DEABM is primarily designed to solve non-stiff and mildly
        stiff differential equations when derivative evaluations are
        expensive, high accuracy results are needed or answers at
        many specific points are required.  DEABM attempts to discover
        when it is not suitable for the task posed.

        DEBDF is a variable order (one through five) backward
```

differentiation formula code. It is the most complicated of the three choices. DEBDF is primarily designed to solve stiff differential equations at crude to moderate tolerances. If the problem is very stiff at all, DERKF and DEABM will be quite inefficient compared to DEBDF. However, DEBDF will be inefficient compared to DERKF and DEABM on non-stiff problems because it uses much more storage, has a much larger overhead, and the low order formulas will not give high accuracies efficiently.

The concept of stiffness cannot be described in a few words. If you do not know the problem to be stiff, try either DERKF or DEABM. Both of these codes will inform you of stiffness when the cost of solving such problems becomes important.

```
**********************************************************************
** ABSTRACT **
**********************************************************************
```


Subroutine DEBDF uses the backward differentiation formulas of orders one through five to integrate a system of NEQ first order ordinary differential equations of the form

DU/DX = F(X,U)

when the vector Y(*) of initial values for U(*) at X=T is given. The subroutine integrates from T to TOUT. It is easy to continue the integration to get results at additional TOUT. This is the interval mode of operation. It is also easy for the routine to return with The solution at each intermediate step on the way to TOUT. This is the intermediate-output mode of operation.


```
**********************************************************************
** DESCRIPTION OF THE ARGUMENTS TO DEBDF (AN OVERVIEW) **
**********************************************************************
```

The Parameters are:

F -- This is the name of a subroutine which you provide to define the differential equations.

NEQ -- This is the number of (first order) differential equations to be integrated.

T -- This is a value of the independent variable.

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

TOUT -- This is a point at which a solution is desired.

INFO(*) -- The basic task of the code is to integrate the differential equations 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.

RTOL, ATOL -- 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.

IDID -- 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(*), LRW -- RWORK(*) is a REAL work array of length LRW which provides the code with needed storage space.

IWORK(*), LIW -- IWORK(*) is an INTEGER work array of length LIW which provides the code with needed storage space and an across call flag.

RPAR, IPAR -- These are REAL and INTEGER parameter arrays which you can use for communication between your calling program and the F subroutine (and the JAC subroutine).

JAC -- This is the name of a subroutine which you may choose to provide for defining the Jacobian matrix of partial derivatives DF/DU.

Quantities which are used as input items are
NEQ, T, Y(*), TOUT, INFO(*),
RTOL, ATOL, RWORK(1), LRW,
IWORK(1), IWORK(2), and LIW.

Quantities which may be altered by the code are
T, Y(*), INFO(1), RTOL, ATOL,
IDID, RWORK(*) and IWORK(*).

```
**********************************************************************
* INPUT -- What To Do On The First Call To DEBDF *
**********************************************************************
```

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.

F -- provide a subroutine of the form

F(X,U,UPRIME,RPAR,IPAR)

to define the system of first order differential equations which is to be solved. For the given values of X and the vector U(*)=(U(1),U(2),...,U(NEQ)) , the subroutine must evaluate the NEQ components of the system of differential equations DU/DX=F(X,U) and store the derivatives in the array UPRIME(*), that is, UPRIME(I) = * DU(I)/DX * for equations I=1,...,NEQ.

Subroutine F must not alter X or U(*). You must declare the name F in an external statement in your program that calls DEBDF. You must dimension U and UPRIME in F.

RPAR and IPAR are REAL and INTEGER parameter arrays which you can use for communication between your calling program and subroutine F. They are not used or altered by DEBDF. 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 F 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. You must use a program variable for T because the code changes its value.

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

TOUT -- Set it to the first point at which a solution is desired. You can take TOUT = T, in which case the code will evaluate the derivative of the solution at T and return. 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 following each intermediate step (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 to accommodate other members of DEPAC or possible future extensions, though DEBDF uses only the first six 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 stiff problems it is necessary to use the
       Jacobian matrix of partial derivatives of the system
       of differential equations.  If you do not provide a
       subroutine to evaluate it analytically (see the
       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.

       If your problem is linear, i.e. has the form
       DU/DX = F(X,U) = J(X)*U + G(X)   for some matrix J(X)
       and vector G(X), the Jacobian matrix  DF/DU = J(X).
       Since you must provide a subroutine to evaluate F(X,U)
```

```
          analytically, it is little extra trouble to provide
          subroutine JAC for evaluating J(X) analytically.
          Furthermore, in such cases, numerical differencing is
          much more expensive than analytic evaluation.

       **** 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
                    Jacobian matrix ****

  INFO(6) -- DEBDF will perform much better if the Jacobian
          matrix is banded and the code is told this.  In this
          case, the storage needed will be greatly reduced,
          numerical differencing will be performed more cheaply,
          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 Jacobian,
          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 Jacobian matrix 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 Jacobian matrix in the special form
          indicated in the description of JAC.

       **** Do you want to solve the problem using a full
            (dense) Jacobian 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 ****

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
        program 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. Roughly speaking, they produce a solution Y(T) which satisfies the differential equations with a residual R(T), DY(T)/DT = F(T,Y(T)) + R(T) , and, almost always, R(T) is bounded by the error tolerances. 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 threshold ATOL.

Proper selection of the absolute error control parameters ATOL requires you to have some idea of the scale of the solution components. To acquire this information may mean that you will have to solve the problem more than once. In the absence of scale information, you should ask for some relative accuracy in all the components (by setting RTOL values non-zero) and perhaps impose extremely small absolute error tolerances to protect against the danger of a solution component becoming zero.

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.

RWORK(1) -- If you have set INFO(4)=0, you can ignore this optional input parameter. Otherwise you must define a stopping point TSTOP by setting RWORK(1) = TSTOP. (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.)

LRW -- Set it to the declared length of the RWORK array. You must have

```
     LRW .GE. 250+10*NEQ+NEQ**2
```

for the full (dense) Jacobian case (when INFO(6)=0), or

```
     LRW .GE. 250+10*NEQ+(2*ML+MU+1)*NEQ
```

for the banded Jacobian case (when INFO(6)=1).

IWORK(*) -- Dimension this INTEGER work array of length LIW in your calling program.

IWORK(1), IWORK(2) -- If you have set INFO(6)=0, you can ignore these optional input parameters. Otherwise you must define the half-bandwidths ML (lower) and MU (upper) of the Jacobian matrix by setting IWORK(1) = ML and IWORK(2) = MU. (The code will work with a full matrix of NEQ**2 elements unless it is told that the problem has a banded Jacobian, in which case the code will work with a matrix containing at most (2*ML+MU+1)*NEQ elements.)

LIW -- Set it to the declared length of the IWORK array. You must have LIW .GE. 56+NEQ.

RPAR, IPAR -- These are parameter arrays, of REAL and INTEGER type, respectively. You can use them for communication between your program that calls DEBDF and the F subroutine (and the JAC subroutine). They are not used or altered by DEBDF. 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 F (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. (For some compilers you may have to write a dummy subroutine named JAC in order to avoid problems associated with missing external routine names.) Otherwise, you must provide a subroutine of the form

```
JAC(X,U,PD,NROWPD,RPAR,IPAR)
```

to define the Jacobian matrix of partial derivatives DF/DU of the system of differential equations DU/DX = F(X,U). For the given values of X and the vector U(*)=(U(1),U(2),...,U(NEQ)), the subroutine must evaluate the non-zero partial derivatives DF(I)/DU(J) for each differential equation I=1,...,NEQ and each solution component J=1,...,NEQ , 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 X, U(*), or NROWPD. You must declare the name JAC in an EXTERNAL statement in your program that calls DEBDF. NROWPD is the row dimension of the PD matrix and is assigned by the code. Therefore you must dimension PD in JAC according to

```
DIMENSION PD(NROWPD,1)
```

You must also dimension U in JAC.

The way you must store the elements into the PD matrix depends on the structure of the Jacobian which you indicated by INFO(6).

*** INFO(6)=0 -- Full (Dense) Jacobian ***
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) = * DF(I)/DU(J) *
```

*** INFO(6)=1 -- Banded Jacobian with ML Lower and MU Upper Diagonal Bands (refer to INFO(6) description of

```
                ML and MU) ***
                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) = * DF(I)/DU(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 DEBDF.  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.

**********************************************************************
* OUTPUT -- After any return from DDEBDF *
**********************************************************************

  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.
            You may also be interested in the approximate derivative
            of the solution at T.  It is contained in
            RWORK(21),...,RWORK(20+NEQ).

     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 TOUT was successfully
                      completed (T=TOUT) by stepping exactly to TOUT.

            IDID = 3 -- The integration to TOUT was successfully
                      completed (T=TOUT) by stepping past TOUT.
                      Y(*) is obtained by interpolation.

                          *** Task Interrupted ***
                   Reported by negative values of IDID

            IDID = -1 -- A large amount of work has been expended.
                      (500 steps attempted)

            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 = -4,-5 -- Not applicable for this code but used by other members of DEPAC.

IDID = -6 -- DEBDF had repeated convergence test failures on the last attempted step.

IDID = -7 -- DEBDF had repeated error test failures on the last attempted step.

IDID = -8,..,-32 -- Not applicable for this code but used by other members of DEPAC or possible future extensions.

*** 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(11)--which contains the step size H to be attempted on the next step.

RWORK(12)--If the tolerances have been increased by the code (IDID = -2) , they were multiplied by the value in RWORK(12).

RWORK(13)--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(20+I)--which contains the approximate derivative of the solution component Y(I). In DEBDF, it is never obtained by calling subroutine F to evaluate the differential equation using T and Y(*), except at the initial point of integration.

****
** 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(*), RWORK(*), IWORK(*) or
       the differential equation in subroutine F.  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.

       The parameter INFO(1) is used by the code to indicate the
       beginning of a new problem and to indicate whether integration
       is to be continued.  You must input the value  INFO(1) = 0
       when starting a new problem.  You must input the value
       INFO(1) = 1  if you wish to continue after an interrupted task.
       Do not set  INFO(1) = 0  on a continuation call unless you
       want the code to restart at the current T.

                        *** 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 reset INFO(1) = 1
        If
```

```
            IDID = -1, the code has attempted 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 but used
                    by other members of DEPAC.

            IDID = -6, repeated convergence test failures occurred
                    on the last attempted step in DEBDF.  An inaccu-
                    rate Jacobian may be the problem.  If you are
                    absolutely certain you want to continue, restart
                    the integration at the current T by setting
                    INFO(1)=0 and call the code again.

            IDID = -7, repeated error test failures occurred on the
                    last attempted step in DEBDF.  A singularity in
                    the solution may be present.  You should re-
                    examine the problem being solved.  If you are
                    absolutely certain you want to continue, restart
                    the integration at the current T by setting
                    INFO(1)=0 and call the code again.

            IDID = -8,..,-32  --- cannot occur with this code but
                    used by other members of DEPAC or possible future
                    extensions.

                        *** 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.

**********************************************************************

        ***** Warning *****

    If DEBDF is to be used in an overlay situation, you must save and
    restore certain items used internally by DEBDF  (values in the
    common block DEBDF1).  This can be accomplished as follows.

    To save the necessary values upon return from DEBDF, simply call
       SVCO(RWORK(22+NEQ),IWORK(21+NEQ)).

    To restore the necessary values before the next call to DEBDF,
```

```
     simply call    RSCO(RWORK(22+NEQ),IWORK(21+NEQ)).

***REFERENCES  L. F. Shampine and H. A. Watts, DEPAC - design of a user
                 oriented package of ODE solvers, Report SAND79-2374,
                 Sandia Laboratories, 1979.
***ROUTINES CALLED  LSOD, XERMSG
***COMMON BLOCKS    DEBDF1
***REVISION HISTORY  (YYMMDD)
   800901  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   891024  Changed references from VNORM to HVNRM.  (WRB)
   891024  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   900510  Convert XERRWV calls to XERMSG calls, change Prologue
           comments to agree with DDEBDF.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DEFC

```
      SUBROUTINE DEFC (NDATA, XDATA, YDATA, SDDATA, NORD, NBKPT, BKPT,
     +   MDEIN, MDEOUT, COEFF, LW, W)
***BEGIN PROLOGUE  DEFC
***PURPOSE  Fit a piecewise polynomial curve to discrete data.
            The piecewise polynomials are represented as B-splines.
            The fitting is done in a weighted least squares sense.
***LIBRARY   SLATEC
***CATEGORY  K1A1A1, K1A2A, L8A3
***TYPE      DOUBLE PRECISION (EFC-S, DEFC-D)
***KEYWORDS  B-SPLINE, CONSTRAINED LEAST SQUARES, CURVE FITTING
***AUTHOR  Hanson, R. J., (SNLA)
***DESCRIPTION

      This subprogram fits a piecewise polynomial curve
      to discrete data.  The piecewise polynomials are
      represented as B-splines.
      The fitting is done in a weighted least squares sense.

      The data can be processed in groups of modest size.
      The size of the group is chosen by the user.  This feature
      may be necessary for purposes of using constrained curve fitting
      with subprogram DFC( ) on a very large data set.

      For a description of the B-splines and usage instructions to
      evaluate them, see

      C. W. de Boor, Package for Calculating with B-Splines.
                     SIAM J. Numer. Anal., p. 441, (June, 1977).

      For further discussion of (constrained) curve fitting using
      B-splines, see

      R. J. Hanson, Constrained Least Squares Curve Fitting
                    to Discrete Data Using B-Splines, a User's
                    Guide. Sandia Labs. Tech. Rept. SAND-78-1291,
                    December, (1978).

  Input.. All TYPE REAL variables are DOUBLE PRECISION
      NDATA,XDATA(*),
      YDATA(*),
      SDDATA(*)
                         The NDATA discrete (X,Y) pairs and the Y value
                         standard deviation or uncertainty, SD, are in
                         the respective arrays XDATA(*), YDATA(*), and
                         SDDATA(*).  No sorting of XDATA(*) is
                         required.  Any non-negative value of NDATA is
                         allowed.  A negative value of NDATA is an
                         error.  A zero value for any entry of
                         SDDATA(*) will weight that data point as 1.
                         Otherwise the weight of that data point is
                         the reciprocal of this entry.

      NORD,NBKPT,
      BKPT(*)
                         The NBKPT knots of the B-spline of order NORD
                         are in the array BKPT(*).  Normally the
```

problem data interval will be included between the limits BKPT(NORD) and BKPT(NBKPT-NORD+1). The additional end knots BKPT(I),I=1,..., NORD-1 and I=NBKPT-NORD+2,...,NBKPT, are required to compute the functions used to fit the data. No sorting of BKPT(*) is required. Internal to DEFC( ) the extreme end knots may be reduced and increased respectively to accommodate any data values that are exterior to the given knot values. The contents of BKPT(*) is not changed.

NORD must be in the range 1 .LE. NORD .LE. 20. The value of NBKPT must satisfy the condition NBKPT .GE. 2*NORD.
Other values are considered errors.

(The order of the spline is one more than the degree of the piecewise polynomial defined on each interval. This is consistent with the B-spline package convention. For example, NORD=4 when we are using piecewise cubics.)

MDEIN

An integer flag, with one of two possible values (1 or 2), that directs the subprogram action with regard to new data points provided by the user.

=1 The first time that DEFC( ) has been entered. There are NDATA points to process.

=2 This is another entry to DEFC(). The subprogram DEFC( ) has been entered with MDEIN=1 exactly once before for this problem. There are NDATA new additional points to merge and process with any previous points.
(When using DEFC( ) with MDEIN=2 it is important that the set of knots remain fixed at the same values for all entries to DEFC( ).)

LW

The amount of working storage actually allocated for the working array W(*). This quantity is compared with the actual amount of storage needed in DEFC( ). Insufficient storage allocated for W(*) is an error. This feature was included in DEFC because misreading the storage formula for W(*) might very well lead to subtle and hard-to-find programming bugs.

The length of the array W(*) must satisfy

```
LW .GE. (NBKPT-NORD+3)*(NORD+1)+
        (NBKPT+1)*(NORD+1)+
      2*MAX(NDATA,NBKPT)+NBKPT+NORD**2
```

Output.. All TYPE REAL variables are DOUBLE PRECISION

MDEOUT

An output flag that indicates the status

of the curve fit.

=-1 A usage error of DEFC( ) occurred. The offending condition is noted with the SLATEC library error processor, XERMSG( ). In case the working array W(*) is not long enough, the minimal acceptable length is printed.

=1 The B-spline coefficients for the fitted curve have been returned in array COEFF(*).

=2 Not enough data has been processed to determine the B-spline coefficients. The user has one of two options. Continue to process more data until a unique set of coefficients is obtained, or use the subprogram DFC( ) to obtain a specific set of coefficients. The user should read the usage instructions for DFC( ) for further details if this second option is chosen.

COEFF(*)

If the output value of MDEOUT=1, this array contains the unknowns obtained from the least squares fitting process. These N=NBKPT-NORD parameters are the B-spline coefficients. For MDEOUT=2, not enough data was processed to uniquely determine the B-spline coefficients. In this case, and also when MDEOUT=-1, all values of COEFF(*) are set to zero.

If the user is not satisfied with the fitted curve returned by DEFC( ), the constrained least squares curve fitting subprogram DFC( ) may be required. The work done within DEFC( ) to accumulate the data can be utilized by the user, if so desired. This involves saving the first (NBKPT-NORD+3)*(NORD+1) entries of W(*) and providing this data to DFC( ) with the "old problem" designation. The user should read the usage instructions for subprogram DFC( ) for further details.

Working Array.. All TYPE REAL variables are DOUBLE PRECISION

W(*)

This array is typed DOUBLE PRECISION. Its length is specified as an input parameter in LW as noted above. The contents of W(*) must not be modified by the user between calls to DEFC( ) with values of MDEIN=1,2,2,... . The first (NBKPT-NORD+3)*(NORD+1) entries of W(*) are acceptable as direct input to DFC( ) for an "old problem" only when MDEOUT=1 or 2.

Evaluating the Fitted Curve..

To evaluate derivative number IDER at XVAL, use the function subprogram DBVALU( ).

```
F = DBVALU(BKPT,COEFF,NBKPT-NORD,NORD,IDER,
           XVAL,INBV,WORKB)
```

```
                              The output of this subprogram will not be
                              defined unless an output value of MDEOUT=1
                              was obtained from DEFC( ), XVAL is in the data
                              interval, and IDER is nonnegative and .LT.
                              NORD.

                              The first time DBVALU( ) is called, INBV=1
                              must be specified.  This value of INBV is the
                              overwritten by DBVALU( ).  The array WORKB(*)
                              must be of length at least 3*NORD, and must
                              not be the same as the W(*) array used in the
                              call to DEFC( ).

                              DBVALU( ) expects the breakpoint array BKPT(*)
                              to be sorted.

***REFERENCES  R. J. Hanson, Constrained least squares curve fitting
                 to discrete data using B-splines, a users guide,
                 Report SAND78-1291, Sandia Laboratories, December
                 1978.
***ROUTINES CALLED  DEFCMN
***REVISION HISTORY  (YYMMDD)
   800801  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 Prologue comments to refer to XERMSG.  (RWC)
   900607  Editorial changes to Prologue to make Prologues for EFC,
           DEFC, FC, and DFC look as much the same as possible.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DEI

```
      DOUBLE PRECISION FUNCTION DEI (X)
***BEGIN PROLOGUE  DEI
***PURPOSE  Compute the exponential integral Ei(X).
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C5
***TYPE      DOUBLE PRECISION (EI-S, DEI-D)
***KEYWORDS  EI FUNCTION, EXPONENTIAL INTEGRAL, FNLIB,
             SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DEI calculates the double precision exponential integral, Ei(X), for
 positive double precision argument X and the Cauchy principal value
 for negative X.  If principal values are used everywhere, then, for
 all X,

    Ei(X) = -E1(-X)
 or
    E1(X) = -Ei(-X).

***REFERENCES  (NONE)
***ROUTINES CALLED  DE1
***REVISION HISTORY  (YYMMDD)
   770701  DATE WRITTEN
   891115  Modified prologue description.  (WRB)
   891115  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   END PROLOGUE
```

DERF

```
      DOUBLE PRECISION FUNCTION DERF (X)
***BEGIN PROLOGUE  DERF
***PURPOSE  Compute the error function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C8A, L5A1E
***TYPE      DOUBLE PRECISION (ERF-S, DERF-D)
***KEYWORDS  ERF, ERROR FUNCTION, FNLIB, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DERF(X) calculates the double precision error function for double
 precision argument X.

 Series for ERF        on the interval  0.          to  1.00000E+00
                                       with weighted error   1.28E-32
                                        log weighted error  31.89
                                 significant figures required  31.05
                                      decimal places required  32.55

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DCSEVL, DERFC, INITDS
***REVISION HISTORY  (YYMMDD)
   770701  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)
   900727  Added EXTERNAL statement.  (WRB)
   920618  Removed space from variable name.  (RWC, WRB)
   END PROLOGUE
```

DERFC

```
      DOUBLE PRECISION FUNCTION DERFC (X)
***BEGIN PROLOGUE  DERFC
***PURPOSE  Compute the complementary error function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C8A, L5A1E
***TYPE      DOUBLE PRECISION (ERFC-S, DERFC-D)
***KEYWORDS  COMPLEMENTARY ERROR FUNCTION, ERFC, FNLIB,
             SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DERFC(X) calculates the double precision complementary error function
 for double precision argument X.

 Series for ERF        on the interval  0.          to  1.00000E+00
                                        with weighted Error   1.28E-32
                                         log weighted Error  31.89
                                 significant figures required  31.05
                                      decimal places required  32.55

 Series for ERC2       on the interval  2.50000E-01 to  1.00000E+00
                                        with weighted Error   2.67E-32
                                         log weighted Error  31.57
                                 significant figures required  30.31
                                      decimal places required  32.42

 Series for ERFC       on the interval  0.          to  2.50000E-01
                                        with weighted error   1.53E-31
                                         log weighted error  30.82
                                 significant figures required  29.47
                                      decimal places required  31.70

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   920618  Removed space from variable names.  (RWC, WRB)
   END PROLOGUE
```

DERKF

```
      SUBROUTINE DERKF (F, NEQ, T, Y, TOUT, INFO, RTOL, ATOL, IDID,
     +   RWORK, LRW, IWORK, LIW, RPAR, IPAR)
***BEGIN PROLOGUE  DERKF
***PURPOSE  Solve an initial value problem in ordinary differential
            equations using a Runge-Kutta-Fehlberg scheme.
***LIBRARY   SLATEC (DEPAC)
***CATEGORY  I1A1A
***TYPE      SINGLE PRECISION (DERKF-S, DDERKF-D)
***KEYWORDS  DEPAC, INITIAL VALUE PROBLEMS, ODE,
             ORDINARY DIFFERENTIAL EQUATIONS, RKF,
             RUNGE-KUTTA-FEHLBERG METHODS
***AUTHOR  Watts, H. A., (SNLA)
           Shampine, L. F., (SNLA)
***DESCRIPTION

   This is the Runge-Kutta code in the package of differential equation
   solvers DEPAC, consisting of the codes DERKF, DEABM, and DEBDF.
   Design of the package was by L. F. Shampine and H. A. Watts.
   It is documented in
        SAND-79-2374 , DEPAC - Design of a User Oriented Package of ODE
                                Solvers.
   DERKF is a driver for a modification of the code RKF45 written by
             H. A. Watts and L. F. Shampine
             Sandia Laboratories
             Albuquerque, New Mexico 87185

 ***********************************************************************
 **              DEPAC PACKAGE OVERVIEW           **
 ***********************************************************************

        You have a choice of three differential equation solvers from
        DEPAC.  The following brief descriptions are meant to aid you
        in choosing the most appropriate code for your problem.

        DERKF is a fifth order Runge-Kutta code.  It is the simplest of
        the three choices, both algorithmically and in the use of the
        code.  DERKF is primarily designed to solve non-stiff and mild-
        ly stiff differential equations when derivative evaluations are
        not expensive.  It should generally not be used to get high
        accuracy results nor answers at a great many specific points.
        Because DERKF has very low overhead costs, it will usually
        result in the least expensive integration when solving
        problems requiring a modest amount of accuracy and having
        equations that are not costly to evaluate.  DERKF attempts to
        discover when it is not suitable for the task posed.

        DEABM is a variable order (one through twelve) Adams code.  Its
        complexity lies somewhere between that of DERKF and DEBDF.
        DEABM is primarily designed to solve non-stiff and mildly
        stiff differential equations when derivative evaluations are
        expensive, high accuracy results are needed or answers at
        many specific points are required.  DEABM attempts to discover
        when it is not suitable for the task posed.

        DEBDF is a variable order (one through five) backward
        differentiation formula code.  It is the most complicated of
```

the three choices. DEBDF is primarily designed to solve stiff differential equations at crude to moderate tolerances. If the problem is very stiff at all, DERKF and DEABM will be quite inefficient compared to DEBDF. However, DEBDF will be inefficient compared to DERKF and DEABM on non-stiff problems because it uses much more storage, has a much larger overhead, and the low order formulas will not give high accuracies efficiently.

The concept of stiffness cannot be described in a few words. If you do not know the problem to be stiff, try either DERKF or DEABM. Both of these codes will inform you of stiffness when the cost of solving such problems becomes important.

```
**********************************************************************
** ABSTRACT **
**********************************************************************
```


Subroutine DERKF uses a Runge-Kutta-Fehlberg (4,5) method to integrate a system of NEQ first order ordinary differential equations of the form

$$DU/DX = F(X,U)$$

when the vector Y(*) of initial values for U(*) at X=T is given. The subroutine integrates from T to TOUT. It is easy to continue the integration to get results at additional TOUT. This is the interval mode of operation. It is also easy for the routine to return with the solution at each intermediate step on the way to TOUT. This is the intermediate-output mode of operation.

DERKF uses subprograms DERKFS, DEFEHL, HSTART, HVNRM, R1MACH, and the error handling routine XERMSG. The only machine dependent parameters to be assigned appear in R1MACH.


```
**********************************************************************
** DESCRIPTION OF THE ARGUMENTS TO DERKF (AN OVERVIEW) **
**********************************************************************
```

The Parameters are:

F -- This is the name of a subroutine which you provide to define the differential equations.

NEQ -- This is the number of (first order) differential equations to be integrated.

T -- This is a value of the independent variable.

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

TOUT -- This is a point at which a solution is desired.

INFO(*) -- The basic task of the code is to integrate the differential equations 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.

RTOL, ATOL -- 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.

IDID -- 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(*), LRW -- RWORK(*) is a REAL work array of length LRW which provides the code with needed storage space.

IWORK(*), LIW -- IWORK(*) is an INTEGER work array of length LIW which provides the code with needed storage space and an across call flag.

RPAR, IPAR -- These are REAL and INTEGER parameter arrays which you can use for communication between your calling program and the F subroutine.

Quantities which are used as input items are
NEQ, T, Y(*), TOUT, INFO(*),
RTOL, ATOL, LRW and LIW.

Quantities which may be altered by the code are
T, Y(*), INFO(1), RTOL, ATOL,
IDID, RWORK(*) and IWORK(*).

```
*********************************************************************
** INPUT -- What to do On The First Call To DERKF **
*********************************************************************
```

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.

F -- Provide a subroutine of the form

```
F(X,U,UPRIME,RPAR,IPAR)
```

to define the system of first order differential equations which is to be solved. For the given values of X and the vector U(*)=(U(1),U(2),...,U(NEQ)) , the subroutine must evaluate the NEQ components of the system of differential equations DU/DX=F(X,U) and store the derivatives in the array UPRIME(*), that is, UPRIME(I) = * DU(I)/DX * for equations I=1,...,NEQ.

Subroutine F must not alter X or U(*). You must declare the name F in an external statement in your program that calls DERKF. You must dimension U and UPRIME in F.

RPAR and IPAR are REAL and INTEGER parameter arrays which you can use for communication between your calling program and subroutine F. They are not used or altered by DERKF. 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 F 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.
       You must use a program variable for T because the code
       changes its value.

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

TOUT -- Set it to the first point at which a solution
       is desired.  You can take TOUT = T, in which case the code
       will evaluate the derivative of the solution at T and
       return.  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 following
       each intermediate step (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.  Since DERKF will never step past a TOUT point,
       you need only make sure that no TOUT lies beyond TSTOP.

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 to accommodate other members of
       DEPAC or possible future extensions, though DERKF uses
       only the first three 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).
       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 is
       INEFFICIENT.  The code DEABM in DEPAC handles this task
       more 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 ****

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
       program 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 maximum norm is used to measure
       the size of vectors, and the error test uses the average
       of the magnitude of the solution at the beginning and end
       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.  Roughly speaking, they produce a solution Y(T)
       which satisfies the differential equations with a
       residual R(T),    DY(T)/DT = F(T,Y(T)) + R(T)   ,
       and, almost always, R(T) is bounded by the error
       tolerances.  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. yields 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 threshold ATOL.

Proper selection of the absolute error control parameters ATOL requires you to have some idea of the scale of the solution components. To acquire this information may mean that you will have to solve the problem more than once. In the absence of scale information, you should ask for some relative accuracy in all the components (by setting RTOL values non-zero) and perhaps impose extremely small absolute error tolerances to protect against the danger of a solution component becoming zero.

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. If you want relative accuracies smaller than about 10**(-8), you should not ordinarily use DERKF. The code DEABM in DEPAC obtains stringent accuracies more efficiently.

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. 33+7*NEQ

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. 34

RPAR, IPAR -- These are parameter arrays, of REAL and INTEGER type, respectively. You can use them for communication between your program that calls DERKF and the F subroutine. They are not used or altered by DERKF. 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 F as arrays of appropriate length.

****
** OUTPUT -- After any return from DERKF **
****

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.

You may also be interested in the approximate derivative of the solution at T. It is contained in RWORK(21),...,RWORK(20+NEQ).

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 TOUT was successfully completed (T=TOUT) by stepping exactly to TOUT.

*** Task Interrupted ***
Reported by negative values of IDID

IDID = -1 -- A large amount of work has been expended. (500 steps attempted)

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 = -4 -- The problem appears to be stiff.

IDID = -5 -- DERKF is being used very inefficiently because the natural step size is being restricted by too frequent output.

IDID = -6,-7,..,-32 -- Not applicable for this code but used by other members of DEPAC or possible future extensions.

*** 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(11)--which contains the step size H to be
                       attempted on the next step.

            RWORK(12)--If the tolerances have been increased by the
                       code (IDID = -2) , they were multiplied by the
                       value in RWORK(12).

            RWORK(20+I)--which contains the approximate derivative
                       of the solution component Y(I).  In DERKF, it
                       is always obtained by calling subroutine F to
                       evaluate the differential equation using T and
                       Y(*).

***********************************************************************
** 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 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(*), RWORK(*), IWORK(*) or
       the differential equation in subroutine F.  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.

       The parameter INFO(1) is used by the code to indicate the
       beginning of a new problem and to indicate whether integration
       is to be continued.  You must input the value  INFO(1) = 0
       when starting a new problem.  You must input the value
       INFO(1) = 1  if you wish to continue after an interrupted task.
       Do not set  INFO(1) = 0  on a continuation call unless you
       want the code to restart at the current T.

                        *** Following a Completed Task ***
        If
            IDID = 1, call the code again to continue the integration
                    another step in the direction of TOUT.

            IDID = 2, 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 reset INFO(1) = 1

If

IDID = -1, the code has attempted 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, the problem appears to be stiff. It is very inefficient to solve such problems with DERKF. Code DEBDF in DEPAC handles this task efficiently. If you are absolutely sure you want to continue with DERKF, set INFO(1)=1 and call the code again.

IDID = -5, you are using DERKF very inefficiently by choosing output points TOUT so close together that the step size is repeatedly forced to be rather smaller than necessary. If you are willing to accept solutions at the steps chosen by the code, a good way to proceed is to use the intermediate output mode (setting INFO(3)=1). If you must have solutions at so many specific TOUT points, the code DEABM in DEPAC handles this task efficiently. If you want to continue with DERKF, set INFO(1)=1 and call the code again.

IDID = -6,-7,..,-32 --- cannot occur with this code but used by other members of DEPAC or possible future extensions.

*** 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.

```
**********************************************************************
*Long Description:

**********************************************************************
**              DEPAC Package Overview            **
**********************************************************************

....   You have a choice of three differential equation solvers from
```

```
....   DEPAC. The following brief descriptions are meant to aid you in
....   choosing the most appropriate code for your problem.

....   DERKF is a fifth order Runge-Kutta code. It is the simplest of
....   the three choices, both algorithmically and in the use of the
....   code. DERKF is primarily designed to solve non-stiff and
....   mildly stiff differential equations when derivative evaluations
....   are not expensive. It should generally not be used to get high
....   accuracy results nor answers at a great many specific points.
....   Because DERKF has very low overhead costs, it will usually
....   result in the least expensive integration when solving
....   problems requiring a modest amount of accuracy and having
....   equations that are not costly to evaluate. DERKF attempts to
....   discover when it is not suitable for the task posed.

....   DEABM is a variable order (one through twelve) Adams code.
....   Its complexity lies somewhere between that of DERKF and
....   DEBDF.  DEABM is primarily designed to solve non-stiff and
....   mildly stiff differential equations when derivative evaluations
....   are expensive, high accuracy results are needed or answers at
....   many specific points are required. DEABM attempts to discover
....   when it is not suitable for the task posed.

....   DEBDF is a variable order (one through five) backward
....   differentiation formula code. it is the most complicated of
....   the three choices. DEBDF is primarily designed to solve stiff
....   differential equations at crude to moderate tolerances.
....   If the problem is very stiff at all, DERKF and DEABM will be
....   quite inefficient compared to DEBDF. However, DEBDF will be
....   inefficient compared to DERKF and DEABM on non-stiff problems
....   because it uses much more storage, has a much larger overhead,
....   and the low order formulas will not give high accuracies
....   efficiently.

....   The concept of stiffness cannot be described in a few words.
....   If you do not know the problem to be stiff, try either DERKF
....   or DEABM. Both of these codes will inform you of stiffness
....   when the cost of solving such problems becomes important.

 ***********************************************************************

***REFERENCES  L. F. Shampine and H. A. Watts, DEPAC - design of a user
                 oriented package of ODE solvers, Report SAND79-2374,
                 Sandia Laboratories, 1979.
               L. F. Shampine and H. A. Watts, Practical solution of
                 ordinary differential equations by Runge-Kutta
                 methods, Report SAND76-0585, Sandia Laboratories,
                 1976.
***ROUTINES CALLED  DERKFS, XERMSG
***REVISION HISTORY  (YYMMDD)
   800501  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   891024  Changed references from VNORM to HVNRM.  (WRB)
   891024  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900510  Convert XERRWV calls to XERMSG calls, change Prologue
           comments to agree with DDERKF.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DEXINT

```
      SUBROUTINE DEXINT (X, N, KODE, M, TOL, EN, NZ, IERR)
***BEGIN PROLOGUE  DEXINT
***PURPOSE  Compute an M member sequence of exponential integrals
            E(N+K,X), K=0,1,...,M-1 for N .GE. 1 and X .GE. 0.
***LIBRARY   SLATEC
***CATEGORY  C5
***TYPE      DOUBLE PRECISION (EXINT-S, DEXINT-D)
***KEYWORDS  EXPONENTIAL INTEGRAL, SPECIAL FUNCTIONS
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

          DEXINT computes M member sequences of exponential integrals
          E(N+K,X), K=0,1,...,M-1 for N .GE. 1 and X .GE. 0.  The
          exponential integral is defined by

          E(N,X)=integral on (1,infinity) of EXP(-XT)/T**N

          where X=0.0 and N=1 cannot occur simultaneously.  Formulas
          and notation are found in the NBS Handbook of Mathematical
          Functions (ref. 1).

          The power series is implemented for X .LE. XCUT and the
          confluent hypergeometric representation

                      E(A,X) = EXP(-X)*(X**(A-1))*U(A,A,X)

          is computed for X .GT. XCUT.  Since sequences are computed in
          a stable fashion by recurring away from X, A is selected as
          the integer closest to X within the constraint N .LE. A .LE.
          N+M-1.  For the U computation, A is further modified to be the
          nearest even integer.  Indices are carried forward or
          backward by the two term recursion relation

                      K*E(K+1,X) + X*E(K,X) = EXP(-X)

          once E(A,X) is computed.  The U function is computed by means
          of the backward recursive Miller algorithm applied to the
          three term contiguous relation for U(A+K,A,X), K=0,1,...
          This produces accurate ratios and determines U(A+K,A,X), and
          hence E(A,X), to within a multiplicative constant C.
          Another contiguous relation applied to C*U(A,A,X) and
          C*U(A+1,A,X) gets C*U(A+1,A+1,X), a quantity proportional to
          E(A+1,X).  The normalizing constant C is obtained from the
          two term recursion relation above with K=A.

          The maximum number of significant digits obtainable
          is the smaller of 14 and the number of digits carried in
          double precision arithmetic.

     Description of Arguments

          Input      * X and TOL are double precision *
            X        X .GT. 0.0 for N=1 and  X .GE. 0.0 for N .GE. 2
            N        order of the first member of the sequence, N .GE. 1
                     (X=0.0 and N=1 is an error)
            KODE     a selection parameter for scaled values
```

```
                KODE=1   returns        E(N+K,X), K=0,1,...,M-1.
                    =2   returns EXP(X)*E(N+K,X), K=0,1,...,M-1.
        M       number of exponential integrals in the sequence,
                M .GE. 1
        TOL     relative accuracy wanted, ETOL .LE. TOL .LE. 0.1
                ETOL is the larger of double precision unit
                roundoff = D1MACH(4) and 1.0D-18

      Output    * EN is a double precision vector *
        EN      a vector of dimension at least M containing values
                EN(K) = E(N+K-1,X) or EXP(X)*E(N+K-1,X), K=1,M
                depending on KODE
        NZ      underflow indicator
                NZ=0   a normal return
                NZ=M   X exceeds XLIM and an underflow occurs.
                       EN(K)=0.0D0 , K=1,M returned on KODE=1
        IERR    error flag
                IERR=0, normal return, computation completed
                IERR=1, input error,   no computation
                IERR=2, error,         no computation
                        algorithm termination condition not met

***REFERENCES  M. Abramowitz and I. A. Stegun, Handbook of
                 Mathematical Functions, NBS AMS Series 55, U.S. Dept.
                 of Commerce, 1955.
               D. E. Amos, Computation of exponential integrals, ACM
                 Transactions on Mathematical Software 6, (1980),
                 pp. 365-377 and pp. 420-428.
***ROUTINES CALLED  D1MACH, DPSIXN, I1MACH
***REVISION HISTORY  (YYMMDD)
   800501  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)
   910408  Updated the REFERENCES section.  (WRB)
   920207  Updated with code with a revision date of 880811 from
           D. Amos.  Included correction of argument list.  (WRB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DEXPRL

```
      DOUBLE PRECISION FUNCTION DEXPRL (X)
***BEGIN PROLOGUE  DEXPRL
***PURPOSE  Calculate the relative error exponential (EXP(X)-1)/X.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C4B
***TYPE      DOUBLE PRECISION (EXPREL-S, DEXPRL-D, CEXPRL-C)
***KEYWORDS  ELEMENTARY FUNCTIONS, EXPONENTIAL, FIRST ORDER, FNLIB
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 Evaluate  EXPREL(X) = (EXP(X) - 1.0) / X.   For small ABS(X) the
 Taylor series is used.  If X is negative the reflection formula
         EXPREL(X) = EXP(X) * EXPREL(ABS(X))
 may be used.  This reflection formula will be of use when the
 evaluation for small ABS(X) is done by Chebyshev series rather than
 Taylor series.

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH
***REVISION HISTORY  (YYMMDD)
   770801  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   END PROLOGUE
```

DFAC

```
      DOUBLE PRECISION FUNCTION DFAC (N)
***BEGIN PROLOGUE  DFAC
***PURPOSE  Compute the factorial function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C1
***TYPE      DOUBLE PRECISION (FAC-S, DFAC-D)
***KEYWORDS  FACTORIAL, FNLIB, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DFAC(N) calculates the double precision factorial for integer
 argument N.

***REFERENCES  (NONE)
***ROUTINES CALLED  D9LGMC, DGAMLM, XERMSG
***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)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   END PROLOGUE
```

DFC

```
      SUBROUTINE DFC (NDATA, XDATA, YDATA, SDDATA, NORD, NBKPT, BKPT,
     +   NCONST, XCONST, YCONST, NDERIV, MODE, COEFF, W, IW)
***BEGIN PROLOGUE  DFC
***PURPOSE  Fit a piecewise polynomial curve to discrete data.
            The piecewise polynomials are represented as B-splines.
            The fitting is done in a weighted least squares sense.
            Equality and inequality constraints can be imposed on the
            fitted curve.
***LIBRARY   SLATEC
***CATEGORY  K1A1A1, K1A2A, L8A3
***TYPE      DOUBLE PRECISION (FC-S, DFC-D)
***KEYWORDS  B-SPLINE, CONSTRAINED LEAST SQUARES, CURVE FITTING,
             WEIGHTED LEAST SQUARES
***AUTHOR  Hanson, R. J., (SNLA)
***DESCRIPTION

      This subprogram fits a piecewise polynomial curve
      to discrete data.  The piecewise polynomials are
      represented as B-splines.
      The fitting is done in a weighted least squares sense.
      Equality and inequality constraints can be imposed on the
      fitted curve.

      For a description of the B-splines and usage instructions to
      evaluate them, see

      C. W. de Boor, Package for Calculating with B-Splines.
                     SIAM J. Numer. Anal., p. 441, (June, 1977).

      For further documentation and discussion of constrained
      curve fitting using B-splines, see

      R. J. Hanson, Constrained Least Squares Curve Fitting
                    to Discrete Data Using B-Splines, a User's
                    Guide. Sandia Labs. Tech. Rept. SAND-78-1291,
                    December, (1978).

  Input.. All TYPE REAL variables are DOUBLE PRECISION
      NDATA,XDATA(*),
      YDATA(*),
      SDDATA(*)
                          The NDATA discrete (X,Y) pairs and the Y value
                          standard deviation or uncertainty, SD, are in
                          the respective arrays XDATA(*), YDATA(*), and
                          SDDATA(*).  No sorting of XDATA(*) is
                          required.  Any non-negative value of NDATA is
                          allowed.  A negative value of NDATA is an
                          error.  A zero value for any entry of
                          SDDATA(*) will weight that data point as 1.
                          Otherwise the weight of that data point is
                          the reciprocal of this entry.

      NORD,NBKPT,
      BKPT(*)
                          The NBKPT knots of the B-spline of order NORD
                          are in the array BKPT(*).  Normally the
```

problem data interval will be included between the limits BKPT(NORD) and BKPT(NBKPT-NORD+1). The additional end knots BKPT(I),I=1,..., NORD-1 and I=NBKPT-NORD+2,...,NBKPT, are required to compute the functions used to fit the data. No sorting of BKPT(*) is required. Internal to DFC( ) the extreme end knots may be reduced and increased respectively to accommodate any data values that are exterior to the given knot values. The contents of BKPT(*) is not changed.

NORD must be in the range 1 .LE. NORD .LE. 20. The value of NBKPT must satisfy the condition NBKPT .GE. 2*NORD.
Other values are considered errors.

(The order of the spline is one more than the degree of the piecewise polynomial defined on each interval. This is consistent with the B-spline package convention. For example, NORD=4 when we are using piecewise cubics.)

NCONST,XCONST(*),
YCONST(*),NDERIV(*)

The number of conditions that constrain the B-spline is NCONST. A constraint is specified by an (X,Y) pair in the arrays XCONST(*) and YCONST(*), and by the type of constraint and derivative value encoded in the array NDERIV(*). No sorting of XCONST(*) is required. The value of NDERIV(*) is determined as follows. Suppose the I-th constraint applies to the J-th derivative of the B-spline. (Any non-negative value of J < NORD is permitted. In particular the value J=0 refers to the B-spline itself.)
For this I-th constraint, set
XCONST(I)=X,
YCONST(I)=Y, and
NDERIV(I)=ITYPE+4*J, where

```
ITYPE = 0,      if (J-th deriv. at X) .LE. Y.
      = 1,      if (J-th deriv. at X) .GE. Y.
      = 2,      if (J-th deriv. at X) .EQ. Y.
      = 3,      if (J-th deriv. at X) .EQ.
                   (J-th deriv. at Y).
```

(A value of NDERIV(I)=-1 will cause this constraint to be ignored. This subprogram feature is often useful when temporarily suppressing a constraint while still retaining the source code of the calling program.)

MODE

An input flag that directs the least squares solution method used by DFC( ).

The variance function, referred to below, defines the square of the probable error of

the fitted curve at any point, XVAL.
This feature of DFC( ) allows one to use the square root of this variance function to determine a probable error band around the fitted curve.

=1 a new problem. No variance function.

=2 a new problem. Want variance function.

=3 an old problem. No variance function.

=4 an old problem. Want variance function.

Any value of MODE other than 1-4 is an error.

The user with a new problem can skip directly to the description of the input parameters IW(1), IW(2).

If the user correctly specifies the new or old problem status, the subprogram DFC( ) will perform more efficiently.
By an old problem it is meant that subprogram DFC( ) was last called with this same set of knots, data points and weights.

Another often useful deployment of this old problem designation can occur when one has previously obtained a Q-R orthogonal decomposition of the matrix resulting from B-spline fitting of data (without constraints) at the breakpoints BKPT(I), I=1,...,NBKPT.
For example, this matrix could be the result of sequential accumulation of the least squares equations for a very large data set.
The user writes this code in a manner convenient for the application. For the discussion here let

N=NBKPT-NORD, and K=N+3

Let us assume that an equivalent least squares system

RC=D

has been obtained. Here R is an N+1 by N matrix and D is a vector with N+1 components.
The last row of R is zero. The matrix R is upper triangular and banded. At most NORD of the diagonals are nonzero.
The contents of R and D can be copied to the working array W(*) as follows.

The I-th diagonal of R, which has N-I+1 elements, is copied to W(*) starting at

W((I-1)*K+1),

for I=1,...,NORD.
The vector D is copied to W(*) starting at

W(NORD*K+1)

The input value used for NDATA is arbitrary when an old problem is designated. Because of the feature of DFC( ) that checks the working storage array lengths, a value not exceeding NBKPT should be used. For example, use NDATA=0.

(The constraints or variance function request can change in each call to DFC( ).) A new problem is anything other than an old problem.

IW(1),IW(2)

The amounts of working storage actually allocated for the working arrays W(*) and IW(*). These quantities are compared with the actual amounts of storage needed in DFC( ). Insufficient storage allocated for either W(*) or IW(*) is an error. This feature was included in DFC( ) because misreading the storage formulas for W(*) and IW(*) might very well lead to subtle and hard-to-find programming bugs.

The length of W(*) must be at least

```
NB=(NBKPT-NORD+3)*(NORD+1)+
    2*MAX(NDATA,NBKPT)+NBKPT+NORD**2
```

Whenever possible the code uses banded matrix processors DBNDAC( ) and DBNDSL( ). These are utilized if there are no constraints, no variance function is required, and there is sufficient data to uniquely determine the B-spline coefficients. If the band processors cannot be used to determine the solution, then the constrained least squares code DLSEI is used. In this case the subprogram requires an additional block of storage in W(*). For the discussion here define the integers NEQCON and NINCON respectively as the number of equality (ITYPE=2,3) and inequality (ITYPE=0,1) constraints imposed on the fitted curve. Define

```
L=NBKPT-NORD+1
```

and note that

```
NCONST=NEQCON+NINCON.
```

When the subprogram DFC( ) uses DLSEI( ) the length of the working array W(*) must be at least

```
LW=NB+(L+NCONST)*L+
```

```
                                2*(NEQCON+L)+(NINCON+L)+(NINCON+2)*(L+6)

                          The length of the array IW(*) must be at least

                            IW1=NINCON+2*L

                          in any case.

Output.. All TYPE REAL variables are DOUBLE PRECISION
    MODE
                          An output flag that indicates the status
                          of the constrained curve fit.

                          =-1  a usage error of DFC( ) occurred.  The
                               offending condition is noted with the
                               SLATEC library error processor, XERMSG.
                               In case the working arrays W(*) or IW(*)
                               are not long enough, the minimal
                               acceptable length is printed.

                          = 0  successful constrained curve fit.

                          = 1  the requested equality constraints
                               are contradictory.

                          = 2  the requested inequality constraints
                               are contradictory.

                          = 3  both equality and inequality constraints
                               are contradictory.

    COEFF(*)
                          If the output value of MODE=0 or 1, this array
                          contains the unknowns obtained from the least
                          squares fitting process.  These N=NBKPT-NORD
                          parameters are the B-spline coefficients.
                          For MODE=1, the equality constraints are
                          contradictory.  To make the fitting process
                          more robust, the equality constraints are
                          satisfied in a least squares sense.  In this
                          case the array COEFF(*) contains B-spline
                          coefficients for this extended concept of a
                          solution.  If MODE=-1,2 or 3 on output, the
                          array COEFF(*) is undefined.

Working Arrays.. All Type REAL variables are DOUBLE PRECISION
    W(*),IW(*)
                          These arrays are respectively typed DOUBLE
                          PRECISION and INTEGER.
                          Their required lengths are specified as input
                          parameters in IW(1), IW(2) noted above.  The
                          contents of W(*) must not be modified by the
                          user if the variance function is desired.

Evaluating the
Variance Function..
                          To evaluate the variance function (assuming
                          that the uncertainties of the Y values were
                          provided to DFC( ) and an input value of
                          MODE=2 or 4 was used), use the function
```

subprogram DCV( )

```
VAR=DCV(XVAL,NDATA,NCONST,NORD,NBKPT,
        BKPT,W)
```

Here XVAL is the point where the variance is desired. The other arguments have the same meaning as in the usage of DFC( ).

For those users employing the old problem designation, let MDATA be the number of data points in the problem. (This may be different from NDATA if the old problem designation feature was used.) The value, VAR, should be multiplied by the quantity

```
DBLE(MAX(NDATA-N,1))/DBLE(MAX(MDATA-N,1))
```

The output of this subprogram is not defined if an input value of MODE=1 or 3 was used in FC( ) or if an output value of MODE=-1, 2, or 3 was obtained. The variance function, except for the scaling factor noted above, is given by

```
VAR=(transpose of B(XVAL))*C*B(XVAL)
```

The vector B(XVAL) is the B-spline basis function values at X=XVAL.
The covariance matrix, C, of the solution coefficients accounts only for the least squares equations and the explicitly stated equality constraints. This fact must be considered when interpreting the variance function from a data fitting problem that has inequality constraints on the fitted curve.

Evaluating the Fitted Curve..

To evaluate derivative number IDER at XVAL, use the function subprogram DBVALU( )

```
F = DBVALU(BKPT,COEFF,NBKPT-NORD,NORD,IDER,
           XVAL,INBV,WORKB)
```

The output of this subprogram will not be defined unless an output value of MODE=0 or 1 was obtained from DFC( ), XVAL is in the data interval, and IDER is nonnegative and .LT. NORD.

The first time DBVALU( ) is called, INBV=1 must be specified. This value of INBV is the overwritten by DBVALU( ). The array WORKB(*) must be of length at least 3*NORD, and must not be the same as the W(*) array used in the call to DFC( ).

DBVALU( ) expects the breakpoint array BKPT(*) to be sorted.

```
***REFERENCES  R. J. Hanson, Constrained least squares curve fitting
                 to discrete data using B-splines, a users guide,
                 Report SAND78-1291, Sandia Laboratories, December
                 1978.
***ROUTINES CALLED  DFCMN
***REVISION HISTORY  (YYMMDD)
   780801  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   900510  Convert references to XERRWV to references to XERMSG.  (RWC)
   900607  Editorial changes to Prologue to make Prologues for EFC,
           DEFC, FC, and DFC look as much the same as possible.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DFZERO

```
      SUBROUTINE DFZERO (F, B, C, R, RE, AE, IFLAG)
***BEGIN PROLOGUE  DFZERO
***PURPOSE  Search for a zero of a function F(X) in a given interval
            (B,C).  It is designed primarily for problems where F(B)
            and F(C) have opposite signs.
***LIBRARY   SLATEC
***CATEGORY  F1B
***TYPE      DOUBLE PRECISION (FZERO-S, DFZERO-D)
***KEYWORDS  BISECTION, NONLINEAR, ROOTS, ZEROS
***AUTHOR  Shampine, L. F., (SNLA)
           Watts, H. A., (SNLA)
***DESCRIPTION

      DFZERO searches for a zero of a DOUBLE PRECISION function F(X)
      between the given DOUBLE PRECISION values B and C until the width
      of the interval (B,C) has collapsed to within a tolerance
      specified by the stopping criterion,
         ABS(B-C) .LE. 2.*(RW*ABS(B)+AE).
      The method used is an efficient combination of bisection and the
      secant rule and is due to T. J. Dekker.

      Description Of Arguments

   F     :EXT   - Name of the DOUBLE PRECISION external function.  This
                  name must be in an EXTERNAL statement in the calling
                  program.  F must be a function of one DOUBLE
                  PRECISION argument.

   B     :INOUT - One end of the DOUBLE PRECISION interval (B,C).  The
                  value returned for B usually is the better
                  approximation to a zero of F.

   C     :INOUT - The other end of the DOUBLE PRECISION interval (B,C)

   R     :IN    - A (better) DOUBLE PRECISION guess of a zero of F
                  which could help in speeding up convergence.  If F(B)
                  and F(R) have opposite signs, a root will be found in
                  the interval (B,R);  if not, but F(R) and F(C) have
                  opposite signs, a root will be found in the interval
                  (R,C);  otherwise, the interval (B,C) will be
                  searched for a possible root.  When no better guess
                  is known, it is recommended that R be set to B or C,
                  since if R is not interior to the interval (B,C), it
                  will be ignored.

   RE    :IN    - Relative error used for RW in the stopping criterion.
                  If the requested RE is less than machine precision,
                  then RW is set to approximately machine precision.

   AE    :IN    - Absolute error used in the stopping criterion.  If
                  the given interval (B,C) contains the origin, then a
                  nonzero value should be chosen for AE.

   IFLAG :OUT   - A status code.  User must check IFLAG after each
                  call.  Control returns to the user from DFZERO in all
                  cases.
```

```
                 1  B is within the requested tolerance of a zero.
                    The interval (B,C) collapsed to the requested
                    tolerance, the function changes sign in (B,C), and
                    F(X) decreased in magnitude as (B,C) collapsed.

                 2  F(B) = 0.  However, the interval (B,C) may not have
                    collapsed to the requested tolerance.

                 3  B may be near a singular point of F(X).
                    The interval (B,C) collapsed to the requested tol-
                    erance and the function changes sign in (B,C), but
                    F(X) increased in magnitude as (B,C) collapsed, i.e.
                      ABS(F(B out)) .GT. MAX(ABS(F(B in)),ABS(F(C in)))

                 4  No change in sign of F(X) was found although the
                    interval (B,C) collapsed to the requested tolerance.
                    The user must examine this case and decide whether
                    B is near a local minimum of F(X), or B is near a
                    zero of even multiplicity, or neither of these.

                 5  Too many (.GT. 500) function evaluations used.

***REFERENCES  L. F. Shampine and H. A. Watts, FZERO, a root-solving
                 code, Report SC-TM-70-631, Sandia Laboratories,
                 September 1970.
               T. J. Dekker, Finding a zero by means of successive
                 linear interpolation, Constructive Aspects of the
                 Fundamental Theorem of Algebra, edited by B. Dejon
                 and P. Henrici, Wiley-Interscience, 1969.
***ROUTINES CALLED  D1MACH
***REVISION HISTORY  (YYMMDD)
   700901  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
```

DGAMI

```
      DOUBLE PRECISION FUNCTION DGAMI (A, X)
***BEGIN PROLOGUE  DGAMI
***PURPOSE  Evaluate the incomplete Gamma function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C7E
***TYPE      DOUBLE PRECISION (GAMI-S, DGAMI-D)
***KEYWORDS  FNLIB, INCOMPLETE GAMMA FUNCTION, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 Evaluate the incomplete gamma function defined by

 DGAMI = integral from T = 0 to X of EXP(-T) * T**(A-1.0) .

 DGAMI is evaluated for positive values of A and non-negative values
 of X.  A slight deterioration of 2 or 3 digits accuracy will occur
 when DGAMI is very large or very small, because logarithmic variables
 are used.  The function and both arguments are double precision.

***REFERENCES  (NONE)
***ROUTINES CALLED  DGAMIT, DLNGAM, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   END PROLOGUE
```

DGAMIC

```
      DOUBLE PRECISION FUNCTION DGAMIC (A, X)
***BEGIN PROLOGUE  DGAMIC
***PURPOSE  Calculate the complementary incomplete Gamma function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C7E
***TYPE      DOUBLE PRECISION (GAMIC-S, DGAMIC-D)
***KEYWORDS  COMPLEMENTARY INCOMPLETE GAMMA FUNCTION, FNLIB,
             SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

   Evaluate the complementary incomplete Gamma function

   DGAMIC = integral from X to infinity of EXP(-T) * T**(A-1.)  .

   DGAMIC is evaluated for arbitrary real values of A and for non-
   negative values of X (even though DGAMIC is defined for X .LT.
   0.0), except that for X = 0 and A .LE. 0.0, DGAMIC is undefined.

   DGAMIC, A, and X are DOUBLE PRECISION.

   A slight deterioration of 2 or 3 digits accuracy will occur when
   DGAMIC is very large or very small in absolute value, because log-
   arithmic variables are used.  Also, if the parameter A is very close
   to a negative INTEGER (but not a negative integer), there is a loss
   of accuracy, which is reported if the result is less than half
   machine precision.

***REFERENCES  W. Gautschi, A computational procedure for incomplete
                 gamma functions, ACM Transactions on Mathematical
                 Software 5, 4 (December 1979), pp. 466-481.
               W. Gautschi, Incomplete gamma functions, Algorithm 542,
                 ACM Transactions on Mathematical Software 5, 4
                 (December 1979), pp. 482-489.
***ROUTINES CALLED  D1MACH, D9GMIC, D9GMIT, D9LGIC, D9LGIT, DLGAMS,
                    DLNGAM, XERCLR, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   920528  DESCRIPTION and REFERENCES sections revised.  (WRB)
   END PROLOGUE
```

DGAMIT

```
      DOUBLE PRECISION FUNCTION DGAMIT (A, X)
***BEGIN PROLOGUE  DGAMIT
***PURPOSE  Calculate Tricomi's form of the incomplete Gamma function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C7E
***TYPE      DOUBLE PRECISION (GAMIT-S, DGAMIT-D)
***KEYWORDS  COMPLEMENTARY INCOMPLETE GAMMA FUNCTION, FNLIB,
             SPECIAL FUNCTIONS, TRICOMI
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

   Evaluate Tricomi's incomplete Gamma function defined by

   DGAMIT = X**(-A)/GAMMA(A) * integral from 0 to X of EXP(-T) *
              T**(A-1.)

   for A .GT. 0.0 and by analytic continuation for A .LE. 0.0.
   GAMMA(X) is the complete gamma function of X.

   DGAMIT is evaluated for arbitrary real values of A and for non-
   negative values of X (even though DGAMIT is defined for X .LT.
   0.0), except that for X = 0 and A .LE. 0.0, DGAMIT is infinite,
   which is a fatal error.

   The function and both arguments are DOUBLE PRECISION.

   A slight deterioration of 2 or 3 digits accuracy will occur when
   DGAMIT is very large or very small in absolute value, because log-
   arithmic variables are used.  Also, if the parameter  A  is very
   close to a negative integer (but not a negative integer), there is
   a loss of accuracy, which is reported if the result is less than
   half machine precision.

***REFERENCES  W. Gautschi, A computational procedure for incomplete
                 gamma functions, ACM Transactions on Mathematical
                 Software 5, 4 (December 1979), pp. 466-481.
               W. Gautschi, Incomplete gamma functions, Algorithm 542,
                 ACM Transactions on Mathematical Software 5, 4
                 (December 1979), pp. 482-489.
***ROUTINES CALLED  D1MACH, D9GMIT, D9LGIC, D9LGIT, DGAMR, DLGAMS,
                    DLNGAM, XERCLR, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   920528  DESCRIPTION and REFERENCES sections revised.  (WRB)
   END PROLOGUE
```

DGAMLM

```
      SUBROUTINE DGAMLM (XMIN, XMAX)
***BEGIN PROLOGUE  DGAMLM
***PURPOSE  Compute the minimum and maximum bounds for the argument in
            the Gamma function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C7A, R2
***TYPE      DOUBLE PRECISION (GAMLIM-S, DGAMLM-D)
***KEYWORDS  COMPLETE GAMMA FUNCTION, FNLIB, LIMITS, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 Calculate the minimum and maximum legal bounds for X in gamma(X).
 XMIN and XMAX are not the only bounds, but they are the only non-
 trivial ones to calculate.

             Output Arguments --
 XMIN   double precision minimum legal value of X in gamma(X).  Any
        smaller value of X might result in underflow.
 XMAX   double precision maximum legal value of X in gamma(X).  Any
        larger value of X might cause overflow.

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, XERMSG
***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)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   END PROLOGUE
```

DGAMMA

```
      DOUBLE PRECISION FUNCTION DGAMMA (X)
***BEGIN PROLOGUE  DGAMMA
***PURPOSE  Compute the complete Gamma function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C7A
***TYPE      DOUBLE PRECISION (GAMMA-S, DGAMMA-D, CGAMMA-C)
***KEYWORDS  COMPLETE GAMMA FUNCTION, FNLIB, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DGAMMA(X) calculates the double precision complete Gamma function
 for double precision argument X.

 Series for GAM        on the interval  0.          to  1.00000E+00
                                       with weighted error   5.79E-32
                                        log weighted error  31.24
                                 significant figures required  30.00
                                      decimal places required  32.05

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, D9LGMC, DCSEVL, DGAMLM, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770601  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  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)
   920618  Removed space from variable name.  (RWC, WRB)
   END PROLOGUE
```

DGAMR

```
      DOUBLE PRECISION FUNCTION DGAMR (X)
***BEGIN PROLOGUE  DGAMR
***PURPOSE  Compute the reciprocal of the Gamma function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C7A
***TYPE      DOUBLE PRECISION (GAMR-S, DGAMR-D, CGAMR-C)
***KEYWORDS  FNLIB, RECIPROCAL GAMMA FUNCTION, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DGAMR(X) calculates the double precision reciprocal of the
 complete Gamma function for double precision argument X.

***REFERENCES  (NONE)
***ROUTINES CALLED  DGAMMA, DLGAMS, XERCLR, XGETF, XSETF
***REVISION HISTORY  (YYMMDD)
   770701  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)
   900727  Added EXTERNAL statement.  (WRB)
   END PROLOGUE
```

DGAUS8

```
      SUBROUTINE DGAUS8 (FUN, A, B, ERR, ANS, IERR)
***BEGIN PROLOGUE  DGAUS8
***PURPOSE  Integrate a real function of one variable over a finite
            interval using an adaptive 8-point Legendre-Gauss
            algorithm.  Intended primarily for high accuracy
            integration or integration of smooth functions.
***LIBRARY   SLATEC
***CATEGORY  H2A1A1
***TYPE      DOUBLE PRECISION (GAUS8-S, DGAUS8-D)
***KEYWORDS  ADAPTIVE QUADRATURE, AUTOMATIC INTEGRATOR,
             GAUSS QUADRATURE, NUMERICAL INTEGRATION
***AUTHOR  Jones, R. E., (SNLA)
***DESCRIPTION

      Abstract  *** a DOUBLE PRECISION routine ***
         DGAUS8 integrates real functions of one variable over finite
         intervals using an adaptive 8-point Legendre-Gauss algorithm.
         DGAUS8 is intended primarily for high accuracy integration
         or integration of smooth functions.

         The maximum number of significant digits obtainable in ANS
         is the smaller of 18 and the number of digits carried in
         double precision arithmetic.

      Description of Arguments

         Input--* FUN, A, B, ERR are DOUBLE PRECISION *
         FUN - name of external function to be integrated.  This name
               must be in an EXTERNAL statement in the calling program.
               FUN must be a DOUBLE PRECISION function of one DOUBLE
               PRECISION argument.  The value of the argument to FUN
               is the variable of integration which ranges from A to B.
         A   - lower limit of integration
         B   - upper limit of integration (may be less than A)
         ERR - is a requested pseudorelative error tolerance.  Normally
               pick a value of ABS(ERR) so that DTOL .LT. ABS(ERR) .LE.
               1.0D-3 where DTOL is the larger of 1.0D-18 and the
               double precision unit roundoff D1MACH(4).  ANS will
               normally have no more error than ABS(ERR) times the
               integral of the absolute value of FUN(X).  Usually,
               smaller values of ERR yield more accuracy and require
               more function evaluations.

               A negative value for ERR causes an estimate of the
               absolute error in ANS to be returned in ERR.  Note that
               ERR must be a variable (not a constant) in this case.
               Note also that the user must reset the value of ERR
               before making any more calls that use the variable ERR.

         Output--* ERR,ANS are double precision *
         ERR - will be an estimate of the absolute error in ANS if the
               input value of ERR was negative.  (ERR is unchanged if
               the input value of ERR was non-negative.)  The estimated
               error is solely for information to the user and should
               not be used as a correction to the computed integral.
         ANS - computed value of integral
```

```
        IERR- a status code
            --Normal codes
               1 ANS most likely meets requested error tolerance,
                 or A=B.
              -1 A and B are too nearly equal to allow normal
                 integration.  ANS is set to zero.
            --Abnormal code
               2 ANS probably does not meet requested error tolerance.

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, I1MACH, XERMSG
***REVISION HISTORY  (YYMMDD)
   810223  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  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
```

DGBCO

```
      SUBROUTINE DGBCO (ABD, LDA, N, ML, MU, IPVT, RCOND, Z)
***BEGIN PROLOGUE  DGBCO
***PURPOSE  Factor a band matrix by Gaussian elimination and
            estimate the condition number of the matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A2
***TYPE      DOUBLE 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

      DGBCO factors a double precision band matrix by Gaussian
      elimination and estimates the condition of the matrix.

      If  RCOND  is not needed, DGBFA is slightly faster.
      To solve  A*X = B , follow DGBCO by DGBSL.
      To compute  INVERSE(A)*C , follow DGBCO by DGBSL.
      To compute  DETERMINANT(A) , follow DGBCO by DGBDI.

      On Entry

         ABD     DOUBLE PRECISION(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   DOUBLE PRECISION
        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       DOUBLE PRECISION(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  DASUM, DAXPY, DDOT, DGBFA, DSCAL
***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
```

DGBDI

```
      SUBROUTINE DGBDI (ABD, LDA, N, ML, MU, IPVT, DET)
***BEGIN PROLOGUE  DGBDI
***PURPOSE  Compute the determinant of a band matrix using the factors
            computed by DGBCO or DGBFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D3A2
***TYPE      DOUBLE PRECISION (SGBDI-S, DGBDI-D, CGBDI-C)
***KEYWORDS  BANDED, DETERMINANT, INVERSE, LINEAR ALGEBRA, LINPACK,
             MATRIX
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      DGBDI computes the determinant of a band matrix
      using the factors computed by DGBCO or DGBFA.
      If the inverse is needed, use DGBSL  N  times.

      On Entry

         ABD     DOUBLE PRECISION(LDA, N)
                 the output from DGBCO or DGBFA.

         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 DGBCO or DGBFA.

      On Return

         DET     DOUBLE PRECISION(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
   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
```

DGBFA

```
      SUBROUTINE DGBFA (ABD, LDA, N, ML, MU, IPVT, INFO)
***BEGIN PROLOGUE  DGBFA
***PURPOSE  Factor a band matrix using Gaussian elimination.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A2
***TYPE      DOUBLE PRECISION (SGBFA-S, DGBFA-D, CGBFA-C)
***KEYWORDS  BANDED, LINEAR ALGEBRA, LINPACK, MATRIX FACTORIZATION
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      DGBFA factors a double precision band matrix by elimination.

      DGBFA is usually called by DGBCO, but it can be called
      directly with a saving in time if  RCOND  is not needed.

      On Entry

         ABD     DOUBLE PRECISION(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 DGBSL will divide by zero if
                      called.  Use  RCOND  in DGBCO 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 DAXPY, DSCAL, IDAMAX
***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

DGBMV

```
      SUBROUTINE DGBMV (TRANS, M, N, KL, KU, ALPHA, A, LDA,
     +     X, INCX, BETA, Y, INCY)
***BEGIN PROLOGUE  DGBMV
***PURPOSE  Perform one of the matrix-vector operations.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      DOUBLE 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

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

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

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

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

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

INCY - INTEGER.
On entry, INCY specifies the increment for the 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
```

DGBSL

```
      SUBROUTINE DGBSL (ABD, LDA, N, ML, MU, IPVT, B, JOB)
***BEGIN PROLOGUE  DGBSL
***PURPOSE  Solve the real band system A*X=B or TRANS(A)*X=B using
            the factors computed by DGBCO or DGBFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A2
***TYPE      DOUBLE PRECISION (SGBSL-S, DGBSL-D, CGBSL-C)
***KEYWORDS  BANDED, LINEAR ALGEBRA, LINPACK, MATRIX, SOLVE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      DGBSL solves the double precision band system
      A * X = B  or  TRANS(A) * X = B
      using the factors computed by DGBCO or DGBFA.

      On Entry

         ABD     DOUBLE PRECISION(LDA, N)
                 the output from DGBCO or DGBFA.

         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 DGBCO or DGBFA.

         B       DOUBLE PRECISION(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 DGBCO has set RCOND .GT. 0.0
         or DGBFA has set INFO .EQ. 0 .
```

```
     To compute  INVERSE(A) * C  where  C  is a matrix
     with  P  columns
           CALL DGBCO(ABD,LDA,N,ML,MU,IPVT,RCOND,Z)
           IF (RCOND is too small) GO TO ...
           DO 10 J = 1, P
              CALL DGBSL(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  DAXPY, DDOT
***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
```

DGECO

```
      SUBROUTINE DGECO (A, LDA, N, IPVT, RCOND, Z)
***BEGIN PROLOGUE  DGECO
***PURPOSE  Factor a matrix using Gaussian elimination and estimate
            the condition number of the matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A1
***TYPE      DOUBLE 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

      DGECO factors a double precision matrix by Gaussian elimination
      and estimates the condition of the matrix.

      If  RCOND  is not needed, DGEFA is slightly faster.
      To solve  A*X = B , follow DGECO by DGESL.
      To compute  INVERSE(A)*C , follow DGECO by DGESL.
      To compute  DETERMINANT(A) , follow DGECO by DGEDI.
      To compute  INVERSE(A) , follow DGECO by DGEDI.

      On Entry

         A       DOUBLE PRECISION(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   DOUBLE PRECISION
                 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       DOUBLE PRECISION(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  DASUM, DAXPY, DDOT, DGEFA, DSCAL
***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
```

DGEDI

```
      SUBROUTINE DGEDI (A, LDA, N, IPVT, DET, WORK, JOB)
***BEGIN PROLOGUE  DGEDI
***PURPOSE  Compute the determinant and inverse of a matrix using the
            factors computed by DGECO or DGEFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D3A1, D2A1
***TYPE      DOUBLE PRECISION (SGEDI-S, DGEDI-D, CGEDI-C)
***KEYWORDS  DETERMINANT, INVERSE, LINEAR ALGEBRA, LINPACK, MATRIX
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      DGEDI computes the determinant and inverse of a matrix
      using the factors computed by DGECO or DGEFA.

      On Entry

         A       DOUBLE PRECISION(LDA, N)
                 the output from DGECO or DGEFA.

         LDA     INTEGER
                 the leading dimension of the array  A .

         N       INTEGER
                 the order of the matrix  A .

         IPVT    INTEGER(N)
                 the pivot vector from DGECO or DGEFA.

         WORK    DOUBLE PRECISION(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     DOUBLE PRECISION(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 DGECO has set RCOND .GT. 0.0 or DGEFA 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  DAXPY, DSCAL, DSWAP
***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
```

DGEFA

```
      SUBROUTINE DGEFA (A, LDA, N, IPVT, INFO)
***BEGIN PROLOGUE  DGEFA
***PURPOSE  Factor a matrix using Gaussian elimination.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A1
***TYPE      DOUBLE PRECISION (SGEFA-S, DGEFA-D, CGEFA-C)
***KEYWORDS  GENERAL MATRIX, LINEAR ALGEBRA, LINPACK,
             MATRIX FACTORIZATION
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      DGEFA factors a double precision matrix by Gaussian elimination.

      DGEFA is usually called by DGECO, but it can be called
      directly with a saving in time if  RCOND  is not needed.
      (Time for DGECO) = (1 + 9/N)*(Time for DGEFA) .

      On Entry

         A       DOUBLE PRECISION(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 DGESL or DGEDI will divide by zero
                      if called.  Use  RCOND  in DGECO 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  DAXPY, DSCAL, IDAMAX
***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
```

DGEFS

```
      SUBROUTINE DGEFS (A, LDA, N, V, ITASK, IND, WORK, IWORK)
***BEGIN PROLOGUE  DGEFS
***PURPOSE  Solve a general system of linear equations.
***LIBRARY   SLATEC
***CATEGORY  D2A1
***TYPE      DOUBLE 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 DGEFS solves a general NxN system of double
    precision linear equations using LINPACK subroutines DGECO
    and DGESL.  That is, if A is an NxN double precision matrix
    and if X and B are double precision N-vectors, then DGEFS
    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 DGEFS
    in this case.

  Argument Description ***

    A      DOUBLE PRECISION(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      DOUBLE PRECISION(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   DOUBLE PRECISION(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  D1MACH, DGECO, DGESL, XERMSG
***REVISION HISTORY  (YYMMDD)
   800326  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
```

DGEMM

```
      SUBROUTINE DGEMM (TRANSA, TRANSB, M, N, ALPHA, A, LDA,
     +     B, LDB, BETA, C, LDC)
***BEGIN PROLOGUE  DGEMM
***PURPOSE  Perform one of the matrix-matrix operations.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B6
***TYPE      DOUBLE 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

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

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

C - DOUBLE PRECISION array of DIMENSION (LDC, n).
Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry.
On exit, the array C is overwritten by the m by n matrix (alpha*op(A)*op(B) + beta*C).

LDC - INTEGER.

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

***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
```

DGEMV

```
      SUBROUTINE DGEMV (TRANS, M, N, ALPHA, A, LDA, X, INCX,
     +    BETA, Y, INCY)
***BEGIN PROLOGUE  DGEMV
***PURPOSE  Perform one of the matrix-vector operations.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      DOUBLE 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

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

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

  LDA    - INTEGER.
```

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

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

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

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

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

  INCY    - INTEGER.
            On entry, INCY specifies the increment for the 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
```

DGER

```
      SUBROUTINE DGER (M, N, ALPHA, X, INCX, Y, INCY, A, LDA)
***BEGIN PROLOGUE  DGER
***PURPOSE  Perform the rank 1 operation.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B4
***TYPE      DOUBLE PRECISION (DGER-D)
***KEYWORDS  LEVEL 2 BLAS, LINEAR ALGEBRA
***AUTHOR  Dongarra, J. J., (ANL)
           Du Croz, J., (NAG)
           Hammarling, S., (NAG)
           Hanson, R. J., (SNLA)
***DESCRIPTION

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

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

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

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

  INCY   - INTEGER.
           On entry, INCY specifies the increment for the elements of
```

```
            Y. INCY must not be zero.
            Unchanged on exit.

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

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

***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
```

DGESL

```
      SUBROUTINE DGESL (A, LDA, N, IPVT, B, JOB)
***BEGIN PROLOGUE  DGESL
***PURPOSE  Solve the real system A*X=B or TRANS(A)*X=B using the
            factors computed by DGECO or DGEFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A1
***TYPE      DOUBLE PRECISION (SGESL-S, DGESL-D, CGESL-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX, SOLVE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      DGESL solves the double precision system
      A * X = B  or  TRANS(A) * X = B
      using the factors computed by DGECO or DGEFA.

      On Entry

         A       DOUBLE PRECISION(LDA, N)
                 the output from DGECO or DGEFA.

         LDA     INTEGER
                 the leading dimension of the array  A .

         N       INTEGER
                 the order of the matrix  A .

         IPVT    INTEGER(N)
                 the pivot vector from DGECO or DGEFA.

         B       DOUBLE PRECISION(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 DGECO has set RCOND .GT. 0.0
         or DGEFA has set INFO .EQ. 0 .

      To compute  INVERSE(A) * C  where  C  is a matrix
      with  P  columns
            CALL DGECO(A,LDA,N,IPVT,RCOND,Z)
            IF (RCOND is too small) GO TO ...
            DO 10 J = 1, P
               CALL DGESL(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  DAXPY, DDOT
***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
```

DGLSS

```
      SUBROUTINE DGLSS (A, MDA, M, N, B, MDB, NB, RNORM, WORK, LW,
     +   IWORK, LIW, INFO)
***BEGIN PROLOGUE  DGLSS
***PURPOSE  Solve a linear least squares problems by performing a QR
            factorization of the input matrix using Householder
            transformations.  Emphasis is put on detecting possible
            rank deficiency.
***LIBRARY   SLATEC
***CATEGORY  D9, D5
***TYPE      DOUBLE PRECISION (SGLSS-S, DGLSS-D)
***KEYWORDS  LINEAR LEAST SQUARES, LQ FACTORIZATION, QR FACTORIZATION,
             UNDERDETERMINED LINEAR SYSTEMS
***AUTHOR  Manteuffel, T. A., (LANL)
***DESCRIPTION

      DGLSS 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.

      DGLSS assumes full machine precision in the data.
      If more control over the uncertainty in the data
      is desired, the codes DLLSIA and DULSIA are
      recommended.

      DGLSS 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 DGLSS(A,MDA,M,N,B,MDB,NB,RNORM,WORK,LW,IWORK,LIW,INFO)

      Input..All TYPE REAL variables are DOUBLE PRECISION

      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 DGLSS.

     Output..All TYPE REAL variables are DOUBLE PRECISION

     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  DLLSIA, DULSIA
***REVISION HISTORY  (YYMMDD)
   810801  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  REVISION DATE from Version 3.2
```

```
891214  Prologue converted to Version 4.0 format.  (BAB)
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

DGMRES

```
      SUBROUTINE DGMRES(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  DGMRES
***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      DOUBLE 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)
      DOUBLE PRECISION B(N), X(N), A(NELT), TOL, ERR, SB(N), SX(N)
      DOUBLE PRECISION RGWK(LRGW), RWORK(USER DEFINED)
      EXTERNAL  MATVEC, MSOLVE

      CALL DGMRES(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       Double Precision B(N).
         Right-hand side vector.
 X      :INOUT    Double Precision 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       Double Precision 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 double precision 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 ISDGMR (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 /DSLBLK/ 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 /DSLBLK/ 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    Double Precision.
        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      Double Precision.
        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 DGMRES 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       Double Precision 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       Double Precision 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    Double Precision RGWK(LRGW).
        Double Precision array used for workspace by DGMRES.
        On return, RGWK(1) = RHOL.  See IERR for definition of RHOL.
LRGW   :IN       Integer.
        Length of the double precision 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     Double Precision RWORK(USER DEFINED).
        Double Precision 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:
      DGMRES 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 double precision 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 DGMRES 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 DGMRES:

```
DPIGMR  Contains the main iteration loop for GMRES.
DORTH   Orthogonalizes a new vector against older basis vectors.
DHEQR   Computes a QR decomposition of a Hessenberg matrix.
DHELS   Solves a Hessenberg least-squares system, using QR
        factors.
DRLCAL  Computes the scaled residual RL.
DXLCAL  Computes the solution XL.
ISDGMR  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 DSDCG and DSICCG 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 double precision 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  D1MACH, DCOPY, DNRM2, DPIGMR
***REVISION HISTORY  (YYMMDD)
   890404  DATE WRITTEN
   890404  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 DSLBLK.  (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
```

DGTSL

```
      SUBROUTINE DGTSL (N, C, D, E, B, INFO)
***BEGIN PROLOGUE  DGTSL
***PURPOSE  Solve a tridiagonal linear system.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2A2A
***TYPE      DOUBLE PRECISION (SGTSL-S, DGTSL-D, CGTSL-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX, SOLVE, TRIDIAGONAL
***AUTHOR  Dongarra, J., (ANL)
***DESCRIPTION

      DGTSL 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       DOUBLE PRECISION(N)
                 is the subdiagonal of the tridiagonal matrix.
                 C(2) through C(N) should contain the subdiagonal.
                 On output C is destroyed.

         D       DOUBLE PRECISION(N)
                 is the diagonal of the tridiagonal matrix.
                 On output D is destroyed.

         E       DOUBLE PRECISION(N)
                 is the superdiagonal of the tridiagonal matrix.
                 E(1) through E(N-1) should contain the superdiagonal.
                 On output E is destroyed.

         B       DOUBLE PRECISION(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
   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
```

DHFTI

```
      SUBROUTINE DHFTI (A, MDA, M, N, B, MDB, NB, TAU, KRANK, RNORM, H,
     +   G, IP)
***BEGIN PROLOGUE  DHFTI
***PURPOSE  Solve a least squares problem for banded matrices using
            sequential accumulation of rows of the data matrix.
            Exactly one right-hand side vector is permitted.
***LIBRARY   SLATEC
***CATEGORY  D9
***TYPE      DOUBLE PRECISION (HFTI-S, DHFTI-D)
***KEYWORDS  CURVE FITTING, LEAST SQUARES
***AUTHOR  Lawson, C. L., (JPL)
           Hanson, R. J., (SNLA)
***DESCRIPTION

      DIMENSION A(MDA,N),(B(MDB,NB) or B(M)),RNORM(NB),H(N),G(N),IP(N)

      This subroutine solves a linear least squares problem or a set of
      linear least squares problems having the same matrix but different
      right-side vectors.  The problem data consists of an M by N matrix
      A, an M by NB matrix B, and an absolute tolerance parameter TAU
      whose usage is described below.  The NB column vectors of B
      represent right-side vectors for NB distinct linear least squares
      problems.

      This set of problems can also be written as the matrix least
      squares problem

                        AX = B,

      where X is the N by NB solution matrix.

      Note that if B is the M by M identity matrix, then X will be the
      pseudo-inverse of A.

      This subroutine first transforms the augmented matrix (A B) to a
      matrix (R C) using premultiplying Householder transformations with
      column interchanges.  All subdiagonal elements in the matrix R are
      zero and its diagonal elements satisfy

                        ABS(R(I,I)).GE.ABS(R(I+1,I+1)),

                        I = 1,...,L-1, where

                        L = MIN(M,N).

      The subroutine will compute an integer, KRANK, equal to the number
      of diagonal terms of R that exceed TAU in magnitude. Then a
      solution of minimum Euclidean length is computed using the first
      KRANK rows of (R C).

      To be specific we suggest that the user consider an easily
      computable matrix norm, such as, the maximum of all column sums of
      magnitudes.

      Now if the relative uncertainty of B is EPS, (norm of uncertainty/
      norm of B), it is suggested that TAU be set approximately equal to
```

EPS*(norm of A).

The user must dimension all arrays appearing in the call list.. A(MDA,N),(B(MDB,NB) or B(M)),RNORM(NB),H(N),G(N),IP(N). This permits the solution of a range of problems in the same array space.

The entire set of parameters for DHFTI are

INPUT.. All TYPE REAL variables are DOUBLE PRECISION

A(*,*),MDA,M,N	The array A(*,*) initially contains the M by N matrix A of the least squares problem AX = B. The first dimensioning parameter of the array A(*,*) is MDA, which must satisfy MDA.GE.M Either M.GE.N or M.LT.N is permitted. There is no restriction on the rank of A. The condition MDA.LT.M is considered an error.
B(*),MDB,NB	If NB = 0 the subroutine will perform the orthogonal decomposition but will make no references to the array B(*). If NB.GT.0 the array B(*) must initially contain the M by NB matrix B of the least squares problem AX = B. If NB.GE.2 the array B(*) must be doubly subscripted with first dimensioning parameter MDB.GE.MAX(M,N). If NB = 1 the array B(*) may be either doubly or singly subscripted. In the latter case the value of MDB is arbitrary but it should be set to some valid integer value such as MDB = M. The condition of NB.GT.1.AND.MDB.LT. MAX(M,N) is considered an error.
TAU	Absolute tolerance parameter provided by user for pseudorank determination.
H(*),G(*),IP(*)	Arrays of working space used by DHFTI.

OUTPUT.. All TYPE REAL variables are DOUBLE PRECISION

A(*,*)	The contents of the array A(*,*) will be modified by the subroutine. These contents are not generally required by the user.
B(*)	On return the array B(*) will contain the N by NB solution matrix X.
KRANK	Set by the subroutine to indicate the pseudorank of A.
RNORM(*)	On return, RNORM(J) will contain the Euclidean norm of the residual vector for the problem defined by the J-th column vector of the array B(*,*) for J = 1,...,NB.
H(*),G(*)	On return these arrays respectively contain elements of the pre- and post-multiplying Householder transformations used to compute

```
                        the minimum Euclidean length solution.

     IP(*)              Array in which the subroutine records indices
                        describing the permutation of column vectors.
                        The contents of arrays H(*),G(*) and IP(*)
                        are not generally required by the user.

***REFERENCES  C. L. Lawson and R. J. Hanson, Solving Least Squares
                 Problems, Prentice-Hall, Inc., 1974, Chapter 14.
***ROUTINES CALLED  D1MACH, DH12, XERMSG
***REVISION HISTORY  (YYMMDD)
   790101  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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)
   901005  Replace usage of DDIFF with usage of D1MACH.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DINTP

```
      SUBROUTINE DINTP (X, Y, XOUT, YOUT, YPOUT, NEQN, KOLD, PHI, IVC,
     +   IV, KGI, GI, ALPHA, OG, OW, OX, OY)
***BEGIN PROLOGUE  DINTP
***PURPOSE  Approximate the solution at XOUT by evaluating the
            polynomial computed in DSTEPS at XOUT.  Must be used in
            conjunction with DSTEPS.
***LIBRARY   SLATEC (DEPAC)
***CATEGORY  I1A1B
***TYPE      DOUBLE 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  DSTEPS  approximate the solution near  X
   by a polynomial.  Subroutine  DINTP  approximates the solution at
   XOUT  by evaluating the polynomial there.  Information defining this
   polynomial is passed from  DSTEPS  so  DINTP  cannot be used alone.

   Subroutine DSTEPS 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 DINTP --

   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  DSTEPS  and passed to
   DINTP  from that subroutine

   Output from  DINTP --

      YOUT(*) -- solution at  XOUT
      YPOUT(*) -- derivative of solution at  XOUT
   The remaining parameters are returned unaltered from their input
   values.  Integration with  DSTEPS  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
```

DINTRV

```
      SUBROUTINE DINTRV (XT, LXT, X, ILO, ILEFT, MFLAG)
***BEGIN PROLOGUE  DINTRV
***PURPOSE  Compute the largest integer ILEFT in 1 .LE. ILEFT .LE. LXT
            such that XT(ILEFT) .LE. X where XT(*) is a subdivision of
            the X interval.
***LIBRARY   SLATEC
***CATEGORY  E3, K6
***TYPE      DOUBLE PRECISION (INTRV-S, DINTRV-D)
***KEYWORDS  B-SPLINE, DATA FITTING, INTERPOLATION, SPLINES
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Written by Carl de Boor and modified by D. E. Amos

      Abstract    **** a double precision routine ****
          DINTRV is the INTERV routine of the reference.

          DINTRV computes the largest integer ILEFT in 1 .LE. ILEFT .LE.
          LXT such that XT(ILEFT) .LE. X where XT(*) is a subdivision of
          the X interval.  Precisely,

                      X .LT. XT(1)                1         -1
          if  XT(I) .LE. X .LT. XT(I+1)  then  ILEFT=I   , MFLAG=0
            XT(LXT) .LE. X                        LXT        1,

          That is, when multiplicities are present in the break point
          to the left of X, the largest index is taken for ILEFT.

      Description of Arguments

          Input      XT,X are double precision
           XT      - XT is a knot or break point vector of length LXT
           LXT     - length of the XT vector
           X       - argument
           ILO     - an initialization parameter which must be set
                     to 1 the first time the spline array XT is
                     processed by DINTRV.

          Output
           ILO     - ILO contains information for efficient process-
                     ing after the initial call and ILO must not be
                     changed by the user.  Distinct splines require
                     distinct ILO parameters.
           ILEFT   - largest integer satisfying XT(ILEFT) .LE. X
           MFLAG   - signals when X lies out of bounds

      Error Conditions
          None

***REFERENCES  Carl de Boor, Package for calculating with B-splines,
                 SIAM Journal on Numerical Analysis 14, 3 (June 1977),
                 pp. 441-472.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   800901  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
```

DIR

```
      SUBROUTINE DIR(N, B, X, NELT, IA, JA, A, ISYM, MATVEC, MSOLVE, ITOL,
     $     TOL, ITMAX, ITER, ERR, IERR, IUNIT, R, Z, DZ, RWORK, IWORK)
***BEGIN PROLOGUE  DIR
***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      DOUBLE 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)
      DOUBLE PRECISION B(N), X(N), A(NELT), TOL, ERR, R(N), Z(N), DZ(N)
      DOUBLE PRECISION RWORK(USER DEFINED)
      EXTERNAL MATVEC, MSOLVE

      CALL DIR(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       Double Precision B(N).
         Right-hand side vector.
 X      :INOUT    Double Precision 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       Double Precision 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 double precision 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 /DSLBLK/ 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 /DSLBLK/ 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    Double Precision.
        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      Double Precision.
        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*D1MACH(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     Double Precision R(N).
Z      :WORK     Double Precision Z(N).
DZ     :WORK     Double Precision DZ(N).
        Double Precision arrays used for workspace.
RWORK  :WORK     Double Precision RWORK(USER DEFINED).
        Double Precision 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 DSJAC implements this iterative strategy.
If M = L, the lower triangle of A, then iterative refinement is known as Gauss-Seidel. The SLAP routine DSGS 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 DSJAC and DSGS 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 double precision 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, and 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.

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 DSJAC, DSGS

*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 DSJAC, DSGS

***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  D1MACH, ISDIR
***REVISION HISTORY  (YYMMDD)
   890404  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)
   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 DSLBLK.  (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
```

DLBETA

```
      DOUBLE PRECISION FUNCTION DLBETA (A, B)
***BEGIN PROLOGUE  DLBETA
***PURPOSE  Compute the natural logarithm of the complete Beta
            function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C7B
***TYPE      DOUBLE PRECISION (ALBETA-S, DLBETA-D, CLBETA-C)
***KEYWORDS  FNLIB, LOGARITHM OF THE COMPLETE BETA FUNCTION,
             SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DLBETA(A,B) calculates the double precision natural logarithm of
 the complete beta function for double precision arguments
 A and B.

***REFERENCES  (NONE)
***ROUTINES CALLED  D9LGMC, DGAMMA, DLNGAM, DLNREL, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   900727  Added EXTERNAL statement.  (WRB)
   END PROLOGUE
```

DLGAMS

```
      SUBROUTINE DLGAMS (X, DLGAM, SGNGAM)
***BEGIN PROLOGUE  DLGAMS
***PURPOSE  Compute the logarithm of the absolute value of the Gamma
            function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C7A
***TYPE      DOUBLE PRECISION (ALGAMS-S, DLGAMS-D)
***KEYWORDS  ABSOLUTE VALUE OF THE LOGARITHM OF THE GAMMA FUNCTION,
             FNLIB, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DLGAMS(X,DLGAM,SGNGAM) calculates the double precision natural
 logarithm of the absolute value of the Gamma function for
 double precision argument X and stores the result in double
 precision argument DLGAM.

***REFERENCES  (NONE)
***ROUTINES CALLED  DLNGAM
***REVISION HISTORY  (YYMMDD)
   770701  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
```

DLI

```
      DOUBLE PRECISION FUNCTION DLI (X)
***BEGIN PROLOGUE  DLI
***PURPOSE  Compute the logarithmic integral.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C5
***TYPE      DOUBLE PRECISION (ALI-S, DLI-D)
***KEYWORDS  FNLIB, LOGARITHMIC INTEGRAL, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DLI(X) calculates the double precision logarithmic integral
 for double precision argument X.

***REFERENCES  (NONE)
***ROUTINES CALLED  DEI, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  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)
   END PROLOGUE
```

DLLSIA

```
      SUBROUTINE DLLSIA (A, MDA, M, N, B, MDB, NB, RE, AE, KEY, MODE,
     +   NP, KRANK, KSURE, RNORM, W, LW, IWORK, LIW, INFO)
***BEGIN PROLOGUE  DLLSIA
***PURPOSE  Solve linear least squares problems by performing a QR
            factorization of the input matrix using Householder
            transformations.  Emphasis is put on detecting possible
            rank deficiency.
***LIBRARY   SLATEC
***CATEGORY  D9, D5
***TYPE      DOUBLE PRECISION (LLSIA-S, DLLSIA-D)
***KEYWORDS  LINEAR LEAST SQUARES, QR FACTORIZATION
***AUTHOR  Manteuffel, T. A., (LANL)
***DESCRIPTION

      DLLSIA computes the least squares solution(s) to the problem AX=B
      where A is an M by N matrix with M.GE.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.

      DLLSIA requires (MDA+6)*N + (MDB+1)*NB + M dimensioned space

    ******************************************************************
    *                                                                *
    *         WARNING - All input arrays are changed on exit.        *
    *                                                                *
    ******************************************************************
      SUBROUTINE DLLSIA(A,MDA,M,N,B,MDB,NB,RE,AE,KEY,MODE,NP,
     1   KRANK,KSURE,RNORM,W,LW,IWORK,LIW,INFO)

      Input..All TYPE REAL variables are DOUBLE PRECISION

      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.GE.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. Must have
                    MDB.GE.M. 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 column 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 column 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 solution
```

The inexperienced user is advised to set MODE=0

NP — The first NP columns of A will not be interchanged with other columns even though the pivot strategy would suggest otherwise.
The inexperienced user is advised to set NP=0.

WORK() — A real work array dimensioned 5*N. However, if RE or AE have been specified as vectors, dimension WORK 4*N. If both RE and AE have been specified as vectors, dimension WORK 3*N.

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*N locations of WORK as output by the original call to DLLSIA. 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..All TYPE REAL variable are DOUBLE PRECISION

A(,) — Contains the upper triangular part of the reduced matrix and the transformation information. It togeth with the first N elements of WORK (see below) completely specify the QR 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.

     WORK()        The first N 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 solution
                   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  D1MACH, DU11LS, DU12LS, XERMSG
***REVISION HISTORY  (YYMMDD)
   810801  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   891006  Cosmetic changes to prologue.  (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
```

DLLTI2

```
      SUBROUTINE DLLTI2 (N, B, X, NEL, IEL, JEL, EL, DINV)
***BEGIN PROLOGUE  DLLTI2
***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      DOUBLE 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)
     DOUBLE PRECISION B(N), X(N), EL(NEL), DINV(N)

     CALL DLLTI2( N, B, X, NEL, IEL, JEL, EL, DINV )

 *Arguments:
 N      :IN       Integer
         Order of the Matrix.
 B      :IN       Double Precision B(N).
         Right hand side vector.
 X      :OUT      Double Precision 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       Double Precision     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 DS2LT routine.  See  the
         "Description", below for more details about the  SLAP  Row
         format.
 DINV   :IN       Double Precision 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 DSICCG.   It must be called via the
       SLAP  MSOLVE calling sequence  convention  interface routine
       DSLLI.
         **** 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 DSICS 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 double precision 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)) are the first elements of the IROW-th row in JA and A, and JA(IA(IROW+1)-1), A(IA(IROW+1)-1) are the last elements 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  DSICCG, DSICS
***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
```

DLNGAM

```
      DOUBLE PRECISION FUNCTION DLNGAM (X)
***BEGIN PROLOGUE  DLNGAM
***PURPOSE  Compute the logarithm of the absolute value of the Gamma
            function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C7A
***TYPE      DOUBLE PRECISION (ALNGAM-S, DLNGAM-D, CLNGAM-C)
***KEYWORDS  ABSOLUTE VALUE, COMPLETE GAMMA FUNCTION, FNLIB, LOGARITHM,
             SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DLNGAM(X) calculates the double precision logarithm of the
 absolute value of the Gamma function for double precision
 argument X.

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, D9LGMC, DGAMMA, XERMSG
***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)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900727  Added EXTERNAL statement.  (WRB)
   END PROLOGUE
```

DLNREL

```
      DOUBLE PRECISION FUNCTION DLNREL (X)
***BEGIN PROLOGUE  DLNREL
***PURPOSE  Evaluate ln(1+X) accurate in the sense of relative error.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C4B
***TYPE      DOUBLE PRECISION (ALNREL-S, DLNREL-D, CLNREL-C)
***KEYWORDS  ELEMENTARY FUNCTIONS, FNLIB, LOGARITHM
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DLNREL(X) calculates the double precision natural logarithm of
 (1.0+X) for double precision argument X.  This routine should
 be used when X is small and accurate to calculate the logarithm
 accurately (in the relative error sense) in the neighborhood
 of 1.0.

 Series for ALNR       on the interval -3.75000E-01 to  3.75000E-01
                                        with weighted error   6.35E-32
                                         log weighted error  31.20
                                 significant figures required  30.93
                                      decimal places required  32.01

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DCSEVL, INITDS, XERMSG
***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)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   END PROLOGUE
```

DLPDOC

```
     SUBROUTINE DLPDOC
***BEGIN PROLOGUE  DLPDOC
***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      DOUBLE 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
                     Double Precision Routines

               @@@@  @            @@    @@@@
              @       @ @           @   @   @       @
              @         @          @     @  @       @
               @@@@  @         @        @ @@@@
                       @ @         @@@@@ @
              @       @ @         @       @ @
               @@@@  @@@@@ @       @ @

      @       @                              @@@@        @@@
      @       @                             @       @      @    @
      @       @  @@@@  @ @                     @     @    @  @
      @       @ @       @ @  @              @@@      @   @   @
       @     @  @@@@@ @                @           @  @    @
        @   @   @         @              @          @@  @    @
          @@     @@@@  @                @@@@@ @@   @@@
```

```
==================================================================
========================== 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 double precision 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, and 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 (DCG, DSDCG, DSICSG). If you're not sure it's SPD (symmetric and Positive Definite) then try DCG 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 (DIR) or the GMRES code (DGMRES) if DIR 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 (DGMRES). Both of these methods converge very quickly and do not require A' or M' (' denotes transpose) information. DGMRES 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 DCG 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 double precision version of SLAP, all routine names start with a D. The brackets around the S and M designate that these fields are optional.

Here are some examples of the routines:

1) DBCG: Double 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) DSDBCG: Double precision, SLAP data structure BCG with Diagonal scaling.
3) DSLUBC: Double precision, SLAP data structure BCG with incomplete LU factorization as the preconditioning.
4) DCG: Double precision Conjugate Gradient "core" routine.
5) DSDCG: Double precision, SLAP data structure Conjugate Gradient with Diagonal scaling.
6) DSICCG: Double precision, SLAP data structure Conjugate Gradient 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, DXXX, has a corresponding stop routine, ISDXXX. 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 ISDXXX accordingly.

```
============================== dir.f ==============================
```

DIR: Preconditioned Iterative Refinement Sparse Ax = b Solver.
DSJAC: Jacobi's Method Iterative Sparse Ax = b Solver.
DSGS: Gauss-Seidel Method Iterative Sparse Ax = b Solver.
DSILUR: Incomplete LU Iterative Refinement Sparse Ax = b Solver.

```
============================== dcg.f ==============================
```

DCG: Preconditioned Conjugate Gradient Sparse Ax=b Solver.
DSDCG: Diagonally Scaled Conjugate Gradient Sparse Ax=b Solver.
DSICCG: Incomplete Cholesky Conjugate Gradient Sparse Ax=b Solver.

```
============================= dcgn.f =============================
DCGN: Preconditioned CG Sparse Ax=b Solver for Normal Equations.
DSDCGN: Diagonally Scaled CG Sparse Ax=b Solver for Normal Eqn's.
DSLUCN: Incomplete LU CG Sparse Ax=b Solver for Normal Equations.

============================= dbcg.f =============================
DBCG: Preconditioned BiConjugate Gradient Sparse Ax = b Solver.
DSDBCG: Diagonally Scaled BiConjugate Gradient Sparse Ax=b Solver.
DSLUBC: Incomplete LU BiConjugate Gradient Sparse Ax=b Solver.

============================= dcgs.f =============================
DCGS: Preconditioned BiConjugate Gradient Squared Ax=b Solver.
DSDCGS: Diagonally Scaled CGS Sparse Ax=b Solver.
DSLUCS: Incomplete LU BiConjugate Gradient Squared Ax=b Solver.

============================= domn.f =============================
DOMN: Preconditioned Orthomin Sparse Iterative Ax=b Solver.
DSDOMN: Diagonally Scaled Orthomin Sparse Iterative Ax=b Solver.
DSLUOM: Incomplete LU Orthomin Sparse Iterative Ax=b Solver.

============================ dgmres.f ============================
DGMRES: Preconditioned GMRES Iterative Sparse Ax=b Solver.
DSDGMR: Diagonally Scaled GMRES Iterative Sparse Ax=b Solver.
DSLUGM: Incomplete LU GMRES Iterative Sparse Ax=b Solver.

============================ dmset.f =============================
   The following routines are used to set up preconditioners.

DSDS: Diagonal Scaling Preconditioner SLAP Set Up.
DSDSCL: Diagonally Scales/Unscales a SLAP Column Matrix.
DSD2S: Diagonal Scaling Preconditioner SLAP Normal Eqns Set Up.
DS2LT: Lower Triangle Preconditioner SLAP Set Up.
DSICS: Incomplete Cholesky Decomp. Preconditioner SLAP Set Up.
DSILUS: Incomplete LU Decomposition Preconditioner SLAP Set Up.

============================ dmvops.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.

DSMV: SLAP Column Format Sparse Matrix Vector Product.
DSMTV: SLAP Column Format Sparse Matrix (transpose) Vector Prod.
DSDI: Diagonal Matrix Vector Multiply.
DSLI: SLAP MSOLVE for Lower Triangle Matrix (set up for DSLI2).
DSLI2: Lower Triangle Matrix Backsolve.
DSLLTI: SLAP MSOLVE for LDL' (IC) Fact. (set up for DLLTI2).
DLLTI2: Backsolve routine for LDL' Factorization.
DSLUI: SLAP MSOLVE for LDU Factorization (set up for DSLUI2).
DSLUI2: SLAP Backsolve for LDU Factorization.
DSLUTI: SLAP MTSOLV for LDU Factorization (set up for DSLUI4).
DSLUI4: SLAP Backsolve for LDU Factorization.
DSMMTI: SLAP MSOLVE for LDU Fact of Normal Eq (set up for DSMMI2).
DSMMI2: SLAP Backsolve for LDU Factorization of Normal Equations.
```

```
    ============================ dlaputil.f ==========================
       The following utility routines are useful additions to SLAP.

    DBHIN: Read Sparse Linear System in the Boeing/Harwell Format.
    DCHKW: SLAP WORK/IWORK Array Bounds Checker.
    DCPPLT: Printer Plot of SLAP Column Format Matrix.
    DS2Y: SLAP Triad to SLAP Column Format Converter.
    QS2I1D: Quick Sort Integer array, moving integer and DP arrays.
            (Used by DS2Y.)
    DTIN: Read in SLAP Triad Format Linear System.
    DTOUT: 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)
   890404  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
```

DLSEI

```
      SUBROUTINE DLSEI (W, MDW, ME, MA, MG, N, PRGOPT, X, RNORME,
     +   RNORML, MODE, WS, IP)
***BEGIN PROLOGUE  DLSEI
***PURPOSE  Solve a linearly constrained least squares problem with
            equality and inequality constraints, and optionally compute
            a covariance matrix.
***LIBRARY   SLATEC
***CATEGORY  K1A2A, D9
***TYPE      DOUBLE PRECISION (LSEI-S, DLSEI-D)
***KEYWORDS  CONSTRAINED LEAST SQUARES, CURVE FITTING, DATA FITTING,
             EQUALITY CONSTRAINTS, INEQUALITY CONSTRAINTS,
             QUADRATIC PROGRAMMING
***AUTHOR  Hanson, R. J., (SNLA)
           Haskell, K. H., (SNLA)
***DESCRIPTION

      Abstract

      This subprogram solves a linearly constrained least squares
      problem with both equality and inequality constraints, and, if the
      user requests, obtains a covariance matrix of the solution
      parameters.

      Suppose there are given matrices E, A and G of respective
      dimensions ME by N, MA by N and MG by N, and vectors F, B and H of
      respective lengths ME, MA and MG.  This subroutine solves the
      linearly constrained least squares problem

                   EX = F, (E ME by N) (equations to be exactly
                                       satisfied)
                   AX = B, (A MA by N) (equations to be
                                       approximately satisfied,
                                       least squares sense)
                   GX .GE. H,(G MG by N) (inequality constraints)

      The inequalities GX .GE. H mean that every component of the
      product GX must be .GE. the corresponding component of H.

      In case the equality constraints cannot be satisfied, a
      generalized inverse solution residual vector length is obtained
      for F-EX.  This is the minimal length possible for F-EX.

      Any values ME .GE. 0, MA .GE. 0, or MG .GE. 0 are permitted.  The
      rank of the matrix E is estimated during the computation.  We call
      this value KRANKE.  It is an output parameter in IP(1) defined
      below.  Using a generalized inverse solution of EX=F, a reduced
      least squares problem with inequality constraints is obtained.
      The tolerances used in these tests for determining the rank
      of E and the rank of the reduced least squares problem are
      given in Sandia Tech. Rept. SAND-78-1290.  They can be
      modified by the user if new values are provided in
      the option list of the array PRGOPT(*).

      The user must dimension all arrays appearing in the call list..
      W(MDW,N+1),PRGOPT(*),X(N),WS(2*(ME+N)+K+(MG+2)*(N+7)),IP(MG+2*N+2)
      where K=MAX(MA+MG,N).  This allows for a solution of a range of
```

problems in the given working space. The dimension of WS(*) given is a necessary overestimate. Once a particular problem has been run, the output parameter IP(3) gives the actual dimension required for that problem.

The parameters for DLSEI( ) are

Input.. All TYPE REAL variables are DOUBLE PRECISION

W(*,*),MDW, ME,MA,MG,N — The array W(*,*) is doubly subscripted with first dimensioning parameter equal to MDW. For this discussion let us call M = ME+MA+MG. 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)
(G  H)
```

in rows and columns 1,...,M and 1,...,N+1 respectively.

The integers ME, MA, and MG are the respective matrix row dimensions of E, A and G. Each matrix has N columns.

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=1

Compute in W(*,*) the N by N covariance matrix of the solution variables as an output parameter. Nominally the covariance matrix will not be computed. (This requires no user input.) The data set for this option is a single value. It must be nonzero when the covariance matrix is desired. If it is zero, the covariance matrix is not computed. When the covariance matrix is computed, the first dimensioning parameter of the array W(*,*) must satisfy MDW .GE. MAX(M,N).

KEY=10

Suppress scaling of the inverse of the normal matrix by the scale factor RNORM**2/ MAX(1, no. of degrees of freedom). This option only applies when the option for computing the covariance matrix (KEY=1) is used. With KEY=1 and KEY=10 used as options the unscaled inverse of the normal matrix is returned in W(*,*). The data set for this option is a single value. When it is nonzero no scaling is done. When it is zero scaling is done. The nominal case is to do scaling so if option (KEY=1) is used alone, the matrix will be scaled on output.

KEY=2

Scale the nonzero columns of the entire data matrix.

(E)
(A)
(G)

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=3
Scale columns of the entire data matrix
(E)
(A)
(G)

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=4
Change the rank determination tolerance for
the equality constraint equations from
the nominal value of SQRT(DRELPR). This quantity can
be no smaller than DRELPR, the arithmetic-
storage precision. The quantity DRELPR is the
largest positive number such that T=1.+DRELPR
satisfies T .EQ. 1. The quantity used
here is internally restricted to be at
least DRELPR. The data set for this option
is the new tolerance.

KEY=5
Change the rank determination tolerance for
the reduced least squares equations from
the nominal value of SQRT(DRELPR). This quantity can
be no smaller than DRELPR, the arithmetic-
storage precision. The quantity used
here is internally restricted to be at
least DRELPR. The data set for this option
is the new tolerance.

For example, suppose we want to change
the tolerance for the reduced least squares
problem, compute the covariance matrix of
the solution parameters, and provide
column scaling for the data matrix. For
these options the dimension of PRGOPT(*)
must be at least N+9. The Fortran statements
defining these options would be as follows:

```
PRGOPT(1)=4 (link to entry 4 in PRGOPT(*))
PRGOPT(2)=1 (covariance matrix key)
PRGOPT(3)=1 (covariance matrix wanted)

PRGOPT(4)=7 (link to entry 7 in PRGOPT(*))
PRGOPT(5)=5 (least squares equas.  tolerance key)
PRGOPT(6)=... (new value of the tolerance)

PRGOPT(7)=N+9 (link to entry N+9 in PRGOPT(*))
PRGOPT(8)=3 (user-provided column scaling key)

CALL DCOPY (N, D, 1, PRGOPT(9), 1)  (Copy the N
  scaling factors from the user array D(*)
  to PRGOPT(9)-PRGOPT(N+8))

PRGOPT(N+9)=1 (no more options to change)
```

The contents of PRGOPT(*) are not modified by the subprogram.
The options for WNNLS( ) can also be included in this array. The values of KEY recognized by WNNLS( ) are 6, 7 and 8. Their functions are documented in the usage instructions for subroutine WNNLS( ). Normally these options do not need to be modified when using DLSEI( ).

IP(1), IP(2) — The amounts of working storage actually allocated for the working arrays WS(*) and IP(*), respectively. These quantities are compared with the actual amounts of storage needed by DLSEI( ). Insufficient storage allocated for either WS(*) or IP(*) is an error. This feature was included in DLSEI( ) because miscalculating the storage formulas for WS(*) and IP(*) might very well lead to subtle and hard-to-find execution errors.

The length of WS(*) must be at least

LW = 2*(ME+N)+K+(MG+2)*(N+7)

where K = max(MA+MG,N)
This test will not be made if IP(1).LE.0.

The length of IP(*) must be at least

LIP = MG+2*N+2
This test will not be made if IP(2).LE.0.

Output.. All TYPE REAL variables are DOUBLE PRECISION

X(*),RNORME, RNORML — The array X(*) contains the solution parameters if the integer output flag MODE = 0 or 1. The definition of MODE is given directly below. When MODE = 0 or 1, RNORME and RNORML respectively contain the residual vector Euclidean lengths of F - EX and B - AX. When MODE=1 the equality constraint equations EX=F are contradictory, so RNORME .NE. 0. The residual vector F-EX has minimal Euclidean length. For MODE .GE. 2, none of these parameters is defined.

MODE — Integer flag that indicates the subprogram status after completion. If MODE .GE. 2, no solution has been computed.

MODE =

0 Both equality and inequality constraints are compatible and have been satisfied.

1 Equality constraints are contradictory. A generalized inverse solution of EX=F was used to minimize the residual vector length F-EX. In this sense, the solution is still meaningful.

2 Inequality constraints are contradictory.

3 Both equality and inequality constraints are contradictory.

The following interpretation of MODE=1,2 or 3 must be made. The sets consisting of all solutions of the equality constraints EX=F and all vectors satisfying GX .GE. H have no points in common. (In particular this does not say that each individual set has no points at all, although this could be the case.)

4 Usage error occurred. The value of MDW is .LT. ME+MA+MG, MDW is .LT. N and a covariance matrix is requested, or the option vector PRGOPT(*) is not properly defined, or the lengths of the working arrays WS(*) and IP(*), when specified in IP(1) and IP(2) respectively, are not long enough.

W(*,*) The array W(*,*) contains the N by N symmetric covariance matrix of the solution parameters, provided this was requested on input with the option vector PRGOPT(*) and the output flag is returned with MODE = 0 or 1.

IP(*) The integer working array has three entries that provide rank and working array length information after completion.

IP(1) = rank of equality constraint matrix. Define this quantity as KRANKE.

IP(2) = rank of reduced least squares problem.

IP(3) = the amount of storage in the working array WS(*) that was actually used by the subprogram. The formula given above for the length of WS(*) is a necessary overestimate. If exactly the same problem matrices are used in subsequent executions, the declared dimension of WS(*) can be reduced to this output value.

User Designated Working Arrays..

WS(*),IP(*) These are respectively type real and type integer working arrays. Their required minimal lengths are given above.

```
***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.
***ROUTINES CALLED  D1MACH, DASUM, DAXPY, DCOPY, DDOT, DH12, DLSI,
                    DNRM2, DSCAL, DSWAP, XERMSG
***REVISION HISTORY  (YYMMDD)
   790701  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890618  Completely restructured and extensively revised (WRB & RWC)
   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)
   900604  DP version created from SP version.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DNBCO

```
      SUBROUTINE DNBCO (ABE, LDA, N, ML, MU, IPVT, RCOND, Z)
***BEGIN PROLOGUE  DNBCO
***PURPOSE  Factor a band matrix using Gaussian elimination and
            estimate the condition number.
***LIBRARY   SLATEC
***CATEGORY  D2A2
***TYPE      DOUBLE PRECISION (SNBCO-S, DNBCO-D, CNBCO-C)
***KEYWORDS  BANDED, LINEAR EQUATIONS, MATRIX FACTORIZATION,
             NONSYMMETRIC
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

      DNBCO factors a double precision band matrix by Gaussian
      elimination and estimates the condition of the matrix.

      If RCOND is not needed, DNBFA is slightly faster.
      To solve  A*X = B , follow DNBCO by DNBSL.
      To compute  INVERSE(A)*C , follow DNBCO by DNBSL.
      To compute  DETERMINANT(A) , follow DNBCO by DNBDI.

      On Entry

         ABE     DOUBLE PRECISION(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   DOUBLE PRECISION
           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       DOUBLE PRECISION(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  DASUM, DAXPY, DDOT, DNBFA, DSCAL
***REVISION HISTORY  (YYMMDD)
   800728  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
```

DNBDI

```
      SUBROUTINE DNBDI (ABE, LDA, N, ML, MU, IPVT, DET)
***BEGIN PROLOGUE  DNBDI
***PURPOSE  Compute the determinant of a band matrix using the factors
            computed by DNBCO or DNBFA.
***LIBRARY   SLATEC
***CATEGORY  D3A2
***TYPE      DOUBLE PRECISION (SNBDI-S, DNBDI-D, CNBDI-C)
***KEYWORDS  BANDED, DETERMINANT, LINEAR EQUATIONS, NONSYMMETRIC
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

      DNBDI computes the determinant of a band matrix
      using the factors computed by DNBCO or DNBFA.
      If the inverse is needed, use DNBSL  N  times.

      On Entry

         ABE     DOUBLE PRECISION(LDA, NC)
                 the output from DNBCO or DNBFA.
                 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 DNBCO or DNBFA.

      On Return

         DET     DOUBLE PRECISION(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)
   800728  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
```

DNBFA

```
      SUBROUTINE DNBFA (ABE, LDA, N, ML, MU, IPVT, INFO)
***BEGIN PROLOGUE  DNBFA
***PURPOSE  Factor a band matrix by elimination.
***LIBRARY   SLATEC
***CATEGORY  D2A2
***TYPE      DOUBLE PRECISION (SNBFA-S, DNBFA-D, CNBFA-C)
***KEYWORDS  BANDED, LINEAR EQUATIONS, MATRIX FACTORIZATION,
             NONSYMMETRIC
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

      DNBFA factors a double precision band matrix by elimination.

      DNBFA is usually called by DNBCO, but it can be called
      directly with a saving in time if RCOND is not needed.

      On Entry

         ABE     DOUBLE PRECISION(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 DNBSL will divide by zero if called. Use RCOND in DNBCO 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 DAXPY, DSCAL, DSWAP, IDAMAX
***REVISION HISTORY (YYMMDD)
800728 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

DNBFS

```
      SUBROUTINE DNBFS (ABE, LDA, N, ML, MU, V, ITASK, IND, WORK, IWORK)
***BEGIN PROLOGUE  DNBFS
***PURPOSE  Solve a general nonsymmetric banded system of linear
            equations.
***LIBRARY   SLATEC
***CATEGORY  D2A2
***TYPE      DOUBLE PRECISION (SNBFS-S, DNBFS-D, CNBFS-C)
***KEYWORDS  BANDED, LINEAR EQUATIONS, NONSYMMETRIC
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

    Subroutine DNBFS solves a general nonsymmetric banded NxN
    system of double precision real linear equations using
    SLATEC subroutines DNBCO and DNBSL.  These are adaptations
    of the LINPACK subroutines DGBCO and DGBSL which require
    a different format for storing the matrix elements.  If
    A  is an NxN double precision matrix and if  X  and  B  are
    double precision N-vectors, then DNBFS 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.
    DNBFS uses less time and storage than the corresponding
    program for general matrices (DGEFS) 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 DNBFS
    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 DOUBLE PRECISION(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 DNBFS. 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 DOUBLE PRECISION(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   DOUBLE PRECISION(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  D1MACH, DNBCO, DNBSL, XERMSG
***REVISION HISTORY  (YYMMDD)
   800812  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, changed GOTOs to
           IF-THEN-ELSEs.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DNBSL

```
      SUBROUTINE DNBSL (ABE, LDA, N, ML, MU, IPVT, B, JOB)
***BEGIN PROLOGUE  DNBSL
***PURPOSE  Solve a real band system using the factors computed by
            DNBCO or DNBFA.
***LIBRARY   SLATEC
***CATEGORY  D2A2
***TYPE      DOUBLE PRECISION (SNBSL-S, DNBSL-D, CNBSL-C)
***KEYWORDS  BANDED, LINEAR EQUATIONS, NONSYMMETRIC, SOLVE
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

      DNBSL solves the double precision band system
      A * X = B  or  TRANS(A) * X = B
      using the factors computed by DNBCO or DNBFA.

      On Entry

         ABE     DOUBLE PRECISION(LDA, NC)
                 the output from DNBCO or DNBFA.
                 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 DNBCO or DNBFA.

         B       DOUBLE PRECISION(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 DNBCO has set RCOND .GT. 0.0
         or DNBFA has set INFO .EQ. 0 .
```

```
     To compute  INVERSE(A) * C  where  C  is a matrix
     with  P  columns
           CALL DNBCO(ABE,LDA,N,ML,MU,IPVT,RCOND,Z)
           IF (RCOND is too small) GO TO ...
           DO 10 J = 1, P
             CALL DNBSL(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  DAXPY, DDOT
***REVISION HISTORY  (YYMMDD)
   800728  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
```

DNLS1

```
      SUBROUTINE DNLS1 (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  DNLS1
***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      DOUBLE 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 DNLS1 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 subrou-
       tine 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 DNLS1(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)
       DOUBLE PRECISION FTOL,XTOL,GTOL,EPSFCN,FACTOR
       DOUBLE PRECISION 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 DNLS1 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 DNLS1.

      FCN is the name of the user-supplied subroutine which calculate
         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
DOUBLE PRECISION X(N),FVEC(M)
----------
FJAC and LDFJAC may be ignored       , if IOPT=1.
DOUBLE PRECISION FJAC(LDFJAC,N)      , if IOPT=2.
DOUBLE PRECISION 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 DNLS1. 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 DNLS1 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. DNLS1 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 DNLS1 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 DNLS1 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 DNLS1 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 DNLS1, 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 DNLS1 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 DNLS1. In this case, it may be possible to remedy the situation by rerunning DNLS1 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 DNLS1 with MODE set to 1.

6. Characteristics of the Algorithm.

DNLS1 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 DNLS1 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 DNLS1 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 DNLS1 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 DNLS1 will be strongly influenced by the time spent in FCN.

Storage. DNLS1 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 DNLS1 example.
C
      INTEGER J,IOPT,M,N,LDFJAC,MAXFEV,MODE,NPRINT,INFO,NFEV,NJEV,
     *        NWRITE
      INTEGER IPVT(3)
      DOUBLE PRECISION FTOL,XTOL,GTOL,FACTOR,FNORM,EPSFCN
      DOUBLE PRECISION X(3),FVEC(15),FJAC(15,3),DIAG(3),QTF(3),
     *     WA1(3),WA2(3),WA3(3),WA4(15)
      DOUBLE PRECISION DENORM,D1MACH
      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 DNLS1(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 I,M,N,IFLAG
      DOUBLE PRECISION X(N),FVEC(M),Y(15)
      DOUBLE PRECISION TMP1,TMP2,TMP3,TMP4
      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 I,LDFJAC,M,N,IFLAG
      DOUBLE PRECISION X(N),FVEC(M),FJAC(LDFJAC,N),Y(15)
      DOUBLE PRECISION TMP1,TMP2,TMP3,TMP4
      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 I,M,N,IFLAG
      DOUBLE PRECISION X(N),FVEC(M),FJAC(N),Y(15)
      DOUBLE PRECISION TMP1,TMP2,TMP3,TMP4
      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 D1MACH, DCKDER, DENORM, DFDJC3, DMPAR, DQRFAC, DWUPDT, XERMSG

***REVISION HISTORY (YYMMDD)

800301	DATE WRITTEN
890531	Changed all specific intrinsics to generic. (WRB)
890831	Modified array declarations. (WRB)
891006	Cosmetic changes to prologue. (WRB)
891006	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)
920205	Corrected XERN1 declaration. (WRB)

```
920501  Reformatted the REFERENCES section.  (WRB)
END PROLOGUE
```

DNLS1E

```
      SUBROUTINE DNLS1E (FCN, IOPT, M, N, X, FVEC, TOL, NPRINT, INFO,
     +   IW, WA, LWA)
***BEGIN PROLOGUE  DNLS1E
***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      DOUBLE 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 DNLS1E 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 DNLS1. 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 DNLS1E(FCN,IOPT,M,N,X,FVEC,TOL,NPRINT,
     *                  INFO,IW,WA,LWA)
      INTEGER IOPT,M,N,NPRINT,INFO,LWAC,IW(N)
      DOUBLE PRECISION TOL,X(N),FVEC(M),WA(LWA)
      EXTERNAL FCN
```

3. Parameters. ALL TYPE REAL parameters are DOUBLE PRECISION

Parameters designated as input parameters must be specified on entry to DNLS1E 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 DNLS1E.

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
  DOUBLE PRECISION X(N),FVEC(M)
  ----------
  FJAC and LDFJAC may be ignored       , if IOPT=1.
  DOUBLE PRECISION FJAC(LDFJAC,N)      , if IOPT=2.
  DOUBLE PRECISION 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 DNLS1E. 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 DNLS1E 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. DNLS1E terminates when any of the tests is satisfied. If TOL is less than the machine precision (as defined by the function

R1MACH(4)), then DNLS1E 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 DNLS1E 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 DNLS1E 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 DNLS1E) 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 DNLS1E, 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 DNLS1E 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 DNLS1E. 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 DNLS1E, thereby forcing it to disregard old (and possibly harmful) information.

6. Characteristics of the Algorithm.

DNLS1E 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 DNLS1E 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 DNLS1E 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 DNLS1E is about N**3 to process each evaluation of the functions (call to FCN) and to process each evaluation of the Jacobian DNLS1E 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 DNLS1E will be strongly influenced by the time spent in FCN.

Storage. DNLS1E 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 DNLS1E example.
C
      INTEGER I,IOPT,M,N,NPRINT,JNFO,LWA,NWRITE
      INTEGER IW(3)
      DOUBLE PRECISION TOL,FNORM,X(3),FVEC(15),WA(75)
      DOUBLE PRECISION DENORM,D1MACH
      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 DNLS1E(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
     *        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 I,M,N,IFLAG
      DOUBLE PRECISION X(N),FVEC(M),Y(15)
      DOUBLE PRECISION TMP1,TMP2,TMP3,TMP4
      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 I,LDFJAC,M,N,IFLAG
      DOUBLE PRECISION X(N),FVEC(M),FJAC(LDFJAC,N),Y(15)
      DOUBLE PRECISION TMP1,TMP2,TMP3,TMP4
      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 I,M,N,IFLAG
      DOUBLE PRECISION X(N),FVEC(M),FJAC(N),Y(15)
      DOUBLE PRECISION TMP1,TMP2,TMP3,TMP4
      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  DNLS1, XERMSG
***REVISION HISTORY  (YYMMDD)
   800301  DATE WRITTEN
   890831  Modified array declarations.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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
```

DNRM2

```
      DOUBLE PRECISION FUNCTION DNRM2 (N, DX, INCX)
***BEGIN PROLOGUE  DNRM2
***PURPOSE  Compute the Euclidean length (L2 norm) of a vector.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A3B
***TYPE      DOUBLE 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)
       DX  double precision vector with N elements
     INCX  storage spacing between elements of DX

     --Output--
    DNRM2  double precision result (zero if N .LE. 0)

     Euclidean norm of the N-vector stored in DX 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
```

DNSQ

```
      SUBROUTINE DNSQ (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  DNSQ
***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      DOUBLE PRECISION (SNSQ-S, DNSQ-D)
***KEYWORDS  NONLINEAR SQUARE SYSTEM, POWELL HYBRID METHOD, ZEROS
***AUTHOR  Hiebert, K. L. (SNLA)
***DESCRIPTION

 1. Purpose.

       The purpose of DNSQ is to find a zero of a system of N nonlinear
       functions in N variables by a modification of the Powell
       hybrid method.  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)
       HYBRD and HYBRDJ.

 2. Subroutine and Type Statements.

       SUBROUTINE DNSQ(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
       DOUBLE PRECISION XTOL,EPSFCN,FACTOR
       DOUBLE PRECISION
       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 DNSQ 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 DNSQ.

       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
         DOUBLE PRECISION 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 DNSQ. 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
DOUBLE PRECISION 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 DNSQ. 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 nonnegative 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 nonnegative 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 nonnegative 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 iterations 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

evaluations.

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 DNSQ 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. DNSQ terminates when the test is satisfied. If the convergence parameter is less than the machine precision (as defined by the function D1MACH(4)), then DNSQ 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 DNSQ may incorrectly indicate convergence. The coding of the Jacobian can be checked by the subroutine DCKDER. If the Jacobian is coded correctly or IOPT=2, then the validity of the answer can be checked, for example, by rerunning DNSQ with a tighter tolerance.

Convergence Test. If DENORM(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

```
DENORM(D*(X-XSOL)) .LE. XTOL*DENORM(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 DNSQ 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 DNSQ 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 DNSQ. In this case, it may be possible to remedy the situation by rerunning DNSQ 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 DNSQ, causing termination with info = 4 or INFO = 5.

Lack of Good Progress. DNSQ 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 DNSQ from a different starting point may be helpful.

6. Characteristics of The Algorithm.

DNSQ 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 DNSQ 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 DNSQ 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 DNSQ will be strongly influenced by the time

spent in FCN and JAC.

Storage. DNSQ 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.

*Long Description:

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 DNSQ example.
C
      INTEGER J,IOPT,N,MAXFEV,ML,MU,MODE,NPRINT,INFO,NFEV,LDFJAC,LR,
     *        NWRITE
      DOUBLE PRECISION XTOL,EPSFCN,FACTOR,FNORM
      DOUBLE PRECISION X(9),FVEC(9),DIAG(9),FJAC(9,9),R(45),QTF(9),
     *     WA1(9),WA2(9),WA3(9),WA4(9)
      DOUBLE PRECISION DENORM,D1MACH
      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(D1MACH(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 DNSQ(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 = DENORM(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
      DOUBLE PRECISION X(N),FVEC(N)
      INTEGER K
      DOUBLE PRECISION 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  D1MACH, D1MPYQ, D1UPDT, DDOGLG, DENORM, DFDJC1,
                    DQFORM, DQRFAC, 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
```

DNSQE

```
      SUBROUTINE DNSQE (FCN, JAC, IOPT, N, X, FVEC, TOL, NPRINT, INFO,
     +   WA, LWA)
***BEGIN PROLOGUE  DNSQE
***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      DOUBLE 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 DNSQE is to find a zero of a system of N
       nonlinear functions in N variables by a modification of the
       Powell hybrid method.  This is done by using the more general
       nonlinear equation solver DNSQ.  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 DNSQE(FCN,JAC,IOPT,N,X,FVEC,TOL,NPRINT,INFO,
      *                 WA,LWA)
       INTEGER IOPT,N,NPRINT,INFO,LWA
       DOUBLE PRECISION TOL
       DOUBLE PRECISION X(N),FVEC(N),WA(LWA)
       EXTERNAL FCN,JAC

 3. Parameters.

       Parameters designated as input parameters must be specified on
       entry to DNSQE 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 DNSQE.

       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
         DOUBLE PRECISION 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 DNSQE. 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
DOUBLE PRECISION 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 DNSQE. 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 nonnegative 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 iterations 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 DNSQE 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. DNSQE terminates when the test is satisfied. If TOL is less than the machine precision (as defined by the function D1MACH(4)), then DNSQE 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 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 DNSQE may incorrectly indicate convergence. The coding of the Jacobian can be checked by the subroutine DCKDER. If the Jacobian is coded correctly or IOPT=2, then the validity of the answer can be checked, for example, by rerunning DNSQE with a tighter tolerance.

Convergence Test. If DENORM(Z) denotes the Euclidean norm of a vector Z, then this test attempts to guarantee that

```
DENORM(X-XSOL) .LE. TOL*DENORM(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 DNSQE usually avoids this possibility.

5. Unsuccessful Completion.

Unsuccessful termination of DNSQE can be due to improper input parameters, arithmetic interrupts, an excessive number of function 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 DNSQE. 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 DNSQE, 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, DNSQE may fail (usually with INFO = 4). The user should then either use DNSQ and set the step length or use IOPT=1 and supply the Jacobian.

Lack of Good Progress. DNSQE 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 DNSQE from a different starting point may be helpful.

6. Characteristics of The Algorithm.

DNSQE 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 DNSQE 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 DNSQE 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 DNSQE will be strongly influenced by the time spent in FCN and JAC.

Storage. DNSQE 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.

*Long Description:

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 DNSQE EXAMPLE.
C
      INTEGER J,N,IOPT,NPRINT,INFO,LWA,NWRITE
      DOUBLE PRECISION TOL,FNORM
      DOUBLE PRECISION X(9),FVEC(9),WA(180)
      DOUBLE PRECISION DENORM,D1MACH
      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(D1MACH(4))
C
      CALL DNSQE(FCN,JAC,IOPT,N,X,FVEC,TOL,NPRINT,INFO,WA,LWA)
      FNORM = DENORM(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
      DOUBLE PRECISION X(N),FVEC(N)
      INTEGER K
      DOUBLE PRECISION 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  DNSQ, 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
```

DOMN

```
      SUBROUTINE DOMN(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  DOMN
***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      DOUBLE 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)
     DOUBLE PRECISION B(N), X(N), A(NELT), TOL, ERR, R(N), Z(N)
     DOUBLE PRECISION P(N,0:NSAVE), AP(N,0:NSAVE), EMAP(N,0:NSAVE)
     DOUBLE PRECISION DZ(N), CSAV(NSAVE), RWORK(USER DEFINED)
     EXTERNAL MATVEC, MSOLVE

     CALL DOMN(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       Double Precision B(N).
         Right-hand side vector.
 X      :INOUT    Double Precision 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       Double Precision 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 double precision 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 /DSLBLK/ 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 /DSLBLK/ 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    Double Precision.
        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      Double Precision.
        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*D1MACH(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     Double Precision R(N).
Z      :WORK     Double Precision Z(N).
P      :WORK     Double Precision P(N,0:NSAVE).
AP     :WORK     Double Precision AP(N,0:NSAVE).
EMAP   :WORK     Double Precision EMAP(N,0:NSAVE).
DZ     :WORK     Double Precision DZ(N).
CSAV   :WORK     Double Precision CSAV(NSAVE)
        Double Precision arrays used for workspace.
RWORK  :WORK     Double Precision RWORK(USER DEFINED).
        Double Precision 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 DSDOMN and DSLUOM 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 double precision 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, and 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.

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 DSDOMN, DSLUOM, ISDOMN

***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 D1MACH, DAXPY, DCOPY, DDOT, ISDOMN

***REVISION HISTORY (YYMMDD)

890404 DATE WRITTEN
890404 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)

```
910502  Removed MATVEC and MSOLVE from ROUTINES CALLED list.  (FNF)
920407  COMMON BLOCK renamed DSLBLK.  (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
```

DP1VLU

```
      SUBROUTINE DP1VLU (L, NDER, X, YFIT, YP, A)
***BEGIN PROLOGUE  DP1VLU
***PURPOSE  Use the coefficients generated by DPOLFT to evaluate the
            polynomial fit of degree L, along with the first NDER of
            its derivatives, at a specified point.
***LIBRARY   SLATEC
***CATEGORY  K6
***TYPE      DOUBLE PRECISION (PVALUE-S, DP1VLU-D)
***KEYWORDS  CURVE FITTING, LEAST SQUARES, POLYNOMIAL APPROXIMATION
***AUTHOR  Shampine, L. F., (SNLA)
           Davenport, S. M., (SNLA)
***DESCRIPTION

      Abstract

      The subroutine  DP1VLU  uses the coefficients generated by  DPOLFT
      to evaluate the polynomial fit of degree  L , along with the first
      NDER  of its derivatives, at a specified point.  Computationally
      stable recurrence relations are used to perform this task.

      The parameters for  DP1VLU  are

      Input -- ALL TYPE REAL variables are DOUBLE PRECISION
          L -      the degree of polynomial to be evaluated.  L  may be
                   any non-negative integer which is less than or equal
                   to  NDEG , the highest degree polynomial provided
                   by  DPOLFT .
          NDER -   the number of derivatives to be evaluated.  NDER
                   may be 0 or any positive value.  If NDER is less
                   than 0, it will be treated as 0.
          X -      the argument at which the polynomial and its
                   derivatives are to be evaluated.
          A -      work and output array containing values from last
                   call to  DPOLFT .

      Output -- ALL TYPE REAL variables are DOUBLE PRECISION
          YFIT -   value of the fitting polynomial of degree  L  at  X
          YP -     array containing the first through  NDER  derivatives
                   of the polynomial of degree  L .  YP  must be
                   dimensioned at least  NDER  in the calling program.

***REFERENCES  L. F. Shampine, S. M. Davenport and R. E. Huddleston,
                 Curve fitting by polynomials in one variable, Report
                 SLA-74-0270, Sandia Laboratories, June 1974.
***ROUTINES CALLED  XERMSG
***REVISION HISTORY  (YYMMDD)
   740601  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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
```

DPBCO

```
      SUBROUTINE DPBCO (ABD, LDA, N, M, RCOND, Z, INFO)
***BEGIN PROLOGUE  DPBCO
***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      DOUBLE 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

      DPBCO factors a double precision symmetric positive definite
      matrix stored in band form and estimates the condition of the
      matrix.

      If  RCOND  is not needed, DPBFA is slightly faster.
      To solve  A*X = B , follow DPBCO by DPBSL.
      To compute  INVERSE(A)*C , follow DPBCO by DPBSL.
      To compute  DETERMINANT(A) , follow DPBCO by DPBDI.

      On Entry

         ABD     DOUBLE PRECISION(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   DOUBLE PRECISION
                 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      DOUBLE PRECISION(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  DASUM, DAXPY, DDOT, DPBFA, DSCAL
***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
```

DPBDI

```
      SUBROUTINE DPBDI (ABD, LDA, N, M, DET)
***BEGIN PROLOGUE  DPBDI
***PURPOSE  Compute the determinant of a symmetric positive definite
            band matrix using the factors computed by DPBCO or DPBFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D3B2
***TYPE      DOUBLE 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

      DPBDI computes the determinant
      of a double precision symmetric positive definite band matrix
      using the factors computed by DPBCO or DPBFA.
      If the inverse is needed, use DPBSL  N  times.

      On Entry

         ABD     DOUBLE PRECISION(LDA, N)
                 the output from DPBCO or DPBFA.

         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     DOUBLE PRECISION(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
```

DPBFA

```
      SUBROUTINE DPBFA (ABD, LDA, N, M, INFO)
***BEGIN PROLOGUE  DPBFA
***PURPOSE  Factor a real symmetric positive definite matrix stored in
            in band form.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B2
***TYPE      DOUBLE 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

      DPBFA factors a double precision symmetric positive definite
      matrix stored in band form.

      DPBFA is usually called by DPBCO, but it can be called
      directly with a saving in time if  RCOND  is not needed.

      On Entry

         ABD     DOUBLE PRECISION(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  DDOT
***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
```

DPBSL

```
      SUBROUTINE DPBSL (ABD, LDA, N, M, B)
***BEGIN PROLOGUE  DPBSL
***PURPOSE  Solve a real symmetric positive definite band system
            using the factors computed by DPBCO or DPBFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B2
***TYPE      DOUBLE 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

      DPBSL solves the double precision symmetric positive definite
      band system  A*X = B
      using the factors computed by DPBCO or DPBFA.

      On Entry

         ABD     DOUBLE PRECISION(LDA, N)
                 the output from DPBCO or DPBFA.

         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       DOUBLE PRECISION(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 DPBCO(ABD,LDA,N,RCOND,Z,INFO)
            IF (RCOND is too small .OR. INFO .NE. 0) GO TO ...
            DO 10 J = 1, P
               CALL DPBSL(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  DAXPY, DDOT
```

```
***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
```

DPCHBS

```
      SUBROUTINE DPCHBS (N, X, F, D, INCFD, KNOTYP, NKNOTS, T, BCOEF,
     +   NDIM, KORD, IERR)
***BEGIN PROLOGUE  DPCHBS
***PURPOSE  Piecewise Cubic Hermite to B-Spline converter.
***LIBRARY   SLATEC (PCHIP)
***CATEGORY  E3
***TYPE      DOUBLE PRECISION (PCHBS-S, DPCHBS-D)
***KEYWORDS  B-SPLINES, CONVERSION, CUBIC HERMITE INTERPOLATION,
             PIECEWISE CUBIC INTERPOLATION
***AUTHOR  Fritsch, F. N., (LLNL)
             Computing and Mathematics Research Division
             Lawrence Livermore National Laboratory
             P.O. Box 808  (L-316)
             Livermore, CA  94550
             FTS 532-4275, (510) 422-4275
***DESCRIPTION

 *Usage:

        INTEGER  N, INCFD, KNOTYP, NKNOTS, NDIM, KORD, IERR
        PARAMETER  (INCFD = ...)
        DOUBLE PRECISION  X(nmax), F(INCFD,nmax), D(INCFD,nmax),
       *      T(2*nmax+4), BCOEF(2*nmax)

        CALL DPCHBS (N, X, F, D, INCFD, KNOTYP, NKNOTS, T, BCOEF,
       *             NDIM, KORD, IERR)

 *Arguments:

     N:IN  is the number of data points, N.ge.2 .  (not checked)

     X:IN  is the real array of independent variable values.  The
           elements of X must be strictly increasing:
                X(I-1) .LT. X(I),  I = 2(1)N.   (not checked)
           nmax, the dimension of X, must be .ge.N.

     F:IN  is the real array of dependent variable values.
           F(1+(I-1)*INCFD) is the value corresponding to X(I).
           nmax, the second dimension of F, must be .ge.N.

     D:IN  is the real array of derivative values at the data points.
           D(1+(I-1)*INCFD) is the value corresponding to X(I).
           nmax, the second dimension of D, must be .ge.N.

     INCFD:IN  is the increment between successive values in F and D.
           This argument is provided primarily for 2-D applications.
           It may have the value 1 for one-dimensional applications,
           in which case F and D may be singly-subscripted arrays.

     KNOTYP:IN  is a flag to control the knot sequence.
           The knot sequence T is normally computed from X by putting
           a double knot at each X and setting the end knot pairs
           according to the value of KNOTYP:
              KNOTYP = 0:  Quadruple knots at X(1) and X(N).  (default)
              KNOTYP = 1:  Replicate lengths of extreme subintervals:
                           T( 1 ) = T( 2 ) = X(1) - (X(2)-X(1))  ;
```

```
                    T(M+4) = T(M+3) = X(N) + (X(N)-X(N-1)).
         KNOTYP = 2:  Periodic placement of boundary knots:
                    T( 1 ) = T( 2 ) = X(1) - (X(N)-X(N-1));
                    T(M+4) = T(M+3) = X(N) + (X(2)-X(1))  .
         Here M=NDIM=2*N.
      If the input value of KNOTYP is negative, however, it is
      assumed that NKNOTS and T were set in a previous call.
      This option is provided for improved efficiency when used
      in a parametric setting.

NKNOTS:INOUT  is the number of knots.
      If KNOTYP.GE.0, then NKNOTS will be set to NDIM+4.
      If KNOTYP.LT.0, then NKNOTS is an input variable, and an
         error return will be taken if it is not equal to NDIM+4.

T:INOUT  is the array of 2*N+4 knots for the B-representation.
      If KNOTYP.GE.0, T will be returned by DPCHBS with the
         interior double knots equal to the X-values and the
         boundary knots set as indicated above.
      If KNOTYP.LT.0, it is assumed that T was set by a
         previous call to DPCHBS.  (This routine does **not**
         verify that T forms a legitimate knot sequence.)

BCOEF:OUT  is the array of 2*N B-spline coefficients.

NDIM:OUT  is the dimension of the B-spline space.  (Set to 2*N.)

KORD:OUT  is the order of the B-spline.  (Set to 4.)

IERR:OUT  is an error flag.
      Normal return:
         IERR = 0  (no errors).
      "Recoverable" errors:
         IERR = -4  if KNOTYP.GT.2 .
         IERR = -5  if KNOTYP.LT.0 and NKNOTS.NE.(2*N+4).

*Description:
    DPCHBS computes the B-spline representation of the PCH function
    determined by N,X,F,D.  To be compatible with the rest of PCHIP,
    DPCHBS includes INCFD, the increment between successive values of
    the F- and D-arrays.

    The output is the B-representation for the function:  NKNOTS, T,
    BCOEF, NDIM, KORD.

*Caution:
    Since it is assumed that the input PCH function has been
    computed by one of the other routines in the package PCHIP,
    input arguments N, X, INCFD are **not** checked for validity.

*Restrictions/assumptions:
    1. N.GE.2 .  (not checked)
    2. X(i).LT.X(i+1), i=1,...,N .  (not checked)
    3. INCFD.GT.0 .  (not checked)
    4. KNOTYP.LE.2 .  (error return if not)
   *5. NKNOTS = NDIM+4 = 2*N+4 .  (error return if not)
   *6. T(2*k+1) = T(2*k) = X(k), k=1,...,N .  (not checked)

      * Indicates this applies only if KNOTYP.LT.0 .
```

```
 *Portability:
     Argument INCFD is used only to cause the compiler to generate
     efficient code for the subscript expressions (1+(I-1)*INCFD) .
     The normal usage, in which DPCHBS is called with one-dimensional
     arrays F and D, is probably non-Fortran 77, in the strict sense,
     but it works on all systems on which DPCHBS has been tested.

 *See Also:
     PCHIC, PCHIM, or PCHSP can be used to determine an interpolating
        PCH function from a set of data.
     The B-spline routine DBVALU can be used to evaluate the
        B-representation that is output by DPCHBS.
        (See BSPDOC for more information.)

***REFERENCES  F. N. Fritsch, "Representations for parametric cubic
                 splines," Computer Aided Geometric Design 6 (1989),
                 pp.79-82.
***ROUTINES CALLED  DPCHKT, XERMSG
***REVISION HISTORY  (YYMMDD)
   870701  DATE WRITTEN
   900405  Converted Fortran to upper case.
   900405  Removed requirement that X be dimensioned N+1.
   900406  Modified to make PCHKT a subsidiary routine to simplify
           usage.  In the process, added argument INCFD to be com-
           patible with the rest of PCHIP.
   900410  Converted prologue to SLATEC 4.0 format.
   900410  Added calls to XERMSG and changed constant 3. to 3 to
           reduce single/double differences.
   900411  Added reference.
   900430  Produced double precision version.
   900501  Corrected declarations.
   930317  Minor cosmetic changes.  (FNF)
   930514  Corrected problems with dimensioning of arguments and
           clarified DESCRIPTION.  (FNF)
   930604  Removed  NKNOTS from DPCHKT call list.  (FNF)
   END PROLOGUE
```

DPCHCM

```
      SUBROUTINE DPCHCM (N, X, F, D, INCFD, SKIP, ISMON, IERR)
***BEGIN PROLOGUE  DPCHCM
***PURPOSE  Check a cubic Hermite function for monotonicity.
***LIBRARY   SLATEC (PCHIP)
***CATEGORY  E3
***TYPE      DOUBLE PRECISION (PCHCM-S, DPCHCM-D)
***KEYWORDS  CUBIC HERMITE INTERPOLATION, MONOTONE INTERPOLATION,
             PCHIP, PIECEWISE CUBIC INTERPOLATION, UTILITY ROUTINE
***AUTHOR  Fritsch, F. N., (LLNL)
             Computing & Mathematics Research Division
             Lawrence Livermore National Laboratory
             P.O. Box 808  (L-316)
             Livermore, CA  94550
             FTS 532-4275, (510) 422-4275
***DESCRIPTION

 *Usage:

        PARAMETER  (INCFD = ...)
        INTEGER  N, ISMON(N), IERR
        DOUBLE PRECISION  X(N), F(INCFD,N), D(INCFD,N)
        LOGICAL  SKIP

        CALL  DPCHCM (N, X, F, D, INCFD, SKIP, ISMON, IERR)

 *Arguments:

     N:IN  is the number of data points.  (Error return if N.LT.2 .)

     X:IN  is a real*8 array of independent variable values.  The
           elements of X must be strictly increasing:
                X(I-1) .LT. X(I),  I = 2(1)N.
           (Error return if not.)

     F:IN  is a real*8 array of function values.  F(1+(I-1)*INCFD) is
           the value corresponding to X(I).

     D:IN  is a real*8 array of derivative values.  D(1+(I-1)*INCFD) is
           is the value corresponding to X(I).

     INCFD:IN  is the increment between successive values in F and D.
           (Error return if  INCFD.LT.1 .)

     SKIP:INOUT  is a logical variable which should be set to
           .TRUE. if the user wishes to skip checks for validity of
           preceding parameters, or to .FALSE. otherwise.
           This will save time in case these checks have already
           been performed.
           SKIP will be set to .TRUE. on normal return.

     ISMON:OUT  is an integer array indicating on which intervals the
           PCH function defined by  N, X, F, D  is monotonic.
           For data interval [X(I),X(I+1)],
             ISMON(I) = -3  if function is probably decreasing;
             ISMON(I) = -1  if function is strictly decreasing;
             ISMON(I) =  0  if function is constant;
```

```
            ISMON(I) =  1  if function is strictly increasing;
            ISMON(I) =  2  if function is non-monotonic;
            ISMON(I) =  3  if function is probably increasing.
               If ABS(ISMON)=3, this means that the D-values are near
               the boundary of the monotonicity region.  A small
               increase produces non-monotonicity; decrease, strict
               monotonicity.
         The above applies to I=1(1)N-1.  ISMON(N) indicates whether
            the entire function is monotonic on [X(1),X(N)].

    IERR:OUT  is an error flag.
         Normal return:
            IERR = 0  (no errors).
         "Recoverable" errors:
            IERR = -1  if N.LT.2 .
            IERR = -2  if INCFD.LT.1 .
            IERR = -3  if the X-array is not strictly increasing.
        (The ISMON-array has not been changed in any of these cases.)
             NOTE:  The above errors are checked in the order listed,
                 and following arguments have **NOT** been validated.

 *Description:

        DPCHCM:  Piecewise Cubic Hermite -- Check Monotonicity.

    Checks the piecewise cubic Hermite function defined by  N,X,F,D
    for monotonicity.

    To provide compatibility with DPCHIM and DPCHIC, includes an
    increment between successive values of the F- and D-arrays.

 *Cautions:
    This provides the same capability as old DPCHMC, except that a
    new output value, -3, was added February 1989.  (Formerly, -3
    and +3 were lumped together in the single value 3.)  Codes that
    flag nonmonotonicity by "IF (ISMON.EQ.2)" need not be changed.
    Codes that check via "IF (ISMON.GE.3)" should change the test to
    "IF (IABS(ISMON).GE.3)".  Codes that declare monotonicity via
    "IF (ISMON.LE.1)" should change to "IF (IABS(ISMON).LE.1)".

***REFERENCES  F. N. Fritsch and R. E. Carlson, Monotone piecewise
                cubic interpolation, SIAM Journal on Numerical Ana-
                lysis 17, 2 (April 1980), pp. 238-246.
***ROUTINES CALLED  DCHFCM, XERMSG
***REVISION HISTORY  (YYMMDD)
   820518  DATE WRITTEN
   820804  Converted to SLATEC library version.
   831201  Reversed order of subscripts of F and D, so that the
           routine will work properly when INCFD.GT.1 .  (Bug!)
   870707  Corrected XERROR calls for d.p. name(s).
   890206  Corrected XERROR calls.
   890209  Added possible ISMON value of -3 and modified code so
           that 1,3,-1 produces ISMON(N)=2, rather than 3.
   890306  Added caution about changed output.
   890407  Changed name from DPCHMC to DPCHCM, as requested at the
           March 1989 SLATEC CML meeting, and made a few other
           minor modifications necessitated by this change.
   890407  Converted to new SLATEC format.
   890407  Modified DESCRIPTION to LDOC format.
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
```

```
920429  Revised format and order of references.  (WRB,FNF)
END PROLOGUE
```

DPCHFD

```
      SUBROUTINE DPCHFD (N, X, F, D, INCFD, SKIP, NE, XE, FE, DE, IERR)
***BEGIN PROLOGUE  DPCHFD
***PURPOSE  Evaluate a piecewise cubic Hermite function and its first
            derivative at an array of points.  May be used by itself
            for Hermite interpolation, or as an evaluator for DPCHIM
            or DPCHIC. If only function values are required, use
            DPCHFE instead.
***LIBRARY   SLATEC (PCHIP)
***CATEGORY  E3, H1
***TYPE      DOUBLE PRECISION (PCHFD-S, DPCHFD-D)
***KEYWORDS  CUBIC HERMITE DIFFERENTIATION, CUBIC HERMITE EVALUATION,
             HERMITE INTERPOLATION, PCHIP, PIECEWISE CUBIC EVALUATION
***AUTHOR  Fritsch, F. N., (LLNL)
             Lawrence Livermore National Laboratory
             P.O. Box 808  (L-316)
             Livermore, CA  94550
             FTS 532-4275, (510) 422-4275
***DESCRIPTION

          DPCHFD:  Piecewise Cubic Hermite Function and Derivative
                   evaluator

     Evaluates the cubic Hermite function defined by  N, X, F, D,  to-
     gether with its first derivative, at the points  XE(J), J=1(1)NE.

     If only function values are required, use DPCHFE, instead.

     To provide compatibility with DPCHIM and DPCHIC, includes an
     increment between successive values of the F- and D-arrays.

 --------------------------------------------------------------------------

  Calling sequence:

        PARAMETER  (INCFD = ...)
        INTEGER  N, NE, IERR
        DOUBLE PRECISION  X(N), F(INCFD,N), D(INCFD,N), XE(NE), FE(NE),
                          DE(NE)
        LOGICAL  SKIP

        CALL  DPCHFD (N, X, F, D, INCFD, SKIP, NE, XE, FE, DE, IERR)

   Parameters:

     N -- (input) number of data points.  (Error return if N.LT.2 .)

     X -- (input) real*8 array of independent variable values.  The
           elements of X must be strictly increasing:
                X(I-1) .LT. X(I),  I = 2(1)N.
           (Error return if not.)

     F -- (input) real*8 array of function values.  F(1+(I-1)*INCFD) is
           the value corresponding to X(I).

     D -- (input) real*8 array of derivative values.  D(1+(I-1)*INCFD)
           is the value corresponding to X(I).
```

```
      INCFD -- (input) increment between successive values in F and D.
            (Error return if  INCFD.LT.1 .)

      SKIP -- (input/output) logical variable which should be set to
            .TRUE. if the user wishes to skip checks for validity of
            preceding parameters, or to .FALSE. otherwise.
            This will save time in case these checks have already
            been performed (say, in DPCHIM or DPCHIC).
            SKIP will be set to .TRUE. on normal return.

      NE -- (input) number of evaluation points.  (Error return if
            NE.LT.1 .)

      XE -- (input) real*8 array of points at which the functions are to
            be evaluated.


           NOTES:
            1. The evaluation will be most efficient if the elements
               of XE are increasing relative to X;
               that is,   XE(J) .GE. X(I)
               implies    XE(K) .GE. X(I),  all K.GE.J .
            2. If any of the XE are outside the interval [X(1),X(N)],
               values are extrapolated from the nearest extreme cubic,
               and a warning error is returned.

      FE -- (output) real*8 array of values of the cubic Hermite
            function defined by  N, X, F, D  at the points  XE.

      DE -- (output) real*8 array of values of the first derivative of
            the same function at the points  XE.

      IERR -- (output) error flag.
            Normal return:
               IERR = 0  (no errors).
            Warning error:
               IERR.GT.0  means that extrapolation was performed at
                  IERR points.
            "Recoverable" errors:
               IERR = -1  if N.LT.2 .
               IERR = -2  if INCFD.LT.1 .
               IERR = -3  if the X-array is not strictly increasing.
               IERR = -4  if NE.LT.1 .
            (Output arrays have not been changed in any of these cases.)
                NOTE:  The above errors are checked in the order listed,
                     and following arguments have **NOT** been validated.
               IERR = -5  if an error has occurred in the lower-level
                          routine DCHFDV.  NB: this should never happen.
                          Notify the author **IMMEDIATELY** if it does.

***REFERENCES  (NONE)
***ROUTINES CALLED  DCHFDV, XERMSG
***REVISION HISTORY  (YYMMDD)
   811020  DATE WRITTEN
   820803  Minor cosmetic changes for release 1.
   870707  Corrected XERROR calls for d.p. name(s).
   890206  Corrected XERROR calls.
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
```

```
891006  Cosmetic changes to prologue.  (WRB)
891006  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)
END PROLOGUE
```

DPCHFE

```
      SUBROUTINE DPCHFE (N, X, F, D, INCFD, SKIP, NE, XE, FE, IERR)
***BEGIN PROLOGUE  DPCHFE
***PURPOSE  Evaluate a piecewise cubic Hermite function at an array of
            points.  May be used by itself for Hermite interpolation,
            or as an evaluator for DPCHIM or DPCHIC.
***LIBRARY   SLATEC (PCHIP)
***CATEGORY  E3
***TYPE      DOUBLE PRECISION (PCHFE-S, DPCHFE-D)
***KEYWORDS  CUBIC HERMITE EVALUATION, HERMITE INTERPOLATION, PCHIP,
             PIECEWISE CUBIC EVALUATION
***AUTHOR  Fritsch, F. N., (LLNL)
             Lawrence Livermore National Laboratory
             P.O. Box 808  (L-316)
             Livermore, CA  94550
             FTS 532-4275, (510) 422-4275
***DESCRIPTION

          DPCHFE:  Piecewise Cubic Hermite Function Evaluator

     Evaluates the cubic Hermite function defined by  N, X, F, D  at
     the points  XE(J), J=1(1)NE.

     To provide compatibility with DPCHIM and DPCHIC, includes an
     increment between successive values of the F- and D-arrays.

 --------------------------------------------------------------------

  Calling sequence:

        PARAMETER  (INCFD = ...)
        INTEGER  N, NE, IERR
        DOUBLE PRECISION  X(N), F(INCFD,N), D(INCFD,N), XE(NE), FE(NE)
        LOGICAL  SKIP

        CALL  DPCHFE (N, X, F, D, INCFD, SKIP, NE, XE, FE, IERR)

   Parameters:

     N -- (input) number of data points.  (Error return if N.LT.2 .)

     X -- (input) real*8 array of independent variable values.  The
           elements of X must be strictly increasing:
                X(I-1) .LT. X(I),  I = 2(1)N.
           (Error return if not.)

     F -- (input) real*8 array of function values.  F(1+(I-1)*INCFD) is
           the value corresponding to X(I).

     D -- (input) real*8 array of derivative values.  D(1+(I-1)*INCFD)
           is the value corresponding to X(I).

     INCFD -- (input) increment between successive values in F and D.
           (Error return if  INCFD.LT.1 .)

     SKIP -- (input/output) logical variable which should be set to
           .TRUE. if the user wishes to skip checks for validity of
```

```
            preceding parameters, or to .FALSE. otherwise.
            This will save time in case these checks have already
            been performed (say, in DPCHIM or DPCHIC).
            SKIP will be set to .TRUE. on normal return.

      NE -- (input) number of evaluation points.  (Error return if
            NE.LT.1 .)

      XE -- (input) real*8 array of points at which the function is to
            be evaluated.

           NOTES:
            1. The evaluation will be most efficient if the elements
               of XE are increasing relative to X;
               that is,   XE(J) .GE. X(I)
               implies    XE(K) .GE. X(I),  all K.GE.J .
            2. If any of the XE are outside the interval [X(1),X(N)],
               values are extrapolated from the nearest extreme cubic,
               and a warning error is returned.

      FE -- (output) real*8 array of values of the cubic Hermite
            function defined by  N, X, F, D  at the points  XE.

      IERR -- (output) error flag.
            Normal return:
               IERR = 0  (no errors).
            Warning error:
               IERR.GT.0  means that extrapolation was performed at
                  IERR points.
            "Recoverable" errors:
               IERR = -1  if N.LT.2 .
               IERR = -2  if INCFD.LT.1 .
               IERR = -3  if the X-array is not strictly increasing.
               IERR = -4  if NE.LT.1 .
              (The FE-array has not been changed in any of these cases.)
                NOTE:  The above errors are checked in the order listed,
                    and following arguments have **NOT** been validated.

***REFERENCES  (NONE)
***ROUTINES CALLED  DCHFEV, XERMSG
***REVISION HISTORY  (YYMMDD)
   811020  DATE WRITTEN
   820803  Minor cosmetic changes for release 1.
   870707  Corrected XERROR calls for d.p. name(s).
   890206  Corrected XERROR calls.
   890531  Changed all specific intrinsics to generic.  (WRB)
   890831  Modified array declarations.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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)
   END PROLOGUE
```

DPCHIA

```
      DOUBLE PRECISION FUNCTION DPCHIA (N, X, F, D, INCFD, SKIP, A, B,
     +   IERR)
***BEGIN PROLOGUE  DPCHIA
***PURPOSE  Evaluate the definite integral of a piecewise cubic
            Hermite function over an arbitrary interval.
***LIBRARY   SLATEC (PCHIP)
***CATEGORY  E3, H2A1B2
***TYPE      DOUBLE PRECISION (PCHIA-S, DPCHIA-D)
***KEYWORDS  CUBIC HERMITE INTERPOLATION, NUMERICAL INTEGRATION, PCHIP,
             QUADRATURE
***AUTHOR  Fritsch, F. N., (LLNL)
             Lawrence Livermore National Laboratory
             P.O. Box 808  (L-316)
             Livermore, CA  94550
             FTS 532-4275, (510) 422-4275
***DESCRIPTION

          DPCHIA:  Piecewise Cubic Hermite Integrator, Arbitrary limits

      Evaluates the definite integral of the cubic Hermite function
      defined by  N, X, F, D  over the interval [A, B].

      To provide compatibility with DPCHIM and DPCHIC, includes an
      increment between successive values of the F- and D-arrays.

 ----------------------------------------------------------------------

  Calling sequence:

        PARAMETER  (INCFD = ...)
        INTEGER  N, IERR
        DOUBLE PRECISION  X(N), F(INCFD,N), D(INCFD,N), A, B
        DOUBLE PRECISION  VALUE, DPCHIA
        LOGICAL  SKIP

        VALUE = DPCHIA (N, X, F, D, INCFD, SKIP, A, B, IERR)

   Parameters:

      VALUE -- (output) value of the requested integral.

      N -- (input) number of data points.  (Error return if N.LT.2 .)

      X -- (input) real*8 array of independent variable values.  The
            elements of X must be strictly increasing:
                 X(I-1) .LT. X(I),  I = 2(1)N.
            (Error return if not.)

      F -- (input) real*8 array of function values.  F(1+(I-1)*INCFD) is
            the value corresponding to X(I).

      D -- (input) real*8 array of derivative values.  D(1+(I-1)*INCFD)
            is the value corresponding to X(I).

      INCFD -- (input) increment between successive values in F and D.
            (Error return if  INCFD.LT.1 .)
```

```
      SKIP -- (input/output) logical variable which should be set to
            .TRUE. if the user wishes to skip checks for validity of
            preceding parameters, or to .FALSE. otherwise.
            This will save time in case these checks have already
            been performed (say, in DPCHIM or DPCHIC).
            SKIP will be set to .TRUE. on return with IERR.GE.0 .

      A,B -- (input) the limits of integration.
            NOTE:  There is no requirement that [A,B] be contained in
                   [X(1),X(N)].  However, the resulting integral value
                   will be highly suspect, if not.

      IERR -- (output) error flag.
            Normal return:
               IERR = 0  (no errors).
            Warning errors:
               IERR = 1  if  A  is outside the interval [X(1),X(N)].
               IERR = 2  if  B  is outside the interval [X(1),X(N)].
               IERR = 3  if both of the above are true.  (Note that this
                         means that either [A,B] contains data interval
                         or the intervals do not intersect at all.)
            "Recoverable" errors:
               IERR = -1  if N.LT.2 .
               IERR = -2  if INCFD.LT.1 .
               IERR = -3  if the X-array is not strictly increasing.
                 (VALUE will be zero in any of these cases.)
                NOTE:  The above errors are checked in the order listed,
                    and following arguments have **NOT** been validated.
               IERR = -4  in case of an error return from DPCHID (which
                          should never occur).

***REFERENCES  (NONE)
***ROUTINES CALLED  DCHFIE, DPCHID, XERMSG
***REVISION HISTORY  (YYMMDD)
   820730  DATE WRITTEN
   820804  Converted to SLATEC library version.
   870707  Corrected XERROR calls for d.p. name(s).
   870707  Corrected conversion to double precision.
   870813  Minor cosmetic changes.
   890206  Corrected XERROR calls.
   890411  Added SAVE statements (Vers. 3.2).
   890531  Changed all specific intrinsics to generic.  (WRB)
   890703  Corrected category record.  (WRB)
   890831  Modified array declarations.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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)
   930503  Corrected to set VALUE=0 when IERR.lt.0.  (FNF)
   930504  Changed DCHFIV to DCHFIE.  (FNF)
   END PROLOGUE
```

DPCHIC

```
      SUBROUTINE DPCHIC (IC, VC, SWITCH, N, X, F, D, INCFD, WK, NWK,
     +   IERR)
***BEGIN PROLOGUE  DPCHIC
***PURPOSE  Set derivatives needed to determine a piecewise monotone
            piecewise cubic Hermite interpolant to given data.
            User control is available over boundary conditions and/or
            treatment of points where monotonicity switches direction.
***LIBRARY   SLATEC (PCHIP)
***CATEGORY  E1A
***TYPE      DOUBLE PRECISION (PCHIC-S, DPCHIC-D)
***KEYWORDS  CUBIC HERMITE INTERPOLATION, MONOTONE INTERPOLATION,
             PCHIP, PIECEWISE CUBIC INTERPOLATION,
             SHAPE-PRESERVING INTERPOLATION
***AUTHOR  Fritsch, F. N., (LLNL)
             Lawrence Livermore National Laboratory
             P.O. Box 808  (L-316)
             Livermore, CA  94550
             FTS 532-4275, (510) 422-4275
***DESCRIPTION

         DPCHIC:  Piecewise Cubic Hermite Interpolation Coefficients.

     Sets derivatives needed to determine a piecewise monotone piece-
     wise cubic interpolant to the data given in X and F satisfying the
     boundary conditions specified by IC and VC.

     The treatment of points where monotonicity switches direction is
     controlled by argument SWITCH.

     To facilitate two-dimensional applications, includes an increment
     between successive values of the F- and D-arrays.

     The resulting piecewise cubic Hermite function may be evaluated
     by DPCHFE or DPCHFD.

 ----------------------------------------------------------------------

  Calling sequence:

        PARAMETER  (INCFD = ...)
        INTEGER  IC(2), N, NWK, IERR
        DOUBLE PRECISION  VC(2), SWITCH, X(N), F(INCFD,N), D(INCFD,N),
                          WK(NWK)

        CALL DPCHIC (IC, VC, SWITCH, N, X, F, D, INCFD, WK, NWK, IERR)

   Parameters:

     IC -- (input) integer array of length 2 specifying desired
           boundary conditions:
           IC(1) = IBEG, desired condition at beginning of data.
           IC(2) = IEND, desired condition at end of data.

           IBEG = 0  for the default boundary condition (the same as
                     used by DPCHIM).
           If IBEG.NE.0, then its sign indicates whether the boundary
```

```
                derivative is to be adjusted, if necessary, to be
                compatible with monotonicity:
            IBEG.GT.0  if no adjustment is to be performed.
            IBEG.LT.0  if the derivative is to be adjusted for
                monotonicity.

        Allowable values for the magnitude of IBEG are:
        IBEG = 1  if first derivative at X(1) is given in VC(1).
        IBEG = 2  if second derivative at X(1) is given in VC(1).
        IBEG = 3  to use the 3-point difference formula for D(1).
                  (Reverts to the default b.c. if N.LT.3 .)
        IBEG = 4  to use the 4-point difference formula for D(1).
                  (Reverts to the default b.c. if N.LT.4 .)
        IBEG = 5  to set D(1) so that the second derivative is con-
            tinuous at X(2). (Reverts to the default b.c. if N.LT.4.)
            This option is somewhat analogous to the "not a knot"
            boundary condition provided by DPCHSP.

       NOTES (IBEG):
        1. An error return is taken if ABS(IBEG).GT.5 .
        2. Only in case  IBEG.LE.0  is it guaranteed that the
            interpolant will be monotonic in the first interval.
            If the returned value of D(1) lies between zero and
            3*SLOPE(1), the interpolant will be monotonic.  This
            is **NOT** checked if IBEG.GT.0 .
        3. If IBEG.LT.0 and D(1) had to be changed to achieve mono-
            tonicity, a warning error is returned.

        IEND may take on the same values as IBEG, but applied to
        derivative at X(N).  In case IEND = 1 or 2, the value is
        given in VC(2).

       NOTES (IEND):
        1. An error return is taken if ABS(IEND).GT.5 .
        2. Only in case  IEND.LE.0  is it guaranteed that the
            interpolant will be monotonic in the last interval.
            If the returned value of D(1+(N-1)*INCFD) lies between
            zero and 3*SLOPE(N-1), the interpolant will be monotonic.
            This is **NOT** checked if IEND.GT.0 .
        3. If IEND.LT.0 and D(1+(N-1)*INCFD) had to be changed to
            achieve monotonicity, a warning error is returned.

VC -- (input) real*8 array of length 2 specifying desired boundary
      values, as indicated above.
      VC(1) need be set only if IC(1) = 1 or 2 .
      VC(2) need be set only if IC(2) = 1 or 2 .

SWITCH -- (input) indicates desired treatment of points where
      direction of monotonicity switches:
      Set SWITCH to zero if interpolant is required to be mono-
      tonic in each interval, regardless of monotonicity of data.
        NOTES:
         1. This will cause D to be set to zero at all switch
            points, thus forcing extrema there.
         2. The result of using this option with the default boun-
            dary conditions will be identical to using DPCHIM, but
            will generally cost more compute time.
            This option is provided only to facilitate comparison
            of different switch and/or boundary conditions.
      Set SWITCH nonzero to use a formula based on the 3-point
```

```
         difference formula in the vicinity of switch points.
      If SWITCH is positive, the interpolant on each interval
         containing an extremum is controlled to not deviate from
         the data by more than SWITCH*DFLOC, where DFLOC is the
         maximum of the change of F on this interval and its two
         immediate neighbors.
      If SWITCH is negative, no such control is to be imposed.

N -- (input) number of data points.  (Error return if N.LT.2 .)

X -- (input) real*8 array of independent variable values.  The
      elements of X must be strictly increasing:
           X(I-1) .LT. X(I),  I = 2(1)N.
      (Error return if not.)

F -- (input) real*8 array of dependent variable values to be
      interpolated.  F(1+(I-1)*INCFD) is value corresponding to
      X(I).

D -- (output) real*8 array of derivative values at the data
      points.  These values will determine a monotone cubic
      Hermite function on each subinterval on which the data
      are monotonic, except possibly adjacent to switches in
      monotonicity. The value corresponding to X(I) is stored in
           D(1+(I-1)*INCFD),  I=1(1)N.
      No other entries in D are changed.

INCFD -- (input) increment between successive values in F and D.
      This argument is provided primarily for 2-D applications.
      (Error return if  INCFD.LT.1 .)

WK -- (scratch) real*8 array of working storage.  The user may
      wish to know that the returned values are:
         WK(I)     = H(I)     = X(I+1) - X(I) ;
         WK(N-1+I) = SLOPE(I) = (F(1,I+1) - F(1,I)) / H(I)
      for  I = 1(1)N-1.

NWK -- (input) length of work array.
      (Error return if  NWK.LT.2*(N-1) .)

IERR -- (output) error flag.
      Normal return:
         IERR = 0  (no errors).
      Warning errors:
         IERR = 1  if IBEG.LT.0 and D(1) had to be adjusted for
                   monotonicity.
         IERR = 2  if IEND.LT.0 and D(1+(N-1)*INCFD) had to be
                   adjusted for monotonicity.
         IERR = 3  if both of the above are true.
      "Recoverable" errors:
         IERR = -1  if N.LT.2 .
         IERR = -2  if INCFD.LT.1 .
         IERR = -3  if the X-array is not strictly increasing.
         IERR = -4  if ABS(IBEG).GT.5 .
         IERR = -5  if ABS(IEND).GT.5 .
         IERR = -6  if both of the above are true.
         IERR = -7  if NWK.LT.2*(N-1) .
        (The D-array has not been changed in any of these cases.)
          NOTE:  The above errors are checked in the order listed,
               and following arguments have **NOT** been validated.
```

```
***REFERENCES  1. F. N. Fritsch, Piecewise Cubic Hermite Interpolation
                  Package, Report UCRL-87285, Lawrence Livermore Natio-
                  nal Laboratory, July 1982.  [Poster presented at the
                  SIAM 30th Anniversary Meeting, 19-23 July 1982.]
                2. F. N. Fritsch and J. Butland, A method for construc-
                  ting local monotone piecewise cubic interpolants, SIAM
                  Journal on Scientific and Statistical Computing 5, 2
                  (June 1984), pp. 300-304.
                3. F. N. Fritsch and R. E. Carlson, Monotone piecewise
                  cubic interpolation, SIAM Journal on Numerical Ana-
                  lysis 17, 2 (April 1980), pp. 238-246.
***ROUTINES CALLED  DPCHCE, DPCHCI, DPCHCS, XERMSG
***REVISION HISTORY  (YYMMDD)
   820218  DATE WRITTEN
   820804  Converted to SLATEC library version.
   870707  Corrected XERROR calls for d.p. name(s).
   870813  Updated Reference 2.
   890206  Corrected XERROR calls.
   890411  Added SAVE statements (Vers. 3.2).
   890531  Changed all specific intrinsics to generic.  (WRB)
   890703  Corrected category record.  (WRB)
   890831  Modified array declarations.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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)
   920429  Revised format and order of references.  (WRB,FNF)
   END PROLOGUE
```

DPCHID

```
      DOUBLE PRECISION FUNCTION DPCHID (N, X, F, D, INCFD, SKIP, IA, IB,
     +   IERR)
***BEGIN PROLOGUE  DPCHID
***PURPOSE  Evaluate the definite integral of a piecewise cubic
            Hermite function over an interval whose endpoints are data
            points.
***LIBRARY   SLATEC (PCHIP)
***CATEGORY  E3, H2A1B2
***TYPE      DOUBLE PRECISION (PCHID-S, DPCHID-D)
***KEYWORDS  CUBIC HERMITE INTERPOLATION, NUMERICAL INTEGRATION, PCHIP,
             QUADRATURE
***AUTHOR  Fritsch, F. N., (LLNL)
             Lawrence Livermore National Laboratory
             P.O. Box 808  (L-316)
             Livermore, CA  94550
             FTS 532-4275, (510) 422-4275
***DESCRIPTION

          DPCHID:  Piecewise Cubic Hermite Integrator, Data limits

      Evaluates the definite integral of the cubic Hermite function
      defined by  N, X, F, D  over the interval [X(IA), X(IB)].

      To provide compatibility with DPCHIM and DPCHIC, includes an
      increment between successive values of the F- and D-arrays.

 ----------------------------------------------------------------------

  Calling sequence:

        PARAMETER  (INCFD = ...)
        INTEGER  N, IA, IB, IERR
        DOUBLE PRECISION  X(N), F(INCFD,N), D(INCFD,N)
        LOGICAL  SKIP

        VALUE = DPCHID (N, X, F, D, INCFD, SKIP, IA, IB, IERR)

   Parameters:

      VALUE -- (output) value of the requested integral.

      N -- (input) number of data points.  (Error return if N.LT.2 .)

      X -- (input) real*8 array of independent variable values.  The
            elements of X must be strictly increasing:
                 X(I-1) .LT. X(I),  I = 2(1)N.
            (Error return if not.)

      F -- (input) real*8 array of function values.  F(1+(I-1)*INCFD) is
            the value corresponding to X(I).

      D -- (input) real*8 array of derivative values.  D(1+(I-1)*INCFD)
            is the value corresponding to X(I).

      INCFD -- (input) increment between successive values in F and D.
            (Error return if  INCFD.LT.1 .)
```

```
     SKIP -- (input/output) logical variable which should be set to
           .TRUE. if the user wishes to skip checks for validity of
           preceding parameters, or to .FALSE. otherwise.
           This will save time in case these checks have already
           been performed (say, in DPCHIM or DPCHIC).
           SKIP will be set to .TRUE. on return with IERR = 0 or -4.

     IA,IB -- (input) indices in X-array for the limits of integration.
           both must be in the range [1,N].  (Error return if not.)
           No restrictions on their relative values.

     IERR -- (output) error flag.
           Normal return:
              IERR = 0  (no errors).
           "Recoverable" errors:
              IERR = -1  if N.LT.2 .
              IERR = -2  if INCFD.LT.1 .
              IERR = -3  if the X-array is not strictly increasing.
              IERR = -4  if IA or IB is out of range.
                (VALUE will be zero in any of these cases.)
               NOTE:  The above errors are checked in the order listed,
                   and following arguments have **NOT** been validated.

***REFERENCES  (NONE)
***ROUTINES CALLED  XERMSG
***REVISION HISTORY  (YYMMDD)
   820723  DATE WRITTEN
   820804  Converted to SLATEC library version.
   870707  Corrected XERROR calls for d.p. name(s).
   870813  Minor cosmetic changes.
   890206  Corrected XERROR calls.
   890411  Added SAVE statements (Vers. 3.2).
   890531  Changed all specific intrinsics to generic.  (WRB)
   890703  Corrected category record.  (WRB)
   890831  Modified array declarations.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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)
   930504  Corrected to set VALUE=0 when IERR.ne.0.  (FNF)
   END PROLOGUE
```

DPCHIM

```
      SUBROUTINE DPCHIM (N, X, F, D, INCFD, IERR)
***BEGIN PROLOGUE  DPCHIM
***PURPOSE  Set derivatives needed to determine a monotone piecewise
            cubic Hermite interpolant to given data.  Boundary values
            are provided which are compatible with monotonicity.  The
            interpolant will have an extremum at each point where mono-
            tonicity switches direction.  (See DPCHIC if user control
            is desired over boundary or switch conditions.)
***LIBRARY   SLATEC (PCHIP)
***CATEGORY  E1A
***TYPE      DOUBLE PRECISION (PCHIM-S, DPCHIM-D)
***KEYWORDS  CUBIC HERMITE INTERPOLATION, MONOTONE INTERPOLATION,
             PCHIP, PIECEWISE CUBIC INTERPOLATION
***AUTHOR  Fritsch, F. N., (LLNL)
             Lawrence Livermore National Laboratory
             P.O. Box 808  (L-316)
             Livermore, CA  94550
             FTS 532-4275, (510) 422-4275
***DESCRIPTION

          DPCHIM:  Piecewise Cubic Hermite Interpolation to
                   Monotone data.

     Sets derivatives needed to determine a monotone piecewise cubic
     Hermite interpolant to the data given in X and F.

     Default boundary conditions are provided which are compatible
     with monotonicity.  (See DPCHIC if user control of boundary con-
     ditions is desired.)

     If the data are only piecewise monotonic, the interpolant will
     have an extremum at each point where monotonicity switches direc-
     tion.  (See DPCHIC if user control is desired in such cases.)

     To facilitate two-dimensional applications, includes an increment
     between successive values of the F- and D-arrays.

     The resulting piecewise cubic Hermite function may be evaluated
     by DPCHFE or DPCHFD.

 --------------------------------------------------------------------------

  Calling sequence:

        PARAMETER  (INCFD = ...)
        INTEGER  N, IERR
        DOUBLE PRECISION  X(N), F(INCFD,N), D(INCFD,N)

        CALL  DPCHIM (N, X, F, D, INCFD, IERR)

   Parameters:

     N -- (input) number of data points.  (Error return if N.LT.2 .)
           If N=2, simply does linear interpolation.

     X -- (input) real*8 array of independent variable values.  The
```

```
            elements of X must be strictly increasing:
                 X(I-1) .LT. X(I),  I = 2(1)N.
            (Error return if not.)

      F -- (input) real*8 array of dependent variable values to be
            interpolated.  F(1+(I-1)*INCFD) is value corresponding to
            X(I).  DPCHIM is designed for monotonic data, but it will
            work for any F-array.  It will force extrema at points where
            monotonicity switches direction.  If some other treatment of
            switch points is desired, DPCHIC should be used instead.
                                                 -----
      D -- (output) real*8 array of derivative values at the data
            points.  If the data are monotonic, these values will
            determine a monotone cubic Hermite function.
            The value corresponding to X(I) is stored in
                 D(1+(I-1)*INCFD),  I=1(1)N.
            No other entries in D are changed.

      INCFD -- (input) increment between successive values in F and D.
            This argument is provided primarily for 2-D applications.
            (Error return if  INCFD.LT.1 .)

      IERR -- (output) error flag.
            Normal return:
               IERR = 0  (no errors).
            Warning error:
               IERR.GT.0  means that IERR switches in the direction
                  of monotonicity were detected.
            "Recoverable" errors:
               IERR = -1  if N.LT.2 .
               IERR = -2  if INCFD.LT.1 .
               IERR = -3  if the X-array is not strictly increasing.
              (The D-array has not been changed in any of these cases.)
                NOTE:  The above errors are checked in the order listed,
                     and following arguments have **NOT** been validated.

***REFERENCES  1. F. N. Fritsch and J. Butland, A method for construc-
                  ting local monotone piecewise cubic interpolants, SIAM
                  Journal on Scientific and Statistical Computing 5, 2
                  (June 1984), pp. 300-304.
               2. F. N. Fritsch and R. E. Carlson, Monotone piecewise
                  cubic interpolation, SIAM Journal on Numerical Ana-
                  lysis 17, 2 (April 1980), pp. 238-246.
***ROUTINES CALLED  DPCHST, XERMSG
***REVISION HISTORY  (YYMMDD)
   811103  DATE WRITTEN
   820201  1. Introduced  DPCHST  to reduce possible over/under-
             flow problems.
           2. Rearranged derivative formula for same reason.
   820602  1. Modified end conditions to be continuous functions
             of data when monotonicity switches in next interval.
           2. Modified formulas so end conditions are less prone
             of over/underflow problems.
   820803  Minor cosmetic changes for release 1.
   870707  Corrected XERROR calls for d.p. name(s).
   870813  Updated Reference 1.
   890206  Corrected XERROR calls.
   890411  Added SAVE statements (Vers. 3.2).
   890531  Changed all specific intrinsics to generic.  (WRB)
   890703  Corrected category record.  (WRB)
```

```
890831  Modified array declarations.  (WRB)
891006  Cosmetic changes to prologue.  (WRB)
891006  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)
920429  Revised format and order of references.  (WRB,FNF)
END PROLOGUE
```

DPCHSP

```
      SUBROUTINE DPCHSP (IC, VC, N, X, F, D, INCFD, WK, NWK, IERR)
***BEGIN PROLOGUE  DPCHSP
***PURPOSE  Set derivatives needed to determine the Hermite represen-
            tation of the cubic spline interpolant to given data, with
            specified boundary conditions.
***LIBRARY   SLATEC (PCHIP)
***CATEGORY  E1A
***TYPE      DOUBLE PRECISION (PCHSP-S, DPCHSP-D)
***KEYWORDS  CUBIC HERMITE INTERPOLATION, PCHIP,
             PIECEWISE CUBIC INTERPOLATION, SPLINE INTERPOLATION
***AUTHOR  Fritsch, F. N., (LLNL)
             Lawrence Livermore National Laboratory
             P.O. Box 808  (L-316)
             Livermore, CA  94550
             FTS 532-4275, (510) 422-4275
***DESCRIPTION

          DPCHSP:   Piecewise Cubic Hermite Spline

     Computes the Hermite representation of the cubic spline inter-
     polant to the data given in X and F satisfying the boundary
     conditions specified by IC and VC.

     To facilitate two-dimensional applications, includes an increment
     between successive values of the F- and D-arrays.

     The resulting piecewise cubic Hermite function may be evaluated
     by DPCHFE or DPCHFD.

     NOTE:  This is a modified version of C. de Boor's cubic spline
            routine CUBSPL.

 ----------------------------------------------------------------------

  Calling sequence:

        PARAMETER  (INCFD = ...)
        INTEGER  IC(2), N, NWK, IERR
        DOUBLE PRECISION  VC(2), X(N), F(INCFD,N), D(INCFD,N), WK(NWK)

        CALL  DPCHSP (IC, VC, N, X, F, D, INCFD, WK, NWK, IERR)

   Parameters:

     IC -- (input) integer array of length 2 specifying desired
           boundary conditions:
           IC(1) = IBEG, desired condition at beginning of data.
           IC(2) = IEND, desired condition at end of data.

           IBEG = 0  to set D(1) so that the third derivative is con-
              tinuous at X(2).  This is the "not a knot" condition
              provided by de Boor's cubic spline routine CUBSPL.
              < This is the default boundary condition. >
           IBEG = 1  if first derivative at X(1) is given in VC(1).
           IBEG = 2  if second derivative at X(1) is given in VC(1).
           IBEG = 3  to use the 3-point difference formula for D(1).
```

```
                (Reverts to the default b.c. if N.LT.3 .)
      IBEG = 4  to use the 4-point difference formula for D(1).
                (Reverts to the default b.c. if N.LT.4 .)
     NOTES:
      1. An error return is taken if IBEG is out of range.
      2. For the "natural" boundary condition, use IBEG=2 and
         VC(1)=0.

      IEND may take on the same values as IBEG, but applied to
      derivative at X(N).  In case IEND = 1 or 2, the value is
      given in VC(2).

     NOTES:
      1. An error return is taken if IEND is out of range.
      2. For the "natural" boundary condition, use IEND=2 and
         VC(2)=0.

VC -- (input) real*8 array of length 2 specifying desired boundary
      values, as indicated above.
      VC(1) need be set only if IC(1) = 1 or 2 .
      VC(2) need be set only if IC(2) = 1 or 2 .

N -- (input) number of data points.  (Error return if N.LT.2 .)

X -- (input) real*8 array of independent variable values.  The
      elements of X must be strictly increasing:
           X(I-1) .LT. X(I),  I = 2(1)N.
      (Error return if not.)

F -- (input) real*8 array of dependent variable values to be
      interpolated.  F(1+(I-1)*INCFD) is value corresponding to
      X(I).

D -- (output) real*8 array of derivative values at the data
      points.  These values will determine the cubic spline
      interpolant with the requested boundary conditions.
      The value corresponding to X(I) is stored in
           D(1+(I-1)*INCFD),  I=1(1)N.
      No other entries in D are changed.

INCFD -- (input) increment between successive values in F and D.
      This argument is provided primarily for 2-D applications.
      (Error return if  INCFD.LT.1 .)

WK -- (scratch) real*8 array of working storage.

NWK -- (input) length of work array.
      (Error return if NWK.LT.2*N .)

IERR -- (output) error flag.
      Normal return:
         IERR = 0  (no errors).
      "Recoverable" errors:
         IERR = -1  if N.LT.2 .
         IERR = -2  if INCFD.LT.1 .
         IERR = -3  if the X-array is not strictly increasing.
         IERR = -4  if IBEG.LT.0 or IBEG.GT.4 .
         IERR = -5  if IEND.LT.0 of IEND.GT.4 .
         IERR = -6  if both of the above are true.
         IERR = -7  if NWK is too small.
```

```
                NOTE:  The above errors are checked in the order listed,
                     and following arguments have **NOT** been validated.
              (The D-array has not been changed in any of these cases.)
               IERR = -8  in case of trouble solving the linear system
                          for the interior derivative values.
              (The D-array may have been changed in this case.)
              (             Do **NOT** use it!                )

***REFERENCES  Carl de Boor, A Practical Guide to Splines, Springer-
                 Verlag, New York, 1978, pp. 53-59.
***ROUTINES CALLED  DPCHDF, XERMSG
***REVISION HISTORY  (YYMMDD)
   820503  DATE WRITTEN
   820804  Converted to SLATEC library version.
   870707  Corrected XERROR calls for d.p. name(s).
   890206  Corrected XERROR calls.
   890411  Added SAVE statements (Vers. 3.2).
   890703  Corrected category record.  (WRB)
   890831  Modified array declarations.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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)
   920429  Revised format and order of references.  (WRB,FNF)
   END PROLOGUE
```

DPCOEF

```
      SUBROUTINE DPCOEF (L, C, TC, A)
***BEGIN PROLOGUE  DPCOEF
***PURPOSE  Convert the DPOLFT coefficients to Taylor series form.
***LIBRARY   SLATEC
***CATEGORY  K1A1A2
***TYPE      DOUBLE PRECISION (PCOEF-S, DPCOEF-D)
***KEYWORDS  CURVE FITTING, DATA FITTING, LEAST SQUARES, POLYNOMIAL FIT
***AUTHOR  Shampine, L. F., (SNLA)
           Davenport, S. M., (SNLA)
***DESCRIPTION

      Abstract

      DPOLFT  computes the least squares polynomial fit of degree  L  as
      a sum of orthogonal polynomials.  DPCOEF  changes this fit to its
      Taylor expansion about any point  C , i.e. writes the polynomial
      as a sum of powers of (X-C).  Taking  C=0.  gives the polynomial
      in powers of X, but a suitable non-zero  C  often leads to
      polynomials which are better scaled and more accurately evaluated.

      The parameters for  DPCOEF  are

      INPUT -- All TYPE REAL variables are DOUBLE PRECISION
          L -      Indicates the degree of polynomial to be changed to
                   its Taylor expansion.  To obtain the Taylor
                   coefficients in reverse order, input  L  as the
                   negative of the degree desired.  The absolute value
                   of L  must be less than or equal to NDEG, the highest
                   degree polynomial fitted by  DPOLFT .
          C -      The point about which the Taylor expansion is to be
                   made.
          A -      Work and output array containing values from last
                   call to  DPOLFT .

      OUTPUT -- All TYPE REAL variables are DOUBLE PRECISION
          TC -     Vector containing the first LL+1 Taylor coefficients
                   where LL=ABS(L).  If  L.GT.0 , the coefficients are
                   in the usual Taylor series order, i.e.
                     P(X) = TC(1) + TC(2)*(X-C) + ... + TC(N+1)*(X-C)**N
                   If L .LT. 0, the coefficients are in reverse order,
                   i.e.
                     P(X) = TC(1)*(X-C)**N + ... + TC(N)*(X-C) + TC(N+1)

***REFERENCES  L. F. Shampine, S. M. Davenport and R. E. Huddleston,
                 Curve fitting by polynomials in one variable, Report
                 SLA-74-0270, Sandia Laboratories, June 1974.
***ROUTINES CALLED  DP1VLU
***REVISION HISTORY  (YYMMDD)
   740601  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DPFQAD

```
      SUBROUTINE DPFQAD (F, LDC, C, XI, LXI, K, ID, X1, X2, TOL, QUAD,
     +   IERR)
***BEGIN PROLOGUE  DPFQAD
***PURPOSE  Compute the integral on (X1,X2) of a product of a
            function F and the ID-th derivative of a B-spline,
            (PP-representation).
***LIBRARY   SLATEC
***CATEGORY  H2A2A1, E3, K6
***TYPE      DOUBLE PRECISION (PFQAD-S, DPFQAD-D)
***KEYWORDS  B-SPLINE, DATA FITTING, INTERPOLATION, QUADRATURE, SPLINES
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Abstract    **** a double precision routine ****
          DPFQAD computes the integral on (X1,X2) of a product of a
          function F and the ID-th derivative of a B-spline, using the
          PP-representation (C,XI,LXI,K).  (X1,X2) is normally a sub-
          interval of XI(1) .LE. X .LE. XI(LXI+1).  An integration
          routine, DPPGQ8 (a modification of GAUS8), integrates the
          product on subintervals of (X1,X2) formed by the included
          break points.  Integration outside of (XI(1),XI(LXI+1)) is
          permitted provided F is defined.

          The maximum number of significant digits obtainable in
          DBSQAD is the smaller of 18 and the number of digits
          carried in double precision arithmetic.

      Description of arguments
          Input      F,C,XI,X1,X2,TOL are double precision
            F      - external function of one argument for the
                     integrand PF(X)=F(X)*DPPVAL(LDC,C,XI,LXI,K,ID,X,
                     INPPV)
            LDC    - leading dimension of matrix C, LDC .GE. K
            C(I,J) - right Taylor derivatives at XI(J), I=1,K , J=1,LXI
            XI(*)  - break point array of length LXI+1
            LXI    - number of polynomial pieces
            K      - order of B-spline, K .GE. 1
            ID     - order of the spline derivative, 0 .LE. ID .LE. K-1
                     ID=0 gives the spline function
            X1,X2  - end points of quadrature interval, normally in
                     XI(1) .LE. X .LE. XI(LXI+1)
            TOL    - desired accuracy for the quadrature, suggest
                     10.*DTOL .LT. TOL .LE. 0.1 where DTOL is the
                     maximum of 1.0D-18 and double precision unit
                     roundoff for the machine = D1MACH(4)

          Output     QUAD is double precision
            QUAD   - integral of PF(X) on (X1,X2)
            IERR   - a status code
                     IERR=1 normal return
                          2 some quadrature does not meet the
                            requested tolerance

      Error Conditions
          Improper input is a fatal error.
          Some quadrature does not meet the requested tolerance.
```

```
***REFERENCES  D. E. Amos, Quadrature subroutines for splines and
                 B-splines, Report SAND79-1825, Sandia Laboratories,
                 December 1979.
***ROUTINES CALLED  D1MACH, DINTRV, DPPGQ8, XERMSG
***REVISION HISTORY  (YYMMDD)
   800901  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)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DPLINT

```
      SUBROUTINE DPLINT (N, X, Y, C)
***BEGIN PROLOGUE  DPLINT
***PURPOSE  Produce the polynomial which interpolates a set of discrete
            data points.
***LIBRARY   SLATEC
***CATEGORY  E1B
***TYPE      DOUBLE PRECISION (POLINT-S, DPLINT-D)
***KEYWORDS  POLYNOMIAL INTERPOLATION
***AUTHOR  Huddleston, R. E., (SNLL)
***DESCRIPTION

      Abstract
         Subroutine DPLINT is designed to produce the polynomial which
      interpolates the data  (X(I),Y(I)), I=1,...,N.  DPLINT sets up
      information in the array C which can be used by subroutine DPOLVL
      to evaluate the polynomial and its derivatives and by subroutine
      DPOLCF to produce the coefficients.

      Formal Parameters
      *** All TYPE REAL variables are DOUBLE PRECISION ***
      N  - the number of data points  (N .GE. 1)
      X  - the array of abscissas (all of which must be distinct)
      Y  - the array of ordinates
      C  - an array of information used by subroutines
      *******  Dimensioning Information  *******
      Arrays X,Y, and C must be dimensioned at least N in the calling
      program.

***REFERENCES  L. F. Shampine, S. M. Davenport and R. E. Huddleston,
                 Curve fitting by polynomials in one variable, Report
                 SLA-74-0270, Sandia Laboratories, June 1974.
***ROUTINES CALLED  XERMSG
***REVISION HISTORY  (YYMMDD)
   740601  DATE WRITTEN
   891006  Cosmetic changes to prologue.  (WRB)
   891006  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
```

DPOCH

```
      DOUBLE PRECISION FUNCTION DPOCH (A, X)
***BEGIN PROLOGUE  DPOCH
***PURPOSE  Evaluate a generalization of Pochhammer's symbol.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C1, C7A
***TYPE      DOUBLE PRECISION (POCH-S, DPOCH-D)
***KEYWORDS  FNLIB, POCHHAMMER, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 Evaluate a double precision generalization of Pochhammer's symbol
 (A)-sub-X = GAMMA(A+X)/GAMMA(A) for double precision A and X.
 For X a non-negative integer, POCH(A,X) is just Pochhammer's symbol.
 This is a preliminary version that does not handle wrong arguments
 properly and may not properly handle the case when the result is
 computed to less than half of double precision.

***REFERENCES  (NONE)
***ROUTINES CALLED  D9LGMC, DFAC, DGAMMA, DGAMR, DLGAMS, DLNREL, XERMSG
***REVISION HISTORY  (YYMMDD)
   770701  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  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)
   900727  Added EXTERNAL statement.  (WRB)
   END PROLOGUE
```

DPOCH1

```
      DOUBLE PRECISION FUNCTION DPOCH1 (A, X)
***BEGIN PROLOGUE  DPOCH1
***PURPOSE  Calculate a generalization of Pochhammer's symbol starting
            from first order.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C1, C7A
***TYPE      DOUBLE PRECISION (POCH1-S, DPOCH1-D)
***KEYWORDS  FIRST ORDER, FNLIB, POCHHAMMER, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 Evaluate a double precision generalization of Pochhammer's symbol
 for double precision A and X for special situations that require
 especially accurate values when X is small in
        POCH1(A,X) = (POCH(A,X)-1)/X
                   = (GAMMA(A+X)/GAMMA(A) - 1.0)/X .
 This specification is particularly suited for stably computing
 expressions such as
        (GAMMA(A+X)/GAMMA(A) - GAMMA(B+X)/GAMMA(B))/X
             = POCH1(A,X) - POCH1(B,X)
 Note that POCH1(A,0.0) = PSI(A)

 When ABS(X) is so small that substantial cancellation will occur if
 the straightforward formula is used, we use an expansion due
 to Fields and discussed by Y. L. Luke, The Special Functions and Their
 Approximations, Vol. 1, Academic Press, 1969, page 34.

 The ratio POCH(A,X) = GAMMA(A+X)/GAMMA(A) is written by Luke as
        (A+(X-1)/2)**X * polynomial in (A+(X-1)/2)**(-2) .
 In order to maintain significance in POCH1, we write for positive a
        (A+(X-1)/2)**X = EXP(X*LOG(A+(X-1)/2)) = EXP(Q)
                       = 1.0 + Q*EXPREL(Q) .
 Likewise the polynomial is written
        POLY = 1.0 + X*POLY1(A,X) .
 Thus,
        POCH1(A,X) = (POCH(A,X) - 1) / X
                   = EXPREL(Q)*(Q/X + Q*POLY1(A,X)) + POLY1(A,X)

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DCOT, DEXPRL, DPOCH, DPSI, XERMSG
***REVISION HISTORY  (YYMMDD)
   770801  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  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)
   900727  Added EXTERNAL statement.  (WRB)
   END PROLOGUE
```

DPOCO

```
      SUBROUTINE DPOCO (A, LDA, N, RCOND, Z, INFO)
***BEGIN PROLOGUE  DPOCO
***PURPOSE  Factor a real symmetric positive definite matrix
            and estimate the condition of the matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B
***TYPE      DOUBLE 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

      DPOCO factors a double precision symmetric positive definite
      matrix and estimates the condition of the matrix.

      If  RCOND  is not needed, DPOFA is slightly faster.
      To solve  A*X = B , follow DPOCO by DPOSL.
      To compute  INVERSE(A)*C , follow DPOCO by DPOSL.
      To compute  DETERMINANT(A) , follow DPOCO by DPODI.
      To compute  INVERSE(A) , follow DPOCO by DPODI.

      On Entry

         A       DOUBLE PRECISION(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   DOUBLE PRECISION
                 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       DOUBLE PRECISION(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  DASUM, DAXPY, DDOT, DPOFA, DSCAL
***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
```

DPODI

```
      SUBROUTINE DPODI (A, LDA, N, DET, JOB)
***BEGIN PROLOGUE  DPODI
***PURPOSE  Compute the determinant and inverse of a certain real
            symmetric positive definite matrix using the factors
            computed by DPOCO, DPOFA or DQRDC.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B, D3B1B
***TYPE      DOUBLE 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

      DPODI computes the determinant and inverse of a certain
      double precision symmetric positive definite matrix (see below)
      using the factors computed by DPOCO, DPOFA or DQRDC.

      On Entry

         A       DOUBLE PRECISION(LDA, N)
                 the output  A  from DPOCO or DPOFA
                 or the output  X  from DQRDC.

         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 DPOCO or DPOFA was used to factor  A , then
                 DPODI produces the upper half of INVERSE(A) .
                 If DQRDC was used to decompose  X , then
                 DPODI 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     DOUBLE PRECISION(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 DPOCO or DPOFA 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  DAXPY, DSCAL
***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
```

DPOFA

```
      SUBROUTINE DPOFA (A, LDA, N, INFO)
***BEGIN PROLOGUE  DPOFA
***PURPOSE  Factor a real symmetric positive definite matrix.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B
***TYPE      DOUBLE PRECISION (SPOFA-S, DPOFA-D, CPOFA-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX FACTORIZATION,
             POSITIVE DEFINITE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      DPOFA factors a double precision symmetric positive definite
      matrix.

      DPOFA is usually called by DPOCO, but it can be called
      directly with a saving in time if  RCOND  is not needed.
      (time for DPOCO) = (1 + 18/N)*(time for DPOFA) .

      On Entry

         A       DOUBLE PRECISION(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  DDOT
***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
```

DPOFS

```
      SUBROUTINE DPOFS (A, LDA, N, V, ITASK, IND, WORK)
***BEGIN PROLOGUE  DPOFS
***PURPOSE  Solve a positive definite symmetric system of linear
            equations.
***LIBRARY   SLATEC
***CATEGORY  D2B1B
***TYPE      DOUBLE PRECISION (SPOFS-S, DPOFS-D, CPOFS-C)
***KEYWORDS  HERMITIAN, LINEAR EQUATIONS, POSITIVE DEFINITE, SYMMETRIC
***AUTHOR  Voorhees, E. A., (LANL)
***DESCRIPTION

    Subroutine DPOFS solves a  positive definite symmetric
    NxN system of double precision linear equations using
    LINPACK subroutines DPOCO and DPOSL.  That is, if A is an
    NxN double precision positive definite symmetric matrix and if
    X and B are double precision N-vectors, then DPOFS solves
    the equation

                            A*X=B.

    The matrix A is first factored into upper and lower tri-
    angular matrices R and R-TRANPOSE.  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 only to solve (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 DPOFS
    in this case.

  Argument Description ***

    A      DOUBLE PRECISION(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, A contains in its upper triangle an upper
               triangular matrix R such that A = (R-TRANPOSE) * 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      DOUBLE PRECISION(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   DOUBLE PRECISION(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  D1MACH, DPOCO, DPOSL, XERMSG
***REVISION HISTORY  (YYMMDD)
   800514  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
```

DPOLCF

```
      SUBROUTINE DPOLCF (XX, N, X, C, D, WORK)
***BEGIN PROLOGUE  DPOLCF
***PURPOSE  Compute the coefficients of the polynomial fit (including
            Hermite polynomial fits) produced by a previous call to
            POLINT.
***LIBRARY   SLATEC
***CATEGORY  E1B
***TYPE      DOUBLE PRECISION (POLCOF-S, DPOLCF-D)
***KEYWORDS  COEFFICIENTS, POLYNOMIAL
***AUTHOR  Huddleston, R. E., (SNLL)
***DESCRIPTION

      Abstract
         Subroutine DPOLCF computes the coefficients of the polynomial
      fit (including Hermite polynomial fits ) produced by a previous
      call to DPLINT.  The coefficients of the polynomial, expanded
      about XX, are stored in the array D. The expansion is of the form
      P(Z) = D(1) + D(2)*(Z-XX) +D(3)*((Z-XX)**2) + ... +
                                                   D(N)*((Z-XX)**(N-1)).
      Between the call to DPLINT and the call to DPOLCF the variable N
      and the arrays X and C must not be altered.

      *****  INPUT PARAMETERS
       *** All TYPE REAL variables are DOUBLE PRECISION ***

      XX   - The point about which the Taylor expansion is to be made.

      N    - ****
             *     N, X, and C must remain unchanged between the
      X    - *     call to DPLINT and the call to DPOLCF.
      C    - ****

      *****  OUTPUT PARAMETER
       *** All TYPE REAL variables are DOUBLE PRECISION ***

      D    - The array of coefficients for the Taylor expansion as
             explained in the abstract

      *****  STORAGE PARAMETER

      WORK - This is an array to provide internal working storage. It
             must be dimensioned by at least 2*N in the calling program.


      **** Note - There are two methods for evaluating the fit produced
      by DPLINT. You may call DPOLVL to perform the task, or you may
      call DPOLCF to obtain the coefficients of the Taylor expansion and
      then write your own evaluation scheme. Due to the inherent errors
      in the computations of the Taylor expansion from the Newton
      coefficients produced by DPLINT, much more accuracy may be
      expected by calling DPOLVL as opposed to writing your own scheme.

***REFERENCES  (NONE)
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   890213  DATE WRITTEN
```

```
891006  Cosmetic changes to prologue.  (WRB)
891024  Corrected KEYWORD section.  (WRB)
891024  REVISION DATE from Version 3.2
891214  Prologue converted to Version 4.0 format.  (BAB)
END PROLOGUE
```

DPOLFT

```
      SUBROUTINE DPOLFT (N, X, Y, W, MAXDEG, NDEG, EPS, R, IERR, A)
***BEGIN PROLOGUE  DPOLFT
***PURPOSE  Fit discrete data in a least squares sense by polynomials
            in one variable.
***LIBRARY   SLATEC
***CATEGORY  K1A1A2
***TYPE      DOUBLE PRECISION (POLFIT-S, DPOLFT-D)
***KEYWORDS  CURVE FITTING, DATA FITTING, LEAST SQUARES, POLYNOMIAL FIT
***AUTHOR  Shampine, L. F., (SNLA)
           Davenport, S. M., (SNLA)
           Huddleston, R. E., (SNLL)
***DESCRIPTION

      Abstract

      Given a collection of points X(I) and a set of values Y(I) which
      correspond to some function or measurement at each of the X(I),
      subroutine  DPOLFT  computes the weighted least-squares polynomial
      fits of all degrees up to some degree either specified by the user
      or determined by the routine.  The fits thus obtained are in
      orthogonal polynomial form.  Subroutine  DP1VLU  may then be
      called to evaluate the fitted polynomials and any of their
      derivatives at any point.  The subroutine  DPCOEF  may be used to
      express the polynomial fits as powers of (X-C) for any specified
      point C.

      The parameters for  DPOLFT  are

      Input -- All TYPE REAL variables are DOUBLE PRECISION
          N -      the number of data points.  The arrays X, Y and W
                   must be dimensioned at least  N  (N .GE. 1).
          X -      array of values of the independent variable.  These
                   values may appear in any order and need not all be
                   distinct.
          Y -      array of corresponding function values.
          W -      array of positive values to be used as weights.  If
                   W(1) is negative,  DPOLFT  will set all the weights
                   to 1.0, which means unweighted least squares error
                   will be minimized.  To minimize relative error, the
                   user should set the weights to:  W(I) = 1.0/Y(I)**2,
                   I = 1,...,N .
          MAXDEG - maximum degree to be allowed for polynomial fit.
                   MAXDEG  may be any non-negative integer less than  N.
                   Note -- MAXDEG  cannot be equal to  N-1  when a
                   statistical test is to be used for degree selection,
                   i.e., when input value of  EPS  is negative.
          EPS -    specifies the criterion to be used in determining
                   the degree of fit to be computed.
                   (1)  If  EPS  is input negative,  DPOLFT  chooses the
                        degree based on a statistical F test of
                        significance.  One of three possible
                        significance levels will be used:  .01, .05 or
                        .10.  If  EPS=-1.0 , the routine will
                        automatically select one of these levels based
                        on the number of data points and the maximum
                        degree to be considered.  If  EPS  is input as
```

```
                    -.01, -.05, or -.10, a significance level of
                    .01, .05, or .10, respectively, will be used.
               (2)  If  EPS  is set to 0.,  DPOLFT  computes the
                    polynomials of degrees 0 through  MAXDEG .
               (3)  If  EPS  is input positive,  EPS  is the RMS
                    error tolerance which must be satisfied by the
                    fitted polynomial.  DPOLFT  will increase the
                    degree of fit until this criterion is met or
                    until the maximum degree is reached.

     Output -- All TYPE REAL variables are DOUBLE PRECISION
         NDEG -   degree of the highest degree fit computed.
         EPS -    RMS error of the polynomial of degree  NDEG .
         R -      vector of dimension at least NDEG containing values
                  of the fit of degree  NDEG  at each of the  X(I) .
                  Except when the statistical test is used, these
                  values are more accurate than results from subroutine
                  DP1VLU  normally are.
         IERR -   error flag with the following possible values.
             1 -- indicates normal execution, i.e., either
               (1)  the input value of  EPS  was negative, and the
                    computed polynomial fit of degree  NDEG
                    satisfies the specified F test, or
               (2)  the input value of  EPS  was 0., and the fits of
                    all degrees up to  MAXDEG  are complete, or
               (3)  the input value of  EPS  was positive, and the
                    polynomial of degree  NDEG  satisfies the RMS
                    error requirement.
             2 -- invalid input parameter.  At least one of the input
                  parameters has an illegal value and must be corrected
                  before  DPOLFT  can proceed.  Valid input results
                  when the following restrictions are observed
                    N .GE. 1
                    0 .LE. MAXDEG .LE. N-1  for  EPS .GE. 0.
                    0 .LE. MAXDEG .LE. N-2  for  EPS .LT. 0.
                    W(1)=-1.0  or  W(I) .GT. 0., I=1,...,N .
             3 -- cannot satisfy the RMS error requirement with a
                  polynomial of degree no greater than  MAXDEG .  Best
                  fit found is of degree  MAXDEG .
             4 -- cannot satisfy the test for significance using
                  current value of  MAXDEG .  Statistically, the
                  best fit found is of order  NORD .  (In this case,
                  NDEG will have one of the values:  MAXDEG-2,
                  MAXDEG-1, or MAXDEG).  Using a higher value of
                  MAXDEG  may result in passing the test.
         A -      work and output array having at least 3N+3MAXDEG+3
                  locations

     Note - DPOLFT  calculates all fits of degrees up to and including
            NDEG .  Any or all of these fits can be evaluated or
            expressed as powers of (X-C) using  DP1VLU  and  DPCOEF
            after just one call to  DPOLFT .

***REFERENCES  L. F. Shampine, S. M. Davenport and R. E. Huddleston,
                 Curve fitting by polynomials in one variable, Report
                 SLA-74-0270, Sandia Laboratories, June 1974.
***ROUTINES CALLED  DP1VLU, XERMSG
***REVISION HISTORY  (YYMMDD)
   740601  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
```

```
891006  Cosmetic changes to prologue.  (WRB)
891006  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)
900911  Added variable YP to DOUBLE PRECISION declaration.  (WRB)
920501  Reformatted the REFERENCES section.  (WRB)
920527  Corrected erroneous statements in DESCRIPTION.  (WRB)
END PROLOGUE
```

DPOLVL

```
      SUBROUTINE DPOLVL (NDER, XX, YFIT, YP, N, X, C, WORK, IERR)
***BEGIN PROLOGUE  DPOLVL
***PURPOSE  Calculate the value of a polynomial and its first NDER
            derivatives where the polynomial was produced by a previous
            call to DPLINT.
***LIBRARY   SLATEC
***CATEGORY  E3
***TYPE      DOUBLE PRECISION (POLYVL-S, DPOLVL-D)
***KEYWORDS  POLYNOMIAL EVALUATION
***AUTHOR  Huddleston, R. E., (SNLL)
***DESCRIPTION

      Abstract -
         Subroutine DPOLVL calculates the value of the polynomial and
      its first NDER derivatives where the polynomial was produced by
      a previous call to DPLINT.
         The variable N and the arrays X and C must not be altered
      between the call to DPLINT and the call to DPOLVL.

      ******  Dimensioning Information *******

      YP   must be dimensioned by at least NDER
      X    must be dimensioned by at least N (see the abstract )
      C    must be dimensioned by at least N (see the abstract )
      WORK must be dimensioned by at least 2*N if NDER is .GT. 0.

      *** Note ***
        If NDER=0, neither YP nor WORK need to be dimensioned variables.
        If NDER=1, YP does not need to be a dimensioned variable.


      *****  Input parameters
        ***  All TYPE REAL variables are DOUBLE PRECISION ***

      NDER - the number of derivatives to be evaluated

      XX   - the argument at which the polynomial and its derivatives
             are to be evaluated.

      N    - *****
             *       N, X, and C must not be altered between the call
      X    - *       to DPLINT and the call to DPOLVL.
      C    - *****


      *****  Output Parameters
        ***  All TYPE REAL variables are DOUBLE PRECISION ***

      YFIT - the value of the polynomial at XX

      YP   - the derivatives of the polynomial at XX.  The derivative of
             order J at XX is stored in  YP(J) , J = 1,...,NDER.

      IERR - Output error flag with the following possible values.
           = 1  indicates normal execution
```

```
     ***** Storage Parameters

     WORK  = this is an array to provide internal working storage for
             DPOLVL.  It must be dimensioned by at least 2*N if NDER is
             .GT. 0.  If NDER=0, WORK does not need to be a dimensioned
             variable.

***REFERENCES  L. F. Shampine, S. M. Davenport and R. E. Huddleston,
                 Curve fitting by polynomials in one variable, Report
                 SLA-74-0270, Sandia Laboratories, June 1974.
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   740601  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DPOSL

```
      SUBROUTINE DPOSL (A, LDA, N, B)
***BEGIN PROLOGUE  DPOSL
***PURPOSE  Solve the real symmetric positive definite linear system
            using the factors computed by DPOCO or DPOFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B
***TYPE      DOUBLE PRECISION (SPOSL-S, DPOSL-D, CPOSL-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX, POSITIVE DEFINITE, SOLVE
***AUTHOR  Moler, C. B., (U. of New Mexico)
***DESCRIPTION

      DPOSL solves the double precision symmetric positive definite
      system A * X = B
      using the factors computed by DPOCO or DPOFA.

      On Entry

         A       DOUBLE PRECISION(LDA, N)
                 the output from DPOCO or DPOFA.

         LDA     INTEGER
                 the leading dimension of the array  A .

         N       INTEGER
                 the order of the matrix  A .

         B       DOUBLE PRECISION(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 DPOCO(A,LDA,N,RCOND,Z,INFO)
            IF (RCOND is too small .OR. INFO .NE. 0) GO TO ...
            DO 10 J = 1, P
               CALL DPOSL(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  DAXPY, DDOT
***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
```

DPPCO

```
      SUBROUTINE DPPCO (AP, N, RCOND, Z, INFO)
***BEGIN PROLOGUE  DPPCO
***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      DOUBLE 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

      DPPCO factors a double precision symmetric positive definite
      matrix stored in packed form
      and estimates the condition of the matrix.

      If  RCOND  is not needed, DPPFA is slightly faster.
      To solve  A*X = B , follow DPPCO by DPPSL.
      To compute  INVERSE(A)*C , follow DPPCO by DPPSL.
      To compute  DETERMINANT(A) , follow DPPCO by DPPDI.
      To compute  INVERSE(A) , follow DPPCO by DPPDI.

      On Entry

         AP      DOUBLE PRECISION (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   DOUBLE PRECISION
                 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       DOUBLE PRECISION(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  DASUM, DAXPY, DDOT, DPPFA, DSCAL
***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
```

DPPDI

```
      SUBROUTINE DPPDI (AP, N, DET, JOB)
***BEGIN PROLOGUE  DPPDI
***PURPOSE  Compute the determinant and inverse of a real symmetric
            positive definite matrix using factors from DPPCO or DPPFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B, D3B1B
***TYPE      DOUBLE 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

      DPPDI computes the determinant and inverse
      of a double precision symmetric positive definite matrix
      using the factors computed by DPPCO or DPPFA .

      On Entry

         AP      DOUBLE PRECISION (N*(N+1)/2)
                 the output from DPPCO or DPPFA.

         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     DOUBLE PRECISION(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 DPOCO or DPOFA 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  DAXPY, DSCAL
***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
```

DPPERM

```
      SUBROUTINE DPPERM (DX, N, IPERM, IER)
***BEGIN PROLOGUE  DPPERM
***PURPOSE  Rearrange a given array according to a prescribed
            permutation vector.
***LIBRARY   SLATEC
***CATEGORY  N8
***TYPE      DOUBLE PRECISION (SPPERM-S, DPPERM-D, IPPERM-I, HPPERM-H)
***KEYWORDS  PERMUTATION, REARRANGEMENT
***AUTHOR  McClain, M. A., (NIST)
           Rhoads, G. S., (NBS)
***DESCRIPTION

         DPPERM rearranges the data vector DX according to the
         permutation IPERM: DX(I) <--- DX(IPERM(I)).  IPERM could come
         from one of the sorting routines IPSORT, SPSORT, DPSORT or
         HPSORT.

      Description of Parameters
         DX - input/output -- double precision array of values to be
                   rearranged.
         N - input -- number of values in double precision array DX.
         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
```

DPPFA

```
      SUBROUTINE DPPFA (AP, N, INFO)
***BEGIN PROLOGUE  DPPFA
***PURPOSE  Factor a real symmetric positive definite matrix stored in
            packed form.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B
***TYPE      DOUBLE 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

     DPPFA factors a double precision symmetric positive definite
     matrix stored in packed form.

     DPPFA is usually called by DPPCO, but it can be called
     directly with a saving in time if  RCOND  is not needed.
     (time for DPPCO) = (1 + 18/N)*(time for DPPFA) .

     On Entry

        AP      DOUBLE PRECISION (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  DDOT
***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
```

DPPQAD

```
      SUBROUTINE DPPQAD (LDC, C, XI, LXI, K, X1, X2, PQUAD)
***BEGIN PROLOGUE  DPPQAD
***PURPOSE  Compute the integral on (X1,X2) of a K-th order B-spline
            using the piecewise polynomial (PP) representation.
***LIBRARY   SLATEC
***CATEGORY  H2A2A1, E3, K6
***TYPE      DOUBLE PRECISION (PPQAD-S, DPPQAD-D)
***KEYWORDS  B-SPLINE, DATA FITTING, INTERPOLATION, QUADRATURE, SPLINES
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Abstract    **** a double precision routine ****
          DPPQAD computes the integral on (X1,X2) of a K-th order
          B-spline using the piecewise polynomial representation
          (C,XI,LXI,K).  Here the Taylor expansion about the left
          end point XI(J) of the J-th interval is integrated and
          evaluated on subintervals of (X1,X2) which are formed by
          included break points.  Integration outside (XI(1),XI(LXI+1))
          is permitted.

      Description of Arguments
          Input      C,XI,X1,X2 are double precision
            LDC    - leading dimension of matrix C, LDC .GE. K
            C(I,J) - right Taylor derivatives at XI(J), I=1,K , J=1,LXI
            XI(*)  - break point array of length LXI+1
            LXI    - number of polynomial pieces
            K      - order of B-spline, K .GE. 1
            X1,X2  - end points of quadrature interval, normally in
                     XI(1) .LE. X .LE. XI(LXI+1)

          Output     PQUAD is double precision
            PQUAD  - integral of the PP representation over (X1,X2)

      Error Conditions
          Improper input is a fatal error

***REFERENCES  D. E. Amos, Quadrature subroutines for splines and
                 B-splines, Report SAND79-1825, Sandia Laboratories,
                 December 1979.
***ROUTINES CALLED  DINTRV, XERMSG
***REVISION HISTORY  (YYMMDD)
   800901  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  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
```

DPPSL

```
      SUBROUTINE DPPSL (AP, N, B)
***BEGIN PROLOGUE  DPPSL
***PURPOSE  Solve the real symmetric positive definite system using
            the factors computed by DPPCO or DPPFA.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B1B
***TYPE      DOUBLE 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

      DPPSL solves the double precision symmetric positive definite
      system A * X = B
      using the factors computed by DPPCO or DPPFA.

      On Entry

         AP      DOUBLE PRECISION (N*(N+1)/2)
                 the output from DPPCO or DPPFA.

         N       INTEGER
                 the order of the matrix  A .

         B       DOUBLE PRECISION(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 DPPCO(AP,N,RCOND,Z,INFO)
            IF (RCOND is too small .OR. INFO .NE. 0) GO TO ...
            DO 10 J = 1, P
               CALL DPPSL(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  DAXPY, DDOT
***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
```

DPPVAL

```
      DOUBLE PRECISION FUNCTION DPPVAL (LDC, C, XI, LXI, K, IDERIV, X,
     +   INPPV)
***BEGIN PROLOGUE  DPPVAL
***PURPOSE  Calculate the value of the IDERIV-th derivative of the
            B-spline from the PP-representation.
***LIBRARY   SLATEC
***CATEGORY  E3, K6
***TYPE      DOUBLE PRECISION (PPVAL-S, DPPVAL-D)
***KEYWORDS  B-SPLINE, DATA FITTING, INTERPOLATION, SPLINES
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

      Written by Carl de Boor and modified by D. E. Amos

      Abstract    **** a double precision routine ****
          DPPVAL is the PPVALU function of the reference.

          DPPVAL calculates (at X) the value of the IDERIV-th
          derivative of the B-spline from the PP-representation
          (C,XI,LXI,K).  The Taylor expansion about XI(J) for X in
          the interval XI(J) .LE. X .LT. XI(J+1) is evaluated, J=1,LXI.
          Right limiting values at X=XI(J) are obtained.  DPPVAL will
          extrapolate beyond XI(1) and XI(LXI+1).

          To obtain left limiting values (left derivatives) at XI(J)
          replace LXI by J-1 and set X=XI(J),J=2,LXI+1.

      Description of Arguments

          Input      C,XI,X are double precision
           LDC     - leading dimension of C matrix, LDC .GE. K
           C       - matrix of dimension at least (K,LXI) containing
                     right derivatives at break points XI(*).
           XI      - break point vector of length LXI+1
           LXI     - number of polynomial pieces
           K       - order of B-spline, K .GE. 1
           IDERIV  - order of the derivative, 0 .LE. IDERIV .LE. K-1
                     IDERIV=0 gives the B-spline value
           X       - argument, XI(1) .LE. X .LE. XI(LXI+1)
           INPPV   - an initialization parameter which must be set
                     to 1 the first time DPPVAL is called.

          Output     DPPVAL is double precision
           INPPV   - INPPV contains information for efficient process-
                     ing after the initial call and INPPV must not
                     be changed by the user.  Distinct splines require
                     distinct INPPV parameters.
           DPPVAL  - value of the IDERIV-th derivative at X

      Error Conditions
          Improper input is a fatal error

***REFERENCES  Carl de Boor, Package for calculating with B-splines,
                 SIAM Journal on Numerical Analysis 14, 3 (June 1977),
                 pp. 441-472.
***ROUTINES CALLED  DINTRV, XERMSG
```

```
***REVISION HISTORY  (YYMMDD)
   800901  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
```

DPSI

```
      DOUBLE PRECISION FUNCTION DPSI (X)
***BEGIN PROLOGUE  DPSI
***PURPOSE  Compute the Psi (or Digamma) function.
***LIBRARY   SLATEC (FNLIB)
***CATEGORY  C7C
***TYPE      DOUBLE PRECISION (PSI-S, DPSI-D, CPSI-C)
***KEYWORDS  DIGAMMA FUNCTION, FNLIB, PSI FUNCTION, SPECIAL FUNCTIONS
***AUTHOR  Fullerton, W., (LANL)
***DESCRIPTION

 DPSI calculates the double precision Psi (or Digamma) function for
 double precision argument X.  PSI(X) is the logarithmic derivative
 of the Gamma function of X.

 Series for PSI        on the interval  0.          to  1.00000E+00
                                        with weighted error   5.79E-32
                                         log weighted error  31.24
                                 significant figures required  30.93
                                      decimal places required  32.05


 Series for APSI       on the interval  0.          to  1.00000E-02
                                        with weighted error   7.75E-33
                                         log weighted error  32.11
                                 significant figures required  28.88
                                      decimal places required  32.71

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DCOT, DCSEVL, INITDS, XERMSG
***REVISION HISTORY  (YYMMDD)
   770601  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  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)
   900727  Added EXTERNAL statement.  (WRB)
   920618  Removed space from variable name.  (RWC, WRB)
   END PROLOGUE
```

DPSIFN

```
      SUBROUTINE DPSIFN (X, N, KODE, M, ANS, NZ, IERR)
***BEGIN PROLOGUE  DPSIFN
***PURPOSE  Compute derivatives of the Psi function.
***LIBRARY   SLATEC
***CATEGORY  C7C
***TYPE      DOUBLE PRECISION (PSIFN-S, DPSIFN-D)
***KEYWORDS  DERIVATIVES OF THE GAMMA FUNCTION, POLYGAMMA FUNCTION,
             PSI FUNCTION
***AUTHOR  Amos, D. E., (SNLA)
***DESCRIPTION

         The following definitions are used in DPSIFN:

      Definition 1
         PSI(X) = d/dx (ln(GAMMA(X)), the first derivative of
                  the log GAMMA function.
      Definition 2
                        K   K
         PSI(K,X) = d /dx (PSI(X)), the K-th derivative of PSI(X).
   ___________________________________________________________________
      DPSIFN computes a sequence of SCALED derivatives of
      the PSI function; i.e. for fixed X and M it computes
      the M-member sequence

                    ((-1)**(K+1)/GAMMA(K+1))*PSI(K,X)
                       for K = N,...,N+M-1

      where PSI(K,X) is as defined above.   For KODE=1, DPSIFN returns
      the scaled derivatives as described.  KODE=2 is operative only
      when K=0 and in that case DPSIFN returns -PSI(X) + LN(X).  That
      is, the logarithmic behavior for large X is removed when KODE=2
      and K=0.  When sums or differences of PSI functions are computed
      the logarithmic terms can be combined analytically and computed
      separately to help retain significant digits.

         Note that CALL DPSIFN(X,0,1,1,ANS) results in
                   ANS = -PSI(X)

     Input      X is DOUBLE PRECISION
           X      - Argument, X .gt. 0.0D0
           N      - First member of the sequence, 0 .le. N .le. 100
                    N=0 gives ANS(1) = -PSI(X)       for KODE=1
                                       -PSI(X)+LN(X) for KODE=2
           KODE   - Selection parameter
                    KODE=1 returns scaled derivatives of the PSI
                    function.
                    KODE=2 returns scaled derivatives of the PSI
                    function EXCEPT when N=0. In this case,
                    ANS(1) = -PSI(X) + LN(X) is returned.
           M      - Number of members of the sequence, M.ge.1

    Output     ANS is DOUBLE PRECISION
           ANS    - A vector of length at least M whose first M
                    components contain the sequence of derivatives
                    scaled according to KODE.
           NZ     - Underflow flag
```

```
                    NZ.eq.0, A normal return
                    NZ.ne.0, Underflow, last NZ components of ANS are
                             set to zero, ANS(M-K+1)=0.0, K=1,...,NZ
          IERR   - Error flag
                    IERR=0, A normal return, computation completed
                    IERR=1, Input error,     no computation
                    IERR=2, Overflow,        X too small or N+M-1 too
                            large or both
                    IERR=3, Error,           N too large. Dimensioned
                            array TRMR(NMAX) is not large enough for N

        The nominal computational accuracy is the maximum of unit
        roundoff (=D1MACH(4)) and 1.0D-18 since critical constants
        are given to only 18 digits.

        PSIFN is the single precision version of DPSIFN.

 *Long Description:

        The basic method of evaluation is the asymptotic expansion
        for large X.ge.XMIN followed by backward recursion on a two
        term recursion relation

                 W(X+1) + X**(-N-1) = W(X).

        This is supplemented by a series

                 SUM( (X+K)**(-N-1) , K=0,1,2,... )

        which converges rapidly for large N. Both XMIN and the
        number of terms of the series are calculated from the unit
        roundoff of the machine environment.

***REFERENCES  Handbook of Mathematical Functions, National Bureau
                 of Standards Applied Mathematics Series 55, edited
                 by M. Abramowitz and I. A. Stegun, equations 6.3.5,
                 6.3.18, 6.4.6, 6.4.9 and 6.4.10, pp.258-260, 1964.
               D. E. Amos, A portable Fortran subroutine for
                 derivatives of the Psi function, Algorithm 610, ACM
                 Transactions on Mathematical Software 9, 4 (1983),
                 pp. 494-502.
***ROUTINES CALLED  D1MACH, I1MACH
***REVISION HISTORY  (YYMMDD)
   820601  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   891006  Cosmetic changes to prologue.  (WRB)
   891006  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DPSORT

```
      SUBROUTINE DPSORT (DX, N, IPERM, KFLAG, IER)
***BEGIN PROLOGUE  DPSORT
***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      DOUBLE 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

   DPSORT returns the permutation vector IPERM generated by sorting
   the array DX and, optionally, rearranges the values in DX.  DX may
   be sorted in increasing or decreasing order.  A slightly modified
   quicksort algorithm is used.

   IPERM is such that DX(IPERM(I)) is the Ith value in the
   rearrangement of DX.  IPERM may be applied to another array by
   calling IPPERM, SPPERM, DPPERM or HPPERM.

   The main difference between DPSORT and its active sorting equivalent
   DSORT is that the data are referenced indirectly rather than
   directly.  Therefore, DPSORT should require approximately twice as
   long to execute as DSORT.  However, DPSORT is more general.

   Description of Parameters
      DX - input/output -- double precision array of values to be
           sorted.  If ABS(KFLAG) = 2, then the values in DX will be
           rearranged on output; otherwise, they are unchanged.
      N  - input -- number of values in array DX to be sorted.
      IPERM - output -- permutation array such that IPERM(I) is the
              index of the value in the original order of the
              DX array that is in the Ith location in the sorted
              order.
      KFLAG - input -- control parameter:
            =  2  means return the permutation vector resulting from
                  sorting DX in increasing order and sort DX also.
            =  1  means return the permutation vector resulting from
                  sorting DX in increasing order and do not sort DX.
            = -1  means return the permutation vector resulting from
                  sorting DX in decreasing order and do not sort DX.
            = -2  means return the permutation vector resulting from
                  sorting DX in decreasing order and sort DX 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.
890619  Double precision version of SPSORT created by D. W. Lozier.
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
```

DPTSL

```
      SUBROUTINE DPTSL (N, D, E, B)
***BEGIN PROLOGUE  DPTSL
***PURPOSE  Solve a positive definite tridiagonal linear system.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D2B2A
***TYPE      DOUBLE PRECISION (SPTSL-S, DPTSL-D, CPTSL-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX, POSITIVE DEFINITE, SOLVE,
             TRIDIAGONAL
***AUTHOR  Dongarra, J., (ANL)
***DESCRIPTION

      DPTSL, given a positive definite symmetric tridiagonal matrix and
      a right hand side, will find the solution.

      On Entry

         N        INTEGER
                  is the order of the tridiagonal matrix.

         D        DOUBLE PRECISION(N)
                  is the diagonal of the tridiagonal matrix.
                  On output D is destroyed.

         E        DOUBLE PRECISION(N)
                  is the offdiagonal of the tridiagonal matrix.
                  E(1) through E(N-1) should contain the
                  offdiagonal.

         B        DOUBLE PRECISION(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
```

DQAG

```
      SUBROUTINE DQAG (F, A, B, EPSABS, EPSREL, KEY, RESULT, ABSERR,
     +   NEVAL, IER, LIMIT, LENW, LAST, IWORK, WORK)
***BEGIN PROLOGUE  DQAG
***PURPOSE  The routine calculates an approximation result to a given
            definite integral I = integral of F over (A,B),
            hopefully satisfying following claim for accuracy
            ABS(I-RESULT)LE.MAX(EPSABS,EPSREL*ABS(I)).
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A1A1
***TYPE      DOUBLE PRECISION (QAG-S, DQAG-D)
***KEYWORDS  AUTOMATIC INTEGRATOR, GAUSS-KRONROD RULES,
             GENERAL-PURPOSE, GLOBALLY ADAPTIVE, INTEGRAND EXAMINATOR,
             QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Computation of a definite integral
        Standard fortran subroutine
        Double precision version

            F      - Double precision
                     Function subprogram defining the integrand
                     Function F(X). The actual name for F needs to be
                     Declared E X T E R N A L in the driver program.

            A      - Double precision
                     Lower limit of integration

            B      - Double precision
                     Upper limit of integration

            EPSABS - Double precision
                     Absolute accuracy requested
            EPSREL - Double precision
                     Relative accuracy requested
                     If  EPSABS.LE.0
                     And EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
                     The routine will end with IER = 6.

            KEY    - Integer
                     Key for choice of local integration rule
                     A GAUSS-KRONROD PAIR is used with
                       7 - 15 POINTS If KEY.LT.2,
                      10 - 21 POINTS If KEY = 2,
                      15 - 31 POINTS If KEY = 3,
                      20 - 41 POINTS If KEY = 4,
                      25 - 51 POINTS If KEY = 5,
                      30 - 61 POINTS If KEY.GT.5.

          ON RETURN
            RESULT - Double precision
```

```
            Approximation to the integral

   ABSERR - Double precision
            Estimate of the modulus of the absolute error,
            Which should EQUAL or EXCEED ABS(I-RESULT)

   NEVAL  - Integer
            Number of integrand evaluations

   IER    - Integer
            IER = 0 Normal and reliable termination of the
                    routine. It is assumed that the requested
                    accuracy has been achieved.
            IER.GT.0 Abnormal termination of the routine
                    The estimates for RESULT and ERROR are
                    Less reliable. It is assumed that the
                    requested accuracy has not been achieved.
             ERROR MESSAGES
            IER = 1 Maximum number of subdivisions allowed
                    has been achieved. One can allow more
                    subdivisions by increasing the value of
                    LIMIT (and taking the according dimension
                    adjustments into account). HOWEVER, If
                    this yield no improvement it is advised
                    to analyze the integrand in order to
                    determine the integration difficulties.
                    If the position of a local difficulty can
                    be determined (I.E. SINGULARITY,
                    DISCONTINUITY WITHIN THE INTERVAL) One
                    will probably gain from splitting up the
                    interval at this point and calling the
                    INTEGRATOR on the SUBRANGES. If possible,
                    AN APPROPRIATE SPECIAL-PURPOSE INTEGRATOR
                    should be used which is designed for
                    handling the type of difficulty involved.
                = 2 The occurrence of roundoff error is
                    detected, which prevents the requested
                    tolerance from being achieved.
                = 3 Extremely bad integrand behaviour occurs
                    at some points of the integration
                    interval.
                = 6 The input is invalid, because
                    (EPSABS.LE.0 AND
                     EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28))
                    OR LIMIT.LT.1 OR LENW.LT.LIMIT*4.
                    RESULT, ABSERR, NEVAL, LAST are set
                    to zero.
                    EXCEPT when LENW is invalid, IWORK(1),
                    WORK(LIMIT*2+1) and WORK(LIMIT*3+1) are
                    set to zero, WORK(1) is set to A and
                    WORK(LIMIT+1) to B.

DIMENSIONING PARAMETERS
   LIMIT - Integer
           Dimensioning parameter for IWORK
           Limit determines the maximum number of subintervals
           in the partition of the given integration interval
           (A,B), LIMIT.GE.1.
           If LIMIT.LT.1, the routine will end with IER = 6.
```

```
            LENW  - Integer
                    Dimensioning parameter for work
                    LENW must be at least LIMIT*4.
                    IF LENW.LT.LIMIT*4, the routine will end with
                    IER = 6.

            LAST  - Integer
                    On return, LAST equals the number of subintervals
                    produced in the subdivision process, which
                    determines the number of significant elements
                    actually in the WORK ARRAYS.

         WORK ARRAYS
            IWORK - Integer
                    Vector of dimension at least limit, the first K
                    elements of which contain pointers to the error
                    estimates over the subintervals, such that
                    WORK(LIMIT*3+IWORK(1)),... , WORK(LIMIT*3+IWORK(K))
                    form a decreasing sequence with K = LAST If
                    LAST.LE.(LIMIT/2+2), and K = LIMIT+1-LAST otherwise

            WORK  - Double precision
                    Vector of dimension at least LENW
                    on return
                    WORK(1), ..., WORK(LAST) contain the left end
                    points of the subintervals in the partition of
                     (A,B),
                    WORK(LIMIT+1), ..., WORK(LIMIT+LAST) contain the
                     right end points,
                    WORK(LIMIT*2+1), ..., WORK(LIMIT*2+LAST) contain
                     the integral approximations over the subintervals,
                    WORK(LIMIT*3+1), ..., WORK(LIMIT*3+LAST) contain
                     the error estimates.

***REFERENCES  (NONE)
***ROUTINES CALLED  DQAGE, XERMSG
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQAGE

```
      SUBROUTINE DQAGE (F, A, B, EPSABS, EPSREL, KEY, LIMIT, RESULT,
     +   ABSERR, NEVAL, IER, ALIST, BLIST, RLIST, ELIST, IORD, LAST)
***BEGIN PROLOGUE  DQAGE
***PURPOSE  The routine calculates an approximation result to a given
            definite integral   I = Integral of F over (A,B),
            hopefully satisfying following claim for accuracy
            ABS(I-RESLT).LE.MAX(EPSABS,EPSREL*ABS(I)).
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A1A1
***TYPE      DOUBLE PRECISION (QAGE-S, DQAGE-D)
***KEYWORDS  AUTOMATIC INTEGRATOR, GAUSS-KRONROD RULES,
             GENERAL-PURPOSE, GLOBALLY ADAPTIVE, INTEGRAND EXAMINATOR,
             QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Computation of a definite integral
        Standard fortran subroutine
        Double precision version

        PARAMETERS
         ON ENTRY
            F      - Double precision
                     Function subprogram defining the integrand
                     function F(X). The actual name for F needs to be
                     declared E X T E R N A L in the driver program.

            A      - Double precision
                     Lower limit of integration

            B      - Double precision
                     Upper limit of integration

            EPSABS - Double precision
                     Absolute accuracy requested
            EPSREL - Double precision
                     Relative accuracy requested
                     If  EPSABS.LE.0
                     and EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
                     the routine will end with IER = 6.

            KEY    - Integer
                     Key for choice of local integration rule
                     A Gauss-Kronrod pair is used with
                          7 - 15 points if KEY.LT.2,
                         10 - 21 points if KEY = 2,
                         15 - 31 points if KEY = 3,
                         20 - 41 points if KEY = 4,
                         25 - 51 points if KEY = 5,
                         30 - 61 points if KEY.GT.5.
```

```
   LIMIT  - Integer
            Gives an upper bound on the number of subintervals
            in the partition of (A,B), LIMIT.GE.1.

ON RETURN
   RESULT - Double precision
            Approximation to the integral

   ABSERR - Double precision
            Estimate of the modulus of the absolute error,
            which should equal or exceed ABS(I-RESULT)

   NEVAL  - Integer
            Number of integrand evaluations

   IER    - Integer
            IER = 0 Normal and reliable termination of the
                    routine. It is assumed that the requested
                    accuracy has been achieved.
            IER.GT.0 Abnormal termination of the routine
                    The estimates for result and error are
                    less reliable. It is assumed that the
                    requested accuracy has not been achieved.
   ERROR MESSAGES
            IER = 1 Maximum number of subdivisions allowed
                    has been achieved. One can allow more
                    subdivisions by increasing the value
                    of LIMIT.
                    However, if this yields no improvement it
                    is rather advised to analyze the integrand
                    in order to determine the integration
                    difficulties. If the position of a local
                    difficulty can be determined(e.g.
                    SINGULARITY, DISCONTINUITY within the
                    interval) one will probably gain from
                    splitting up the interval at this point
                    and calling the integrator on the
                    subranges. If possible, an appropriate
                    special-purpose integrator should be used
                    which is designed for handling the type of
                    difficulty involved.
                = 2 The occurrence of roundoff error is
                    detected, which prevents the requested
                    tolerance from being achieved.
                = 3 Extremely bad integrand behaviour occurs
                    at some points of the integration
                    interval.
                = 6 The input is invalid, because
                    (EPSABS.LE.0 and
                     EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
                    RESULT, ABSERR, NEVAL, LAST, RLIST(1) ,
                    ELIST(1) and IORD(1) are set to zero.
                    ALIST(1) and BLIST(1) are set to A and B
                    respectively.

   ALIST   - Double precision
             Vector of dimension at least LIMIT, the first
              LAST  elements of which are the left
             end points of the subintervals in the partition
             of the given integration range (A,B)
```

```
            BLIST   - Double precision
                      Vector of dimension at least LIMIT, the first
                       LAST  elements of which are the right
                      end points of the subintervals in the partition
                      of the given integration range (A,B)

            RLIST   - Double precision
                      Vector of dimension at least LIMIT, the first
                       LAST  elements of which are the
                      integral approximations on the subintervals

            ELIST   - Double precision
                      Vector of dimension at least LIMIT, the first
                       LAST  elements of which are the moduli of the
                      absolute error estimates on the subintervals

            IORD    - Integer
                      Vector of dimension at least LIMIT, the first K
                      elements of which are pointers to the
                      error estimates over the subintervals,
                      such that ELIST(IORD(1)), ...,
                      ELIST(IORD(K)) form a decreasing sequence,
                      with K = LAST if LAST.LE.(LIMIT/2+2), and
                      K = LIMIT+1-LAST otherwise

            LAST    - Integer
                      Number of subintervals actually produced in the
                      subdivision process

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DQK15, DQK21, DQK31, DQK41, DQK51, DQK61,
                    DQPSRT
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQAGI

```
      SUBROUTINE DQAGI (F, BOUND, INF, EPSABS, EPSREL, RESULT, ABSERR,
     +   NEVAL, IER, LIMIT, LENW, LAST, IWORK, WORK)
***BEGIN PROLOGUE  DQAGI
***PURPOSE  The routine calculates an approximation result to a given
            INTEGRAL   I = Integral of F over (BOUND,+INFINITY)
            OR I = Integral of F over (-INFINITY,BOUND)
            OR I = Integral of F over (-INFINITY,+INFINITY)
            Hopefully satisfying following claim for accuracy
            ABS(I-RESULT).LE.MAX(EPSABS,EPSREL*ABS(I)).
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A3A1, H2A4A1
***TYPE      DOUBLE PRECISION (QAGI-S, DQAGI-D)
***KEYWORDS  AUTOMATIC INTEGRATOR, EXTRAPOLATION, GENERAL-PURPOSE,
             GLOBALLY ADAPTIVE, INFINITE INTERVALS, QUADPACK,
             QUADRATURE, TRANSFORMATION
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Integration over infinite intervals
        Standard fortran subroutine

        PARAMETERS
         ON ENTRY
            F      - Double precision
                     Function subprogram defining the integrand
                     function F(X). The actual name for F needs to be
                     declared E X T E R N A L in the driver program.

            BOUND  - Double precision
                     Finite bound of integration range
                     (has no meaning if interval is doubly-infinite)

            INF    - Integer
                     indicating the kind of integration range involved
                     INF = 1 corresponds to  (BOUND,+INFINITY),
                     INF = -1            to  (-INFINITY,BOUND),
                     INF = 2             to (-INFINITY,+INFINITY).

            EPSABS - Double precision
                     Absolute accuracy requested
            EPSREL - Double precision
                     Relative accuracy requested
                     If  EPSABS.LE.0
                     and EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
                     the routine will end with IER = 6.


         ON RETURN
            RESULT - Double precision
                     Approximation to the integral
```

```
ABSERR - Double precision
         Estimate of the modulus of the absolute error,
         which should equal or exceed ABS(I-RESULT)

NEVAL  - Integer
         Number of integrand evaluations

IER    - Integer
         IER = 0 normal and reliable termination of the
                 routine. It is assumed that the requested
                 accuracy has been achieved.
       - IER.GT.0 abnormal termination of the routine. The
                 estimates for result and error are less
                 reliable. It is assumed that the requested
                 accuracy has not been achieved.
ERROR MESSAGES
         IER = 1 Maximum number of subdivisions allowed
                 has been achieved. One can allow more
                 subdivisions by increasing the value of
                 LIMIT (and taking the according dimension
                 adjustments into account). However, if
                 this yields no improvement it is advised
                 to analyze the integrand in order to
                 determine the integration difficulties. If
                 the position of a local difficulty can be
                 determined (e.g. SINGULARITY,
                 DISCONTINUITY within the interval) one
                 will probably gain from splitting up the
                 interval at this point and calling the
                 integrator on the subranges. If possible,
                 an appropriate special-purpose integrator
                 should be used, which is designed for
                 handling the type of difficulty involved.
             = 2 The occurrence of roundoff error is
                 detected, which prevents the requested
                 tolerance from being achieved.
                 The error may be under-estimated.
             = 3 Extremely bad integrand behaviour occurs
                 at some points of the integration
                 interval.
             = 4 The algorithm does not converge.
                 Roundoff error is detected in the
                 extrapolation table.
                 It is assumed that the requested tolerance
                 cannot be achieved, and that the returned
                 RESULT is the best which can be obtained.
             = 5 The integral is probably divergent, or
                 slowly convergent. It must be noted that
                 divergence can occur with any other value
                 of IER.
             = 6 The input is invalid, because
                 (EPSABS.LE.0 and
                  EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28))
                  or LIMIT.LT.1 or LENIW.LT.LIMIT*4.
                 RESULT, ABSERR, NEVAL, LAST are set to
                 zero.  Except when LIMIT or LENIW is
                 invalid, IWORK(1), WORK(LIMIT*2+1) and
                 WORK(LIMIT*3+1) are set to ZERO, WORK(1)
                 is set to A and WORK(LIMIT+1) to B.
```

```
          DIMENSIONING PARAMETERS
             LIMIT - Integer
                     Dimensioning parameter for IWORK
                     LIMIT determines the maximum number of subintervals
                     in the partition of the given integration interval
                     (A,B), LIMIT.GE.1.
                     If LIMIT.LT.1, the routine will end with IER = 6.

             LENW  - Integer
                     Dimensioning parameter for WORK
                     LENW must be at least LIMIT*4.
                     If LENW.LT.LIMIT*4, the routine will end
                     with IER = 6.

             LAST  - Integer
                     On return, LAST equals the number of subintervals
                     produced in the subdivision process, which
                     determines the number of significant elements
                     actually in the WORK ARRAYS.

          WORK ARRAYS
             IWORK - Integer
                     Vector of dimension at least LIMIT, the first
                     K elements of which contain pointers
                     to the error estimates over the subintervals,
                     such that WORK(LIMIT*3+IWORK(1)),... ,
                     WORK(LIMIT*3+IWORK(K)) form a decreasing
                     sequence, with K = LAST if LAST.LE.(LIMIT/2+2), and
                     K = LIMIT+1-LAST otherwise

             WORK  - Double precision
                     Vector of dimension at least LENW
                     on return
                     WORK(1), ..., WORK(LAST) contain the left
                      end points of the subintervals in the
                      partition of (A,B),
                     WORK(LIMIT+1), ..., WORK(LIMIT+LAST) Contain
                      the right end points,
                     WORK(LIMIT*2+1), ...,WORK(LIMIT*2+LAST) contain the
                      integral approximations over the subintervals,
                     WORK(LIMIT*3+1), ..., WORK(LIMIT*3)
                      contain the error estimates.

***REFERENCES  (NONE)
***ROUTINES CALLED  DQAGIE, XERMSG
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQAGIE

```
      SUBROUTINE DQAGIE (F, BOUND, INF, EPSABS, EPSREL, LIMIT, RESULT,
     +   ABSERR, NEVAL, IER, ALIST, BLIST, RLIST, ELIST, IORD, LAST)
***BEGIN PROLOGUE  DQAGIE
***PURPOSE  The routine calculates an approximation result to a given
            integral   I = Integral of F over (BOUND,+INFINITY)
            or I = Integral of F over (-INFINITY,BOUND)
            or I = Integral of F over (-INFINITY,+INFINITY),
            hopefully satisfying following claim for accuracy
            ABS(I-RESULT).LE.MAX(EPSABS,EPSREL*ABS(I))
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A3A1, H2A4A1
***TYPE      DOUBLE PRECISION (QAGIE-S, DQAGIE-D)
***KEYWORDS  AUTOMATIC INTEGRATOR, EXTRAPOLATION, GENERAL-PURPOSE,
             GLOBALLY ADAPTIVE, INFINITE INTERVALS, QUADPACK,
             QUADRATURE, TRANSFORMATION
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

 Integration over infinite intervals
 Standard fortran subroutine

            F      - Double precision
                     Function subprogram defining the integrand
                     function F(X). The actual name for F needs to be
                     declared E X T E R N A L in the driver program.

            BOUND  - Double precision
                     Finite bound of integration range
                     (has no meaning if interval is doubly-infinite)

            INF    - Double precision
                     Indicating the kind of integration range involved
                     INF = 1 corresponds to  (BOUND,+INFINITY),
                     INF = -1            to  (-INFINITY,BOUND),
                     INF = 2             to (-INFINITY,+INFINITY).

            EPSABS - Double precision
                     Absolute accuracy requested
            EPSREL - Double precision
                     Relative accuracy requested
                     If  EPSABS.LE.0
                     and EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
                     the routine will end with IER = 6.

            LIMIT  - Integer
                     Gives an upper bound on the number of subintervals
                     in the partition of (A,B), LIMIT.GE.1

         ON RETURN
            RESULT - Double precision
                     Approximation to the integral
```

```
ABSERR - Double precision
         Estimate of the modulus of the absolute error,
         which should equal or exceed ABS(I-RESULT)

NEVAL  - Integer
         Number of integrand evaluations

IER    - Integer
         IER = 0 Normal and reliable termination of the
                 routine. It is assumed that the requested
                 accuracy has been achieved.
       - IER.GT.0 Abnormal termination of the routine. The
                 estimates for result and error are less
                 reliable. It is assumed that the requested
                 accuracy has not been achieved.
ERROR MESSAGES
         IER = 1 Maximum number of subdivisions allowed
                 has been achieved. One can allow more
                 subdivisions by increasing the value of
                 LIMIT (and taking the according dimension
                 adjustments into account).  However, if
                 this yields no improvement it is advised
                 to analyze the integrand in order to
                 determine the integration difficulties.
                 If the position of a local difficulty can
                 be determined (e.g. SINGULARITY,
                 DISCONTINUITY within the interval) one
                 will probably gain from splitting up the
                 interval at this point and calling the
                 integrator on the subranges. If possible,
                 an appropriate special-purpose integrator
                 should be used, which is designed for
                 handling the type of difficulty involved.
             = 2 The occurrence of roundoff error is
                 detected, which prevents the requested
                 tolerance from being achieved.
                 The error may be under-estimated.
             = 3 Extremely bad integrand behaviour occurs
                 at some points of the integration
                 interval.
             = 4 The algorithm does not converge.
                 Roundoff error is detected in the
                 extrapolation table.
                 It is assumed that the requested tolerance
                 cannot be achieved, and that the returned
                 result is the best which can be obtained.
             = 5 The integral is probably divergent, or
                 slowly convergent. It must be noted that
                 divergence can occur with any other value
                 of IER.
             = 6 The input is invalid, because
                 (EPSABS.LE.0 and
                  EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
                 RESULT, ABSERR, NEVAL, LAST, RLIST(1),
                 ELIST(1) and IORD(1) are set to zero.
                 ALIST(1) and BLIST(1) are set to 0
                 and 1 respectively.

ALIST  - Double precision
```

```
                 Vector of dimension at least LIMIT, the first
                  LAST  elements of which are the left
                 end points of the subintervals in the partition
                 of the transformed integration range (0,1).

        BLIST  - Double precision
                 Vector of dimension at least LIMIT, the first
                  LAST  elements of which are the right
                 end points of the subintervals in the partition
                 of the transformed integration range (0,1).

        RLIST  - Double precision
                 Vector of dimension at least LIMIT, the first
                  LAST  elements of which are the integral
                 approximations on the subintervals

        ELIST  - Double precision
                 Vector of dimension at least LIMIT,  the first
                 LAST elements of which are the moduli of the
                 absolute error estimates on the subintervals

        IORD   - Integer
                 Vector of dimension LIMIT, the first K
                 elements of which are pointers to the
                 error estimates over the subintervals,
                 such that ELIST(IORD(1)), ..., ELIST(IORD(K))
                 form a decreasing sequence, with K = LAST
                 If LAST.LE.(LIMIT/2+2), and K = LIMIT+1-LAST
                 otherwise

        LAST   - Integer
                 Number of subintervals actually produced
                 in the subdivision process

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DQELG, DQK15I, DQPSRT
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQAGP

```
      SUBROUTINE DQAGP (F, A, B, NPTS2, POINTS, EPSABS, EPSREL, RESULT,
     +   ABSERR, NEVAL, IER, LENIW, LENW, LAST, IWORK, WORK)
***BEGIN PROLOGUE  DQAGP
***PURPOSE  The routine calculates an approximation result to a given
            definite integral I = Integral of F over (A,B),
            hopefully satisfying following claim for accuracy
            break points of the integration interval, where local
            difficulties of the integrand may occur (e.g.
            SINGULARITIES, DISCONTINUITIES), are provided by the user.
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A2A1
***TYPE      DOUBLE PRECISION (QAGP-S, DQAGP-D)
***KEYWORDS  AUTOMATIC INTEGRATOR, EXTRAPOLATION, GENERAL-PURPOSE,
             GLOBALLY ADAPTIVE, QUADPACK, QUADRATURE,
             SINGULARITIES AT USER SPECIFIED POINTS
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Computation of a definite integral
        Standard fortran subroutine
        Double precision version

        PARAMETERS
         ON ENTRY
            F      - Double precision
                     Function subprogram defining the integrand
                     Function F(X). The actual name for F needs to be
                     declared E X T E R N A L in the driver program.

            A      - Double precision
                     Lower limit of integration

            B      - Double precision
                     Upper limit of integration

            NPTS2  - Integer
                     Number equal to two more than the number of
                     user-supplied break points within the integration
                     range, NPTS.GE.2.
                     If NPTS2.LT.2, The routine will end with IER = 6.

            POINTS - Double precision
                     Vector of dimension NPTS2, the first (NPTS2-2)
                     elements of which are the user provided break
                     points. If these points do not constitute an
                     ascending sequence there will be an automatic
                     sorting.

            EPSABS - Double precision
                     Absolute accuracy requested
            EPSREL - Double precision
```

```
            Relative accuracy requested
            If  EPSABS.LE.0
            And EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
            The routine will end with IER = 6.

ON RETURN
   RESULT - Double precision
            Approximation to the integral

   ABSERR - Double precision
            Estimate of the modulus of the absolute error,
            which should equal or exceed ABS(I-RESULT)

   NEVAL  - Integer
            Number of integrand evaluations

   IER    - Integer
            IER = 0 Normal and reliable termination of the
                    routine. It is assumed that the requested
                    accuracy has been achieved.
            IER.GT.0 Abnormal termination of the routine.
                    The estimates for integral and error are
                    less reliable. it is assumed that the
                    requested accuracy has not been achieved.
   ERROR MESSAGES
            IER = 1 Maximum number of subdivisions allowed
                    has been achieved. one can allow more
                    subdivisions by increasing the value of
                    LIMIT (and taking the according dimension
                    adjustments into account). However, if
                    this yields no improvement it is advised
                    to analyze the integrand in order to
                    determine the integration difficulties. If
                    the position of a local difficulty can be
                    determined (i.e. SINGULARITY,
                    DISCONTINUITY within the interval), it
                    should be supplied to the routine as an
                    element of the vector points. If necessary
                    an appropriate special-purpose integrator
                    must be used, which is designed for
                    handling the type of difficulty involved.
                = 2 The occurrence of roundoff error is
                    detected, which prevents the requested
                    tolerance from being achieved.
                    The error may be under-estimated.
                = 3 Extremely bad integrand behaviour occurs
                    at some points of the integration
                    interval.
                = 4 The algorithm does not converge.
                    roundoff error is detected in the
                    extrapolation table.
                    It is presumed that the requested
                    tolerance cannot be achieved, and that
                    the returned RESULT is the best which
                    can be obtained.
                = 5 The integral is probably divergent, or
                    slowly convergent. it must be noted that
                    divergence can occur with any other value
                    of IER.GT.0.
                = 6 The input is invalid because
```

```
                    NPTS2.LT.2 or
                    break points are specified outside
                    the integration range or
                    (EPSABS.LE.0 and
                     EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28))
                    RESULT, ABSERR, NEVAL, LAST are set to
                    zero.  Except when LENIW or LENW or NPTS2
                    is invalid, IWORK(1), IWORK(LIMIT+1),
                    WORK(LIMIT*2+1) and WORK(LIMIT*3+1)
                    are set to zero.
                    WORK(1) is set to A and WORK(LIMIT+1)
                    to B (where LIMIT = (LENIW-NPTS2)/2).

DIMENSIONING PARAMETERS
   LENIW - Integer
           Dimensioning parameter for IWORK
           LENIW determines LIMIT = (LENIW-NPTS2)/2,
           which is the maximum number of subintervals in the
           partition of the given integration interval (A,B),
           LENIW.GE.(3*NPTS2-2).
           If LENIW.LT.(3*NPTS2-2), the routine will end with
           IER = 6.

   LENW  - Integer
           Dimensioning parameter for WORK
           LENW must be at least LENIW*2-NPTS2.
           If LENW.LT.LENIW*2-NPTS2, the routine will end
           with IER = 6.

   LAST  - Integer
           On return, LAST equals the number of subintervals
           produced in the subdivision process, which
           determines the number of significant elements
           actually in the WORK ARRAYS.

WORK ARRAYS
   IWORK - Integer
           Vector of dimension at least LENIW. on return,
           the first K elements of which contain
           pointers to the error estimates over the
           subintervals, such that WORK(LIMIT*3+IWORK(1)),...,
           WORK(LIMIT*3+IWORK(K)) form a decreasing
           sequence, with K = LAST if LAST.LE.(LIMIT/2+2), and
           K = LIMIT+1-LAST otherwise
           IWORK(LIMIT+1), ...,IWORK(LIMIT+LAST) Contain the
            subdivision levels of the subintervals, i.e.
            if (AA,BB) is a subinterval of (P1,P2)
            where P1 as well as P2 is a user-provided
            break point or integration LIMIT, then (AA,BB) has
            level L if ABS(BB-AA) = ABS(P2-P1)*2**(-L),
           IWORK(LIMIT*2+1), ..., IWORK(LIMIT*2+NPTS2) have
            no significance for the user,
           note that LIMIT = (LENIW-NPTS2)/2.

   WORK  - Double precision
           Vector of dimension at least LENW
           on return
           WORK(1), ..., WORK(LAST) contain the left
            end points of the subintervals in the
            partition of (A,B),
```

```
                    WORK(LIMIT+1), ..., WORK(LIMIT+LAST) contain
                     the right end points,
                    WORK(LIMIT*2+1), ..., WORK(LIMIT*2+LAST) contain
                     the integral approximations over the subintervals,
                    WORK(LIMIT*3+1), ..., WORK(LIMIT*3+LAST)
                     contain the corresponding error estimates,
                    WORK(LIMIT*4+1), ..., WORK(LIMIT*4+NPTS2)
                     contain the integration limits and the
                     break points sorted in an ascending sequence.
                    note that LIMIT = (LENIW-NPTS2)/2.

***REFERENCES  (NONE)
***ROUTINES CALLED  DQAGPE, XERMSG
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQAGPE

```
      SUBROUTINE DQAGPE (F, A, B, NPTS2, POINTS, EPSABS, EPSREL, LIMIT,
         RESULT, ABSERR, NEVAL, IER, ALIST, BLIST, RLIST, ELIST, PTS,
     +   IORD, LEVEL, NDIN, LAST)
***BEGIN PROLOGUE  DQAGPE
***PURPOSE  Approximate a given definite integral I = Integral of F
            over (A,B), hopefully satisfying the accuracy claim:
                 ABS(I-RESULT).LE.MAX(EPSABS,EPSREL*ABS(I)).
            Break points of the integration interval, where local
            difficulties of the integrand may occur (e.g. singularities
            or discontinuities) are provided by the user.
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A2A1
***TYPE      DOUBLE PRECISION (QAGPE-S, DQAGPE-D)
***KEYWORDS  AUTOMATIC INTEGRATOR, EXTRAPOLATION, GENERAL-PURPOSE,
             GLOBALLY ADAPTIVE, QUADPACK, QUADRATURE,
             SINGULARITIES AT USER SPECIFIED POINTS
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Computation of a definite integral
        Standard fortran subroutine
        Double precision version

        PARAMETERS
         ON ENTRY
            F      - Double precision
                     Function subprogram defining the integrand
                     function F(X). The actual name for F needs to be
                     declared E X T E R N A L in the driver program.

            A      - Double precision
                     Lower limit of integration

            B      - Double precision
                     Upper limit of integration

            NPTS2  - Integer
                     Number equal to two more than the number of
                     user-supplied break points within the integration
                     range, NPTS2.GE.2.
                     If NPTS2.LT.2, the routine will end with IER = 6.

            POINTS - Double precision
                     Vector of dimension NPTS2, the first (NPTS2-2)
                     elements of which are the user provided break
                     POINTS. If these POINTS do not constitute an
                     ascending sequence there will be an automatic
                     sorting.

            EPSABS - Double precision
                     Absolute accuracy requested
```

```
   EPSREL - Double precision
            Relative accuracy requested
            If  EPSABS.LE.0
            and EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
            the routine will end with IER = 6.

   LIMIT  - Integer
            Gives an upper bound on the number of subintervals
            in the partition of (A,B), LIMIT.GE.NPTS2
            If LIMIT.LT.NPTS2, the routine will end with
            IER = 6.

ON RETURN
   RESULT - Double precision
            Approximation to the integral

   ABSERR - Double precision
            Estimate of the modulus of the absolute error,
            which should equal or exceed ABS(I-RESULT)

   NEVAL  - Integer
            Number of integrand evaluations

   IER    - Integer
            IER = 0 Normal and reliable termination of the
                    routine. It is assumed that the requested
                    accuracy has been achieved.
            IER.GT.0 Abnormal termination of the routine.
                    The estimates for integral and error are
                    less reliable. It is assumed that the
                    requested accuracy has not been achieved.
   ERROR MESSAGES
            IER = 1 Maximum number of subdivisions allowed
                    has been achieved. One can allow more
                    subdivisions by increasing the value of
                    LIMIT (and taking the according dimension
                    adjustments into account). However, if
                    this yields no improvement it is advised
                    to analyze the integrand in order to
                    determine the integration difficulties. If
                    the position of a local difficulty can be
                    determined (i.e. SINGULARITY,
                    DISCONTINUITY within the interval), it
                    should be supplied to the routine as an
                    element of the vector points. If necessary
                    an appropriate special-purpose integrator
                    must be used, which is designed for
                    handling the type of difficulty involved.
                = 2 The occurrence of roundoff error is
                    detected, which prevents the requested
                    tolerance from being achieved.
                    The error may be under-estimated.
                = 3 Extremely bad integrand behaviour occurs
                    At some points of the integration
                    interval.
                = 4 The algorithm does not converge.
                    Roundoff error is detected in the
                    extrapolation table. It is presumed that
                    the requested tolerance cannot be
                    achieved, and that the returned result is
```

```
                the best which can be obtained.
            = 5 The integral is probably divergent, or
                slowly convergent. It must be noted that
                divergence can occur with any other value
                of IER.GT.0.
            = 6 The input is invalid because
                NPTS2.LT.2 or
                Break points are specified outside
                the integration range or
                (EPSABS.LE.0 and
                 EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28))
                or LIMIT.LT.NPTS2.
                RESULT, ABSERR, NEVAL, LAST, RLIST(1),
                and ELIST(1) are set to zero. ALIST(1) and
                BLIST(1) are set to A and B respectively.

ALIST  - Double precision
         Vector of dimension at least LIMIT, the first
          LAST  elements of which are the left end points
         of the subintervals in the partition of the given
         integration range (A,B)

BLIST  - Double precision
         Vector of dimension at least LIMIT, the first
          LAST  elements of which are the right end points
         of the subintervals in the partition of the given
         integration range (A,B)

RLIST  - Double precision
         Vector of dimension at least LIMIT, the first
          LAST  elements of which are the integral
         approximations on the subintervals

ELIST  - Double precision
         Vector of dimension at least LIMIT, the first
          LAST  elements of which are the moduli of the
         absolute error estimates on the subintervals

PTS    - Double precision
         Vector of dimension at least NPTS2, containing the
         integration limits and the break points of the
         interval in ascending sequence.

LEVEL  - Integer
         Vector of dimension at least LIMIT, containing the
         subdivision levels of the subinterval, i.e. if
         (AA,BB) is a subinterval of (P1,P2) where P1 as
         well as P2 is a user-provided break point or
         integration limit, then (AA,BB) has level L if
         ABS(BB-AA) = ABS(P2-P1)*2**(-L).

NDIN   - Integer
         Vector of dimension at least NPTS2, after first
         integration over the intervals (PTS(I)),PTS(I+1),
         I = 0,1, ..., NPTS2-2, the error estimates over
         some of the intervals may have been increased
         artificially, in order to put their subdivision
         forward. If this happens for the subinterval
         numbered K, NDIN(K) is put to 1, otherwise
         NDIN(K) = 0.
```

```
            IORD   - Integer
                     Vector of dimension at least LIMIT, the first K
                     elements of which are pointers to the
                     error estimates over the subintervals,
                     such that ELIST(IORD(1)), ..., ELIST(IORD(K))
                     form a decreasing sequence, with K = LAST
                     If LAST.LE.(LIMIT/2+2), and K = LIMIT+1-LAST
                     otherwise

            LAST   - Integer
                     Number of subintervals actually produced in the
                     subdivisions process

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DQELG, DQK21, DQPSRT
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQAGS

```
      SUBROUTINE DQAGS (F, A, B, EPSABS, EPSREL, RESULT, ABSERR, NEVAL,
     +   IER, LIMIT, LENW, LAST, IWORK, WORK)
***BEGIN PROLOGUE  DQAGS
***PURPOSE  The routine calculates an approximation result to a given
            Definite integral  I = Integral of F over (A,B),
            Hopefully satisfying following claim for accuracy
            ABS(I-RESULT).LE.MAX(EPSABS,EPSREL*ABS(I)).
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A1A1
***TYPE      DOUBLE PRECISION (QAGS-S, DQAGS-D)
***KEYWORDS  AUTOMATIC INTEGRATOR, END POINT SINGULARITIES,
             EXTRAPOLATION, GENERAL-PURPOSE, GLOBALLY ADAPTIVE,
             QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Computation of a definite integral
        Standard fortran subroutine
        Double precision version


        PARAMETERS
         ON ENTRY
            F      - Double precision
                     Function subprogram defining the integrand
                     Function F(X). The actual name for F needs to be
                     Declared E X T E R N A L in the driver program.

            A      - Double precision
                     Lower limit of integration

            B      - Double precision
                     Upper limit of integration

            EPSABS - Double precision
                     Absolute accuracy requested
            EPSREL - Double precision
                     Relative accuracy requested
                     If  EPSABS.LE.0
                     And EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
                     The routine will end with IER = 6.

         ON RETURN
            RESULT - Double precision
                     Approximation to the integral

            ABSERR - Double precision
                     Estimate of the modulus of the absolute error,
                     which should equal or exceed ABS(I-RESULT)

            NEVAL  - Integer
```

```
            Number of integrand evaluations

   IER    - Integer
            IER = 0 Normal and reliable termination of the
                    routine. It is assumed that the requested
                    accuracy has been achieved.
            IER.GT.0 Abnormal termination of the routine
                    The estimates for integral and error are
                    less reliable. It is assumed that the
                    requested accuracy has not been achieved.
   ERROR MESSAGES
            IER = 1 Maximum number of subdivisions allowed
                    has been achieved. One can allow more sub-
                    divisions by increasing the value of LIMIT
                    (and taking the according dimension
                    adjustments into account. However, if
                    this yields no improvement it is advised
                    to analyze the integrand in order to
                    determine the integration difficulties. If
                    the position of a local difficulty can be
                    determined (E.G. SINGULARITY,
                    DISCONTINUITY WITHIN THE INTERVAL) one
                    will probably gain from splitting up the
                    interval at this point and calling the
                    integrator on the subranges. If possible,
                    an appropriate special-purpose integrator
                    should be used, which is designed for
                    handling the type of difficulty involved.
                = 2 The occurrence of roundoff error is detec-
                    ted, which prevents the requested
                    tolerance from being achieved.
                    The error may be under-estimated.
                = 3 Extremely bad integrand behaviour
                    occurs at some points of the integration
                    interval.
                = 4 The algorithm does not converge.
                    Roundoff error is detected in the
                    Extrapolation table. It is presumed that
                    the requested tolerance cannot be
                    achieved, and that the returned result is
                    the best which can be obtained.
                = 5 The integral is probably divergent, or
                    slowly convergent. It must be noted that
                    divergence can occur with any other value
                    of IER.
                = 6 The input is invalid, because
                    (EPSABS.LE.0 AND
                     EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28)
                    OR LIMIT.LT.1 OR LENW.LT.LIMIT*4.
                    RESULT, ABSERR, NEVAL, LAST are set to
                    zero.  Except when LIMIT or LENW is
                    invalid, IWORK(1), WORK(LIMIT*2+1) and
                    WORK(LIMIT*3+1) are set to zero, WORK(1)
                    is set to A and WORK(LIMIT+1) TO B.

DIMENSIONING PARAMETERS
   LIMIT - Integer
           DIMENSIONING PARAMETER FOR IWORK
           LIMIT determines the maximum number of subintervals
           in the partition of the given integration interval
```

```
                (A,B), LIMIT.GE.1.
                IF LIMIT.LT.1, the routine will end with IER = 6.

         LENW  - Integer
                DIMENSIONING PARAMETER FOR WORK
                LENW must be at least LIMIT*4.
                If LENW.LT.LIMIT*4, the routine will end
                with IER = 6.

         LAST  - Integer
                On return, LAST equals the number of subintervals
                produced in the subdivision process, determines the
                number of significant elements actually in the WORK
                Arrays.

      WORK ARRAYS
         IWORK - Integer
                Vector of dimension at least LIMIT, the first K
                elements of which contain pointers
                to the error estimates over the subintervals
                such that WORK(LIMIT*3+IWORK(1)),... ,
                WORK(LIMIT*3+IWORK(K)) form a decreasing
                sequence, with K = LAST IF LAST.LE.(LIMIT/2+2),
                and K = LIMIT+1-LAST otherwise

         WORK  - Double precision
                Vector of dimension at least LENW
                on return
                WORK(1), ..., WORK(LAST) contain the left
                 end-points of the subintervals in the
                 partition of (A,B),
                WORK(LIMIT+1), ..., WORK(LIMIT+LAST) contain
                 the right end-points,
                WORK(LIMIT*2+1), ..., WORK(LIMIT*2+LAST) contain
                 the integral approximations over the subintervals,
                WORK(LIMIT*3+1), ..., WORK(LIMIT*3+LAST)
                 contain the error estimates.

***REFERENCES  (NONE)
***ROUTINES CALLED  DQAGSE, XERMSG
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQAGSE

```
      SUBROUTINE DQAGSE (F, A, B, EPSABS, EPSREL, LIMIT, RESULT, ABSERR,
     +   NEVAL, IER, ALIST, BLIST, RLIST, ELIST, IORD, LAST)
***BEGIN PROLOGUE  DQAGSE
***PURPOSE  The routine calculates an approximation result to a given
            definite integral I = Integral of F over (A,B),
            hopefully satisfying following claim for accuracy
            ABS(I-RESULT).LE.MAX(EPSABS,EPSREL*ABS(I)).
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A1A1
***TYPE      DOUBLE PRECISION (QAGSE-S, DQAGSE-D)
***KEYWORDS  AUTOMATIC INTEGRATOR, END POINT SINGULARITIES,
             EXTRAPOLATION, GENERAL-PURPOSE, GLOBALLY ADAPTIVE,
             QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Computation of a definite integral
        Standard fortran subroutine
        Double precision version

        PARAMETERS
         ON ENTRY
            F      - Double precision
                     Function subprogram defining the integrand
                     function F(X). The actual name for F needs to be
                     declared E X T E R N A L in the driver program.

            A      - Double precision
                     Lower limit of integration

            B      - Double precision
                     Upper limit of integration

            EPSABS - Double precision
                     Absolute accuracy requested
            EPSREL - Double precision
                     Relative accuracy requested
                     If  EPSABS.LE.0
                     and EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
                     the routine will end with IER = 6.

            LIMIT  - Integer
                     Gives an upper bound on the number of subintervals
                     in the partition of (A,B)

         ON RETURN
            RESULT - Double precision
                     Approximation to the integral

            ABSERR - Double precision
                     Estimate of the modulus of the absolute error,
```

```
         which should equal or exceed ABS(I-RESULT)

NEVAL  - Integer
         Number of integrand evaluations

IER    - Integer
         IER = 0 Normal and reliable termination of the
                 routine. It is assumed that the requested
                 accuracy has been achieved.
         IER.GT.0 Abnormal termination of the routine
                 the estimates for integral and error are
                 less reliable. It is assumed that the
                 requested accuracy has not been achieved.
ERROR MESSAGES
             = 1 Maximum number of subdivisions allowed
                 has been achieved. One can allow more sub-
                 divisions by increasing the value of LIMIT
                 (and taking the according dimension
                 adjustments into account). However, if
                 this yields no improvement it is advised
                 to analyze the integrand in order to
                 determine the integration difficulties. If
                 the position of a local difficulty can be
                 determined (e.g. singularity,
                 discontinuity within the interval) one
                 will probably gain from splitting up the
                 interval at this point and calling the
                 integrator on the subranges. If possible,
                 an appropriate special-purpose integrator
                 should be used, which is designed for
                 handling the type of difficulty involved.
             = 2 The occurrence of roundoff error is detec-
                 ted, which prevents the requested
                 tolerance from being achieved.
                 The error may be under-estimated.
             = 3 Extremely bad integrand behaviour
                 occurs at some points of the integration
                 interval.
             = 4 The algorithm does not converge.
                 Roundoff error is detected in the
                 extrapolation table.
                 It is presumed that the requested
                 tolerance cannot be achieved, and that the
                 returned result is the best which can be
                 obtained.
             = 5 The integral is probably divergent, or
                 slowly convergent. It must be noted that
                 divergence can occur with any other value
                 of IER.
             = 6 The input is invalid, because
                 EPSABS.LE.0 and
                 EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28).
                 RESULT, ABSERR, NEVAL, LAST, RLIST(1),
                 IORD(1) and ELIST(1) are set to zero.
                 ALIST(1) and BLIST(1) are set to A and B
                 respectively.

ALIST  - Double precision
         Vector of dimension at least LIMIT, the first
          LAST  elements of which are the left end points
```

```
                 of the subintervals in the partition of the
                 given integration range (A,B)

        BLIST  - Double precision
                 Vector of dimension at least LIMIT, the first
                  LAST  elements of which are the right end points
                 of the subintervals in the partition of the given
                 integration range (A,B)

        RLIST  - Double precision
                 Vector of dimension at least LIMIT, the first
                  LAST  elements of which are the integral
                 approximations on the subintervals

        ELIST  - Double precision
                 Vector of dimension at least LIMIT, the first
                  LAST  elements of which are the moduli of the
                 absolute error estimates on the subintervals

        IORD   - Integer
                 Vector of dimension at least LIMIT, the first K
                 elements of which are pointers to the
                 error estimates over the subintervals,
                 such that ELIST(IORD(1)), ..., ELIST(IORD(K))
                 form a decreasing sequence, with K = LAST
                 If LAST.LE.(LIMIT/2+2), and K = LIMIT+1-LAST
                 otherwise

        LAST   - Integer
                 Number of subintervals actually produced in the
                 subdivision process

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DQELG, DQK21, DQPSRT
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQAWC

```
      SUBROUTINE DQAWC (F, A, B, C, EPSABS, EPSREL, RESULT, ABSERR,
     +   NEVAL, IER, LIMIT, LENW, LAST, IWORK, WORK)
***BEGIN PROLOGUE  DQAWC
***PURPOSE  The routine calculates an approximation result to a
            Cauchy principal value I = INTEGRAL of F*W over (A,B)
            (W(X) = 1/((X-C), C.NE.A, C.NE.B), hopefully satisfying
            following claim for accuracy
            ABS(I-RESULT).LE.MAX(EPSABE,EPSREL*ABS(I)).
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A2A1, J4
***TYPE      DOUBLE PRECISION (QAWC-S, DQAWC-D)
***KEYWORDS  AUTOMATIC INTEGRATOR, CAUCHY PRINCIPAL VALUE,
             CLENSHAW-CURTIS METHOD, GLOBALLY ADAPTIVE, QUADPACK,
             QUADRATURE, SPECIAL-PURPOSE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Computation of a Cauchy principal value
        Standard fortran subroutine
        Double precision version


        PARAMETERS
         ON ENTRY
            F      - Double precision
                     Function subprogram defining the integrand
                     Function F(X). The actual name for F needs to be
                     declared E X T E R N A L in the driver program.

            A      - Double precision
                     Under limit of integration

            B      - Double precision
                     Upper limit of integration

            C      - Parameter in the weight function, C.NE.A, C.NE.B.
                     If C = A or C = B, the routine will end with
                     IER = 6 .

            EPSABS - Double precision
                     Absolute accuracy requested
            EPSREL - Double precision
                     Relative accuracy requested
                     If  EPSABS.LE.0
                     and EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
                     the routine will end with IER = 6.

         ON RETURN
            RESULT - Double precision
                     Approximation to the integral
```

```
   ABSERR - Double precision
            Estimate or the modulus of the absolute error,
            Which should equal or exceed ABS(I-RESULT)

   NEVAL  - Integer
            Number of integrand evaluations

   IER    - Integer
            IER = 0 Normal and reliable termination of the
                    routine. It is assumed that the requested
                    accuracy has been achieved.
            IER.GT.0 Abnormal termination of the routine
                    the estimates for integral and error are
                    less reliable. It is assumed that the
                    requested accuracy has not been achieved.
   ERROR MESSAGES
            IER = 1 Maximum number of subdivisions allowed
                    has been achieved. One can allow more sub-
                    divisions by increasing the value of LIMIT
                    (and taking the according dimension
                    adjustments into account). However, if
                    this yields no improvement it is advised
                    to analyze the integrand in order to
                    determine the integration difficulties.
                    If the position of a local difficulty
                    can be determined (e.g. SINGULARITY,
                    DISCONTINUITY within the interval) one
                    will probably gain from splitting up the
                    interval at this point and calling
                    appropriate integrators on the subranges.
                = 2 The occurrence of roundoff error is detec-
                    ted, which prevents the requested
                    tolerance from being achieved.
                = 3 Extremely bad integrand behaviour occurs
                    at some points of the integration
                    interval.
                = 6 The input is invalid, because
                    C = A or C = B or
                    (EPSABS.LE.0 and
                     EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28))
                    or LIMIT.LT.1 or LENW.LT.LIMIT*4.
                    RESULT, ABSERR, NEVAL, LAST are set to
                    zero.  Except when LENW or LIMIT is
                    invalid, IWORK(1), WORK(LIMIT*2+1) and
                    WORK(LIMIT*3+1) are set to zero, WORK(1)
                    is set to A and WORK(LIMIT+1) to B.

DIMENSIONING PARAMETERS
   LIMIT - Integer
           Dimensioning parameter for IWORK
           LIMIT determines the maximum number of subintervals
           in the partition of the given integration interval
           (A,B), LIMIT.GE.1.
           If LIMIT.LT.1, the routine will end with IER = 6.

  LENW   - Integer
           Dimensioning parameter for WORK
           LENW must be at least LIMIT*4.
           If LENW.LT.LIMIT*4, the routine will end with
           IER = 6.
```

```
            LAST  - Integer
                    On return, LAST equals the number of subintervals
                    produced in the subdivision process, which
                    determines the number of significant elements
                    actually in the WORK ARRAYS.

         WORK ARRAYS
            IWORK - Integer
                    Vector of dimension at least LIMIT, the first K
                    elements of which contain pointers
                    to the error estimates over the subintervals,
                    such that WORK(LIMIT*3+IWORK(1)), ... ,
                    WORK(LIMIT*3+IWORK(K)) form a decreasing
                    sequence, with K = LAST if LAST.LE.(LIMIT/2+2),
                    and K = LIMIT+1-LAST otherwise

            WORK  - Double precision
                    Vector of dimension at least LENW
                    On return
                    WORK(1), ..., WORK(LAST) contain the left
                     end points of the subintervals in the
                     partition of (A,B),
                    WORK(LIMIT+1), ..., WORK(LIMIT+LAST) contain
                     the right end points,
                    WORK(LIMIT*2+1), ..., WORK(LIMIT*2+LAST) contain
                     the integral approximations over the subintervals,
                    WORK(LIMIT*3+1), ..., WORK(LIMIT*3+LAST)
                     contain the error estimates.

***REFERENCES  (NONE)
***ROUTINES CALLED  DQAWCE, XERMSG
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQAWCE

```
      SUBROUTINE DQAWCE (F, A, B, C, EPSABS, EPSREL, LIMIT, RESULT,
     +   ABSERR, NEVAL, IER, ALIST, BLIST, RLIST, ELIST, IORD, LAST)
***BEGIN PROLOGUE  DQAWCE
***PURPOSE  The routine calculates an approximation result to a
            CAUCHY PRINCIPAL VALUE I = Integral of F*W over (A,B)
            (W(X) = 1/(X-C), (C.NE.A, C.NE.B), hopefully satisfying
            following claim for accuracy
            ABS(I-RESULT).LE.MAX(EPSABS,EPSREL*ABS(I))
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A2A1, J4
***TYPE      DOUBLE PRECISION (QAWCE-S, DQAWCE-D)
***KEYWORDS  AUTOMATIC INTEGRATOR, CAUCHY PRINCIPAL VALUE,
             CLENSHAW-CURTIS METHOD, QUADPACK, QUADRATURE,
             SPECIAL-PURPOSE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Computation of a CAUCHY PRINCIPAL VALUE
        Standard fortran subroutine
        Double precision version

        PARAMETERS
         ON ENTRY
            F      - Double precision
                     Function subprogram defining the integrand
                     function F(X). The actual name for F needs to be
                     declared E X T E R N A L in the driver program.

            A      - Double precision
                     Lower limit of integration

            B      - Double precision
                     Upper limit of integration

            C      - Double precision
                     Parameter in the WEIGHT function, C.NE.A, C.NE.B
                     If C = A OR C = B, the routine will end with
                     IER = 6.

            EPSABS - Double precision
                     Absolute accuracy requested
            EPSREL - Double precision
                     Relative accuracy requested
                     If  EPSABS.LE.0
                     and EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
                     the routine will end with IER = 6.

            LIMIT  - Integer
                     Gives an upper bound on the number of subintervals
                     in the partition of (A,B), LIMIT.GE.1
```

```
ON RETURN
   RESULT - Double precision
            Approximation to the integral

   ABSERR - Double precision
            Estimate of the modulus of the absolute error,
            which should equal or exceed ABS(I-RESULT)

   NEVAL  - Integer
            Number of integrand evaluations

   IER    - Integer
            IER = 0 Normal and reliable termination of the
                    routine. It is assumed that the requested
                    accuracy has been achieved.
            IER.GT.0 Abnormal termination of the routine
                    the estimates for integral and error are
                    less reliable. It is assumed that the
                    requested accuracy has not been achieved.
   ERROR MESSAGES
            IER = 1 Maximum number of subdivisions allowed
                    has been achieved. One can allow more sub-
                    divisions by increasing the value of
                    LIMIT. However, if this yields no
                    improvement it is advised to analyze the
                    the integrand, in order to determine the
                    the integration difficulties. If the
                    position of a local difficulty can be
                    determined (e.g. SINGULARITY,
                    DISCONTINUITY within the interval) one
                    will probably gain from splitting up the
                    interval at this point and calling
                    appropriate integrators on the subranges.
                = 2 The occurrence of roundoff error is detec-
                    ted, which prevents the requested
                    tolerance from being achieved.
                = 3 Extremely bad integrand behaviour
                    occurs at some interior points of
                    the integration interval.
                = 6 The input is invalid, because
                    C = A or C = B or
                    (EPSABS.LE.0 and
                     EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28))
                    or LIMIT.LT.1.
                    RESULT, ABSERR, NEVAL, RLIST(1), ELIST(1),
                    IORD(1) and LAST are set to zero. ALIST(1)
                    and BLIST(1) are set to A and B
                    respectively.

   ALIST   - Double precision
             Vector of dimension at least LIMIT, the first
              LAST  elements of which are the left
             end points of the subintervals in the partition
             of the given integration range (A,B)

   BLIST   - Double precision
             Vector of dimension at least LIMIT, the first
              LAST  elements of which are the right
             end points of the subintervals in the partition
             of the given integration range (A,B)
```

```
            RLIST   - Double precision
                      Vector of dimension at least LIMIT, the first
                       LAST  elements of which are the integral
                      approximations on the subintervals

            ELIST   - Double precision
                      Vector of dimension LIMIT, the first  LAST
                      elements of which are the moduli of the absolute
                      error estimates on the subintervals

            IORD    - Integer
                      Vector of dimension at least LIMIT, the first K
                      elements of which are pointers to the error
                      estimates over the subintervals, so that
                      ELIST(IORD(1)), ..., ELIST(IORD(K)) with K = LAST
                      If LAST.LE.(LIMIT/2+2), and K = LIMIT+1-LAST
                      otherwise, form a decreasing sequence

            LAST    - Integer
                      Number of subintervals actually produced in
                      the subdivision process

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DQC25C, DQPSRT
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQAWF

```
      SUBROUTINE DQAWF (F, A, OMEGA, INTEGR, EPSABS, RESULT, ABSERR,
     +   NEVAL, IER, LIMLST, LST, LENIW, MAXP1, LENW, IWORK, WORK)
***BEGIN PROLOGUE  DQAWF
***PURPOSE  The routine calculates an approximation result to a given
            Fourier integral I=Integral of F(X)*W(X) over (A,INFINITY)
            where W(X) = COS(OMEGA*X) or W(X) = SIN(OMEGA*X).
            Hopefully satisfying following claim for accuracy
            ABS(I-RESULT).LE.EPSABS.
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A3A1
***TYPE      DOUBLE PRECISION (QAWF-S, DQAWF-D)
***KEYWORDS  AUTOMATIC INTEGRATOR, CONVERGENCE ACCELERATION,
             FOURIER INTEGRALS, INTEGRATION BETWEEN ZEROS, QUADPACK,
             QUADRATURE, SPECIAL-PURPOSE INTEGRAL
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Computation of Fourier integrals
        Standard fortran subroutine
        Double precision version


        PARAMETERS
         ON ENTRY
            F      - Double precision
                     Function subprogram defining the integrand
                     function F(X). The actual name for F needs to be
                     declared E X T E R N A L in the driver program.

            A      - Double precision
                     Lower limit of integration

            OMEGA  - Double precision
                     Parameter in the integrand WEIGHT function

            INTEGR - Integer
                     Indicates which of the WEIGHT functions is used
                     INTEGR = 1      W(X) = COS(OMEGA*X)
                     INTEGR = 2      W(X) = SIN(OMEGA*X)
                     IF INTEGR.NE.1.AND.INTEGR.NE.2, the routine
                     will end with IER = 6.

            EPSABS - Double precision
                     Absolute accuracy requested, EPSABS.GT.0.
                     If EPSABS.LE.0, the routine will end with IER = 6.

         ON RETURN
            RESULT - Double precision
                     Approximation to the integral

            ABSERR - Double precision
```

```
          Estimate of the modulus of the absolute error,
          Which should equal or exceed ABS(I-RESULT)

NEVAL  - Integer
          Number of integrand evaluations

IER    - Integer
          IER = 0 Normal and reliable termination of the
                  routine. It is assumed that the requested
                  accuracy has been achieved.
          IER.GT.0 Abnormal termination of the routine.
                  The estimates for integral and error are
                  less reliable. It is assumed that the
                  requested accuracy has not been achieved.
ERROR MESSAGES
         If OMEGA.NE.0
          IER = 1 Maximum number of cycles allowed
                  has been achieved, i.e. of subintervals
                  (A+(K-1)C,A+KC) where
                  C = (2*INT(ABS(OMEGA))+1)*PI/ABS(OMEGA),
                  FOR K = 1, 2, ..., LST.
                  One can allow more cycles by increasing
                  the value of LIMLST (and taking the
                  according dimension adjustments into
                  account). Examine the array IWORK which
                  contains the error flags on the cycles, in
                  order to look for eventual local
                  integration difficulties.
                  If the position of a local difficulty
                  can be determined (e.g. singularity,
                  discontinuity within the interval) one
                  will probably gain from splitting up the
                  interval at this point and calling
                  appropriate integrators on the subranges.
              = 4 The extrapolation table constructed for
                  convergence acceleration of the series
                  formed by the integral contributions over
                  the cycles, does not converge to within
                  the requested accuracy.
                  As in the case of IER = 1, it is advised
                  to examine the array IWORK which contains
                  the error flags on the cycles.
              = 6 The input is invalid because
                  (INTEGR.NE.1 AND INTEGR.NE.2) or
                   EPSABS.LE.0 or LIMLST.LT.1 or
                   LENIW.LT.(LIMLST+2) or MAXP1.LT.1 or
                   LENW.LT.(LENIW*2+MAXP1*25).
                   RESULT, ABSERR, NEVAL, LST are set to
                   zero.
              = 7 Bad integrand behaviour occurs within
                  one or more of the cycles. Location and
                  type of the difficulty involved can be
                  determined from the first LST elements of
                  vector IWORK.  Here LST is the number of
                  cycles actually needed (see below).
                  IWORK(K) = 1 The maximum number of
                               subdivisions (=(LENIW-LIMLST)
                               /2) has been achieved on the
                               K th cycle.
                           = 2 Occurrence of roundoff error
```

```
                                        is detected and prevents the
                                        tolerance imposed on the K th
                                        cycle, from being achieved
                                        on this cycle.
                                    = 3 Extremely bad integrand
                                        behaviour occurs at some
                                        points of the K th cycle.
                                    = 4 The integration procedure
                                        over the K th cycle does
                                        not converge (to within the
                                        required accuracy) due to
                                        roundoff in the extrapolation
                                        procedure invoked on this
                                        cycle. It is assumed that the
                                        result on this interval is
                                        the best which can be
                                        obtained.
                                    = 5 The integral over the K th
                                        cycle is probably divergent
                                        or slowly convergent. It must
                                        be noted that divergence can
                                        occur with any other value of
                                        IWORK(K).
            If OMEGA = 0 and INTEGR = 1,
            The integral is calculated by means of DQAGIE,
            and IER = IWORK(1) (with meaning as described
            for IWORK(K),K = 1).

DIMENSIONING PARAMETERS
   LIMLST - Integer
            LIMLST gives an upper bound on the number of
            cycles, LIMLST.GE.3.
            If LIMLST.LT.3, the routine will end with IER = 6.

   LST    - Integer
            On return, LST indicates the number of cycles
            actually needed for the integration.
            If OMEGA = 0, then LST is set to 1.

   LENIW  - Integer
            Dimensioning parameter for IWORK. On entry,
            (LENIW-LIMLST)/2 equals the maximum number of
            subintervals allowed in the partition of each
            cycle, LENIW.GE.(LIMLST+2).
            If LENIW.LT.(LIMLST+2), the routine will end with
            IER = 6.

   MAXP1  - Integer
            MAXP1 gives an upper bound on the number of
            Chebyshev moments which can be stored, i.e. for
            the intervals of lengths ABS(B-A)*2**(-L),
            L = 0,1, ..., MAXP1-2, MAXP1.GE.1.
            If MAXP1.LT.1, the routine will end with IER = 6.
   LENW   - Integer
            Dimensioning parameter for WORK
            LENW must be at least LENIW*2+MAXP1*25.
            If LENW.LT.(LENIW*2+MAXP1*25), the routine will
            end with IER = 6.

WORK ARRAYS
```

```
            IWORK  - Integer
                     Vector of dimension at least LENIW
                     On return, IWORK(K) FOR K = 1, 2, ..., LST
                     contain the error flags on the cycles.

            WORK   - Double precision
                     Vector of dimension at least
                     On return,
                     WORK(1), ..., WORK(LST) contain the integral
                      approximations over the cycles,
                     WORK(LIMLST+1), ..., WORK(LIMLST+LST) contain
                      the error estimates over the cycles.
                     further elements of WORK have no specific
                     meaning for the user.

***REFERENCES  (NONE)
***ROUTINES CALLED  DQAWFE, XERMSG
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQAWFE

```
      SUBROUTINE DQAWFE (F, A, OMEGA, INTEGR, EPSABS, LIMLST, LIMIT,
     +   MAXP1, RESULT, ABSERR, NEVAL, IER, RSLST, ERLST, IERLST, LST,
     +   ALIST, BLIST, RLIST, ELIST, IORD, NNLOG, CHEBMO)
***BEGIN PROLOGUE  DQAWFE
***PURPOSE  The routine calculates an approximation result to a
            given Fourier integral
            I = Integral of F(X)*W(X) over (A,INFINITY)
            where W(X)=COS(OMEGA*X) or W(X)=SIN(OMEGA*X),
            hopefully satisfying following claim for accuracy
            ABS(I-RESULT).LE.EPSABS.
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A3A1
***TYPE      DOUBLE PRECISION (QAWFE-S, DQAWFE-D)
***KEYWORDS  AUTOMATIC INTEGRATOR, CONVERGENCE ACCELERATION,
             FOURIER INTEGRALS, INTEGRATION BETWEEN ZEROS, QUADPACK,
             QUADRATURE, SPECIAL-PURPOSE INTEGRAL
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Computation of Fourier integrals
        Standard fortran subroutine
        Double precision version

        PARAMETERS
         ON ENTRY
            F      - Double precision
                     Function subprogram defining the integrand
                     Function F(X). The actual name for F needs to
                     be declared E X T E R N A L in the driver program.

            A      - Double precision
                     Lower limit of integration

            OMEGA  - Double precision
                     Parameter in the WEIGHT function

            INTEGR - Integer
                     Indicates which WEIGHT function is used
                     INTEGR = 1      W(X) = COS(OMEGA*X)
                     INTEGR = 2      W(X) = SIN(OMEGA*X)
                     If INTEGR.NE.1.AND.INTEGR.NE.2, the routine will
                     end with IER = 6.

            EPSABS - Double precision
                     absolute accuracy requested, EPSABS.GT.0
                     If EPSABS.LE.0, the routine will end with IER = 6.

            LIMLST - Integer
                     LIMLST gives an upper bound on the number of
                     cycles, LIMLST.GE.1.
                     If LIMLST.LT.3, the routine will end with IER = 6.
```

```
   LIMIT  - Integer
            Gives an upper bound on the number of subintervals
            allowed in the partition of each cycle, LIMIT.GE.1
            each cycle, LIMIT.GE.1.

   MAXP1  - Integer
            Gives an upper bound on the number of
            Chebyshev moments which can be stored, I.E.
            for the intervals of lengths ABS(B-A)*2**(-L),
            L=0,1, ..., MAXP1-2, MAXP1.GE.1

ON RETURN
   RESULT - Double precision
            Approximation to the integral X

   ABSERR - Double precision
            Estimate of the modulus of the absolute error,
            which should equal or exceed ABS(I-RESULT)

   NEVAL  - Integer
            Number of integrand evaluations

   IER    - IER = 0 Normal and reliable termination of
                    the routine. It is assumed that the
                    requested accuracy has been achieved.
            IER.GT.0 Abnormal termination of the routine. The
                    estimates for integral and error are less
                    reliable. It is assumed that the requested
                    accuracy has not been achieved.
   ERROR MESSAGES
           If OMEGA.NE.0
            IER = 1 Maximum number of  cycles  allowed
                    Has been achieved., i.e. of subintervals
                    (A+(K-1)C,A+KC) where
                    C = (2*INT(ABS(OMEGA))+1)*PI/ABS(OMEGA),
                    for K = 1, 2, ..., LST.
                    One can allow more cycles by increasing
                    the value of LIMLST (and taking the
                    according dimension adjustments into
                    account).
                    Examine the array IWORK which contains
                    the error flags on the cycles, in order to
                    look for eventual local integration
                    difficulties. If the position of a local
                    difficulty can be determined (e.g.
                    SINGULARITY, DISCONTINUITY within the
                    interval) one will probably gain from
                    splitting up the interval at this point
                    and calling appropriate integrators on
                    the subranges.
                = 4 The extrapolation table constructed for
                    convergence acceleration of the series
                    formed by the integral contributions over
                    the cycles, does not converge to within
                    the requested accuracy. As in the case of
                    IER = 1, it is advised to examine the
                    array IWORK which contains the error
                    flags on the cycles.
                = 6 The input is invalid because
```

```
                    (INTEGR.NE.1 AND INTEGR.NE.2) or
                     EPSABS.LE.0 or LIMLST.LT.3.
                     RESULT, ABSERR, NEVAL, LST are set
                     to zero.
                = 7 Bad integrand behaviour occurs within one
                    or more of the cycles. Location and type
                    of the difficulty involved can be
                    determined from the vector IERLST. Here
                    LST is the number of cycles actually
                    needed (see below).
                    IERLST(K) = 1 The maximum number of
                                  subdivisions (= LIMIT) has
                                  been achieved on the K th
                                  cycle.
                              = 2 Occurrence of roundoff error
                                  is detected and prevents the
                                  tolerance imposed on the
                                  K th cycle, from being
                                  achieved.
                              = 3 Extremely bad integrand
                                  behaviour occurs at some
                                  points of the K th cycle.
                              = 4 The integration procedure
                                  over the K th cycle does
                                  not converge (to within the
                                  required accuracy) due to
                                  roundoff in the
                                  extrapolation procedure
                                  invoked on this cycle. It
                                  is assumed that the result
                                  on this interval is the
                                  best which can be obtained.
                              = 5 The integral over the K th
                                  cycle is probably divergent
                                  or slowly convergent. It
                                  must be noted that
                                  divergence can occur with
                                  any other value of
                                  IERLST(K).
        If OMEGA = 0 and INTEGR = 1,
        The integral is calculated by means of DQAGIE
        and IER = IERLST(1) (with meaning as described
        for IERLST(K), K = 1).

RSLST  - Double precision
         Vector of dimension at least LIMLST
         RSLST(K) contains the integral contribution
         over the interval (A+(K-1)C,A+KC) where
         C = (2*INT(ABS(OMEGA))+1)*PI/ABS(OMEGA),
         K = 1, 2, ..., LST.
         Note that, if OMEGA = 0, RSLST(1) contains
         the value of the integral over (A,INFINITY).

ERLST  - Double precision
         Vector of dimension at least LIMLST
         ERLST(K) contains the error estimate corresponding
         with RSLST(K).

IERLST - Integer
         Vector of dimension at least LIMLST
```

```
                IERLST(K) contains the error flag corresponding
                with RSLST(K). For the meaning of the local error
                flags see description of output parameter IER.

        LST    - Integer
                Number of subintervals needed for the integration
                If OMEGA = 0 then LST is set to 1.

        ALIST, BLIST, RLIST, ELIST - Double precision
                vector of dimension at least LIMIT,

        IORD, NNLOG - Integer
                Vector of dimension at least LIMIT, providing
                space for the quantities needed in the subdivision
                process of each cycle

        CHEBMO - Double precision
                Array of dimension at least (MAXP1,25), providing
                space for the Chebyshev moments needed within the
                cycles

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DQAGIE, DQAWOE, DQELG
***REVISION HISTORY  (YYMMDD)
   800101  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   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)
   END PROLOGUE
```

DQAWO

```
      SUBROUTINE DQAWO (F, A, B, OMEGA, INTEGR, EPSABS, EPSREL, RESULT,
     +   ABSERR, NEVAL, IER, LENIW, MAXP1, LENW, LAST, IWORK, WORK)
***BEGIN PROLOGUE  DQAWO
***PURPOSE  Calculate an approximation to a given definite integral
            I= Integral of F(X)*W(X) over (A,B), where
                   W(X) = COS(OMEGA*X)
               or  W(X) = SIN(OMEGA*X),
            hopefully satisfying the following claim for accuracy
                ABS(I-RESULT).LE.MAX(EPSABS,EPSREL*ABS(I)).
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A2A1
***TYPE      DOUBLE PRECISION (QAWO-S, DQAWO-D)
***KEYWORDS  AUTOMATIC INTEGRATOR, CLENSHAW-CURTIS METHOD,
             EXTRAPOLATION, GLOBALLY ADAPTIVE,
             INTEGRAND WITH OSCILLATORY COS OR SIN FACTOR, QUADPACK,
             QUADRATURE, SPECIAL-PURPOSE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Computation of oscillatory integrals
        Standard fortran subroutine
        Double precision version

        PARAMETERS
         ON ENTRY
            F      - Double precision
                     Function subprogram defining the function
                     F(X).  The actual name for F needs to be
                     declared E X T E R N A L in the driver program.

            A      - Double precision
                     Lower limit of integration

            B      - Double precision
                     Upper limit of integration

            OMEGA  - Double precision
                     Parameter in the integrand weight function

            INTEGR - Integer
                     Indicates which of the weight functions is used
                     INTEGR = 1      W(X) = COS(OMEGA*X)
                     INTEGR = 2      W(X) = SIN(OMEGA*X)
                     If INTEGR.NE.1.AND.INTEGR.NE.2, the routine will
                     end with IER = 6.

            EPSABS - Double precision
                     Absolute accuracy requested
            EPSREL - Double precision
                     Relative accuracy requested
                     If EPSABS.LE.0 and
```

```
            EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
            the routine will end with IER = 6.

ON RETURN
   RESULT - Double precision
            Approximation to the integral

   ABSERR - Double precision
            Estimate of the modulus of the absolute error,
            which should equal or exceed ABS(I-RESULT)

   NEVAL  - Integer
            Number of integrand evaluations

   IER    - Integer
            IER = 0 Normal and reliable termination of the
                    routine. It is assumed that the requested
                    accuracy has been achieved.
          - IER.GT.0 Abnormal termination of the routine.
                    The estimates for integral and error are
                    less reliable. It is assumed that the
                    requested accuracy has not been achieved.
   ERROR MESSAGES
            IER = 1 Maximum number of subdivisions allowed
                    has been achieved (= LENIW/2). One can
                    allow more subdivisions by increasing the
                    value of LENIW (and taking the according
                    dimension adjustments into account).
                    However, if this yields no improvement it
                    is advised to analyze the integrand in
                    order to determine the integration
                    difficulties. If the position of a local
                    difficulty can be determined (e.g.
                    SINGULARITY, DISCONTINUITY within the
                    interval) one will probably gain from
                    splitting up the interval at this point
                    and calling the integrator on the
                    subranges. If possible, an appropriate
                    special-purpose integrator should be used
                    which is designed for handling the type of
                    difficulty involved.
                = 2 The occurrence of roundoff error is
                    detected, which prevents the requested
                    tolerance from being achieved.
                    The error may be under-estimated.
                = 3 Extremely bad integrand behaviour occurs
                    at some interior points of the
                    integration interval.
                = 4 The algorithm does not converge.
                    Roundoff error is detected in the
                    extrapolation table. It is presumed that
                    the requested tolerance cannot be achieved
                    due to roundoff in the extrapolation
                    table, and that the returned result is
                    the best which can be obtained.
                = 5 The integral is probably divergent, or
                    slowly convergent. It must be noted that
                    divergence can occur with any other value
                    of IER.
                = 6 The input is invalid, because
```

```
                    (EPSABS.LE.0 and
                     EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28))
                    or (INTEGR.NE.1 AND INTEGR.NE.2),
                    or LENIW.LT.2 OR MAXP1.LT.1 or
                    LENW.LT.LENIW*2+MAXP1*25.
                    RESULT, ABSERR, NEVAL, LAST are set to
                    zero. Except when LENIW, MAXP1 or LENW are
                    invalid, WORK(LIMIT*2+1), WORK(LIMIT*3+1),
                    IWORK(1), IWORK(LIMIT+1) are set to zero,
                    WORK(1) is set to A and WORK(LIMIT+1) to
                    B.

DIMENSIONING PARAMETERS
   LENIW  - Integer
            Dimensioning parameter for IWORK.
            LENIW/2 equals the maximum number of subintervals
            allowed in the partition of the given integration
            interval (A,B), LENIW.GE.2.
            If LENIW.LT.2, the routine will end with IER = 6.

   MAXP1  - Integer
            Gives an upper bound on the number of Chebyshev
            moments which can be stored, i.e. for the
            intervals of lengths ABS(B-A)*2**(-L),
            L=0,1, ..., MAXP1-2, MAXP1.GE.1
            If MAXP1.LT.1, the routine will end with IER = 6.

   LENW   - Integer
            Dimensioning parameter for WORK
            LENW must be at least LENIW*2+MAXP1*25.
            If LENW.LT.(LENIW*2+MAXP1*25), the routine will
            end with IER = 6.

   LAST   - Integer
            On return, LAST equals the number of subintervals
            produced in the subdivision process, which
            determines the number of significant elements
            actually in the WORK ARRAYS.

WORK ARRAYS
   IWORK  - Integer
            Vector of dimension at least LENIW
            on return, the first K elements of which contain
            pointers to the error estimates over the
            subintervals, such that WORK(LIMIT*3+IWORK(1)), ..
            WORK(LIMIT*3+IWORK(K)) form a decreasing
            sequence, with LIMIT = LENW/2 , and K = LAST
            if LAST.LE.(LIMIT/2+2), and K = LIMIT+1-LAST
            otherwise.
            Furthermore, IWORK(LIMIT+1), ..., IWORK(LIMIT+
            LAST) indicate the subdivision levels of the
            subintervals, such that IWORK(LIMIT+I) = L means
            that the subinterval numbered I is of length
            ABS(B-A)*2**(1-L).

   WORK   - Double precision
            Vector of dimension at least LENW
            On return
            WORK(1), ..., WORK(LAST) contain the left
             end points of the subintervals in the
```

```
                        partition of (A,B),
                       WORK(LIMIT+1), ..., WORK(LIMIT+LAST) contain
                        the right end points,
                       WORK(LIMIT*2+1), ..., WORK(LIMIT*2+LAST) contain
                        the integral approximations over the
                        subintervals,
                       WORK(LIMIT*3+1), ..., WORK(LIMIT*3+LAST)
                        contain the error estimates.
                       WORK(LIMIT*4+1), ..., WORK(LIMIT*4+MAXP1*25)
                        Provide space for storing the Chebyshev moments.
                       Note that LIMIT = LENW/2.

***REFERENCES  (NONE)
***ROUTINES CALLED  DQAWOE, XERMSG
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQAWOE

```
      SUBROUTINE DQAWOE (F, A, B, OMEGA, INTEGR, EPSABS, EPSREL, LIMIT,
     +   ICALL, MAXP1, RESULT, ABSERR, NEVAL, IER, LAST, ALIST, BLIST,
     +   RLIST, ELIST, IORD, NNLOG, MOMCOM, CHEBMO)
***BEGIN PROLOGUE  DQAWOE
***PURPOSE  Calculate an approximation to a given definite integral
            I = Integral of F(X)*W(X) over (A,B), where
                     W(X) = COS(OMEGA*X)
                 or  W(X)=SIN(OMEGA*X),
            hopefully satisfying the following claim for accuracy
                 ABS(I-RESULT).LE.MAX(EPSABS,EPSREL*ABS(I)).
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A2A1
***TYPE      DOUBLE PRECISION (QAWOE-S, DQAWOE-D)
***KEYWORDS  AUTOMATIC INTEGRATOR, CLENSHAW-CURTIS METHOD,
             EXTRAPOLATION, GLOBALLY ADAPTIVE,
             INTEGRAND WITH OSCILLATORY COS OR SIN FACTOR, QUADPACK,
             QUADRATURE, SPECIAL-PURPOSE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Computation of Oscillatory integrals
        Standard fortran subroutine
        Double precision version

        PARAMETERS
         ON ENTRY
            F      - Double precision
                     Function subprogram defining the integrand
                     function F(X). The actual name for F needs to be
                     declared E X T E R N A L in the driver program.

            A      - Double precision
                     Lower limit of integration

            B      - Double precision
                     Upper limit of integration

            OMEGA  - Double precision
                     Parameter in the integrand weight function

            INTEGR - Integer
                     Indicates which of the WEIGHT functions is to be
                     used
                     INTEGR = 1      W(X) = COS(OMEGA*X)
                     INTEGR = 2      W(X) = SIN(OMEGA*X)
                     If INTEGR.NE.1 and INTEGR.NE.2, the routine
                     will end with IER = 6.

            EPSABS - Double precision
                     Absolute accuracy requested
            EPSREL - Double precision
```

```
            Relative accuracy requested
            If  EPSABS.LE.0
            and EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
            the routine will end with IER = 6.

   LIMIT  - Integer
            Gives an upper bound on the number of subdivisions
            in the partition of (A,B), LIMIT.GE.1.

   ICALL  - Integer
            If DQAWOE is to be used only once, ICALL must
            be set to 1.  Assume that during this call, the
            Chebyshev moments (for CLENSHAW-CURTIS integration
            of degree 24) have been computed for intervals of
            lengths (ABS(B-A))*2**(-L), L=0,1,2,...MOMCOM-1.
            If ICALL.GT.1 this means that DQAWOE has been
            called twice or more on intervals of the same
            length ABS(B-A). The Chebyshev moments already
            computed are then re-used in subsequent calls.
            If ICALL.LT.1, the routine will end with IER = 6.

   MAXP1  - Integer
            Gives an upper bound on the number of Chebyshev
            moments which can be stored, i.e. for the
            intervals of lengths ABS(B-A)*2**(-L),
            L=0,1, ..., MAXP1-2, MAXP1.GE.1.
            If MAXP1.LT.1, the routine will end with IER = 6.

ON RETURN
   RESULT - Double precision
            Approximation to the integral

   ABSERR - Double precision
            Estimate of the modulus of the absolute error,
            which should equal or exceed ABS(I-RESULT)

   NEVAL  - Integer
            Number of integrand evaluations

   IER    - Integer
            IER = 0 Normal and reliable termination of the
                    routine. It is assumed that the
                    requested accuracy has been achieved.
          - IER.GT.0 Abnormal termination of the routine.
                    The estimates for integral and error are
                    less reliable. It is assumed that the
                    requested accuracy has not been achieved.
   ERROR MESSAGES
            IER = 1 Maximum number of subdivisions allowed
                    has been achieved. One can allow more
                    subdivisions by increasing the value of
                    LIMIT (and taking according dimension
                    adjustments into account). However, if
                    this yields no improvement it is advised
                    to analyze the integrand, in order to
                    determine the integration difficulties.
                    If the position of a local difficulty can
                    be determined (e.g. SINGULARITY,
                    DISCONTINUITY within the interval) one
                    will probably gain from splitting up the
```

interval at this point and calling the integrator on the subranges. If possible, an appropriate special-purpose integrator should be used which is designed for handling the type of difficulty involved.

= 2 The occurrence of roundoff error is detected, which prevents the requested tolerance from being achieved. The error may be under-estimated.

= 3 Extremely bad integrand behaviour occurs at some points of the integration interval.

= 4 The algorithm does not converge. Roundoff error is detected in the extrapolation table. It is presumed that the requested tolerance cannot be achieved due to roundoff in the extrapolation table, and that the returned result is the best which can be obtained.

= 5 The integral is probably divergent, or slowly convergent. It must be noted that divergence can occur with any other value of IER.GT.0.

= 6 The input is invalid, because (EPSABS.LE.0 and EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28)) or (INTEGR.NE.1 and INTEGR.NE.2) or ICALL.LT.1 or MAXP1.LT.1. RESULT, ABSERR, NEVAL, LAST, RLIST(1), ELIST(1), IORD(1) and NNLOG(1) are set to ZERO. ALIST(1) and BLIST(1) are set to A and B respectively.

LAST - Integer
On return, LAST equals the number of subintervals produces in the subdivision process, which determines the number of significant elements actually in the WORK ARRAYS.

ALIST - Double precision
Vector of dimension at least LIMIT, the first LAST elements of which are the left end points of the subintervals in the partition of the given integration range (A,B)

BLIST - Double precision
Vector of dimension at least LIMIT, the first LAST elements of which are the right end points of the subintervals in the partition of the given integration range (A,B)

RLIST - Double precision
Vector of dimension at least LIMIT, the first LAST elements of which are the integral approximations on the subintervals

ELIST - Double precision
Vector of dimension at least LIMIT, the first LAST elements of which are the moduli of the

```
                 absolute error estimates on the subintervals

        IORD   - Integer
                 Vector of dimension at least LIMIT, the first K
                 elements of which are pointers to the error
                 estimates over the subintervals,
                 such that ELIST(IORD(1)), ...,
                 ELIST(IORD(K)) form a decreasing sequence, with
                 K = LAST if LAST.LE.(LIMIT/2+2), and
                 K = LIMIT+1-LAST otherwise.

        NNLOG  - Integer
                 Vector of dimension at least LIMIT, containing the
                 subdivision levels of the subintervals, i.e.
                 IWORK(I) = L means that the subinterval
                 numbered I is of length ABS(B-A)*2**(1-L)

     ON ENTRY AND RETURN
        MOMCOM - Integer
                 Indicating that the Chebyshev moments
                 have been computed for intervals of lengths
                 (ABS(B-A))*2**(-L), L=0,1,2, ..., MOMCOM-1,
                 MOMCOM.LT.MAXP1

        CHEBMO - Double precision
                 Array of dimension (MAXP1,25) containing the
                 Chebyshev moments

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DQC25F, DQELG, DQPSRT
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQAWS

```
      SUBROUTINE DQAWS (F, A, B, ALFA, BETA, INTEGR, EPSABS, EPSREL,
     +   RESULT, ABSERR, NEVAL, IER, LIMIT, LENW, LAST, IWORK, WORK)
***BEGIN PROLOGUE  DQAWS
***PURPOSE  The routine calculates an approximation result to a given
            definite integral I = Integral of F*W over (A,B),
            (where W shows a singular behaviour at the end points
            see parameter INTEGR).
            Hopefully satisfying following claim for accuracy
            ABS(I-RESULT).LE.MAX(EPSABS,EPSREL*ABS(I)).
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A2A1
***TYPE      DOUBLE PRECISION (QAWS-S, DQAWS-D)
***KEYWORDS  ALGEBRAIC-LOGARITHMIC END POINT SINGULARITIES,
             AUTOMATIC INTEGRATOR, CLENSHAW-CURTIS METHOD,
             GLOBALLY ADAPTIVE, QUADPACK, QUADRATURE, SPECIAL-PURPOSE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Integration of functions having algebraico-logarithmic
        end point singularities
        Standard fortran subroutine
        Double precision version

        PARAMETERS
         ON ENTRY
            F      - Double precision
                     Function subprogram defining the integrand
                     function F(X). The actual name for F needs to be
                     declared E X T E R N A L in the driver program.

            A      - Double precision
                     Lower limit of integration

            B      - Double precision
                     Upper limit of integration, B.GT.A
                     If B.LE.A, the routine will end with IER = 6.

            ALFA   - Double precision
                     Parameter in the integrand function, ALFA.GT.(-1)
                     If ALFA.LE.(-1), the routine will end with
                     IER = 6.

            BETA   - Double precision
                     Parameter in the integrand function, BETA.GT.(-1)
                     If BETA.LE.(-1), the routine will end with
                     IER = 6.

            INTEGR - Integer
                     Indicates which WEIGHT function is to be used
                     = 1  (X-A)**ALFA*(B-X)**BETA
                     = 2  (X-A)**ALFA*(B-X)**BETA*LOG(X-A)
```

```
            = 3  (X-A)**ALFA*(B-X)**BETA*LOG(B-X)
            = 4  (X-A)**ALFA*(B-X)**BETA*LOG(X-A)*LOG(B-X)
            If INTEGR.LT.1 or INTEGR.GT.4, the routine
            will end with IER = 6.

   EPSABS - Double precision
            Absolute accuracy requested
   EPSREL - Double precision
            Relative accuracy requested
            If  EPSABS.LE.0
            and EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
            the routine will end with IER = 6.

ON RETURN
   RESULT - Double precision
            Approximation to the integral

   ABSERR - Double precision
            Estimate of the modulus of the absolute error,
            Which should equal or exceed ABS(I-RESULT)

   NEVAL  - Integer
            Number of integrand evaluations

   IER    - Integer
            IER = 0 Normal and reliable termination of the
                    routine. It is assumed that the requested
                    accuracy has been achieved.
            IER.GT.0 Abnormal termination of the routine
                    The estimates for the integral and error
                    are less reliable. It is assumed that the
                    requested accuracy has not been achieved.
   ERROR MESSAGES
            IER = 1 Maximum number of subdivisions allowed
                    has been achieved. One can allow more
                    subdivisions by increasing the value of
                    LIMIT (and taking the according dimension
                    adjustments into account). However, if
                    this yields no improvement it is advised
                    to analyze the integrand, in order to
                    determine the integration difficulties
                    which prevent the requested tolerance from
                    being achieved. In case of a jump
                    discontinuity or a local singularity
                    of algebraico-logarithmic type at one or
                    more interior points of the integration
                    range, one should proceed by splitting up
                    the interval at these points and calling
                    the integrator on the subranges.
                = 2 The occurrence of roundoff error is
                    detected, which prevents the requested
                    tolerance from being achieved.
                = 3 Extremely bad integrand behaviour occurs
                    at some points of the integration
                    interval.
                = 6 The input is invalid, because
                    B.LE.A or ALFA.LE.(-1) or BETA.LE.(-1) or
                    or INTEGR.LT.1 or INTEGR.GT.4 or
                    (EPSABS.LE.0 and
                     EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28))
```

```
                              or LIMIT.LT.2 or LENW.LT.LIMIT*4.
                              RESULT, ABSERR, NEVAL, LAST are set to
                              zero. Except when LENW or LIMIT is invalid
                              IWORK(1), WORK(LIMIT*2+1) and
                              WORK(LIMIT*3+1) are set to zero, WORK(1)
                              is set to A and WORK(LIMIT+1) to B.

            DIMENSIONING PARAMETERS
               LIMIT  - Integer
                        Dimensioning parameter for IWORK
                        LIMIT determines the maximum number of
                        subintervals in the partition of the given
                        integration interval (A,B), LIMIT.GE.2.
                        If LIMIT.LT.2, the routine will end with IER = 6.

               LENW   - Integer
                        Dimensioning parameter for WORK
                        LENW must be at least LIMIT*4.
                        If LENW.LT.LIMIT*4, the routine will end
                        with IER = 6.

               LAST   - Integer
                        On return, LAST equals the number of
                        subintervals produced in the subdivision process,
                        which determines the significant number of
                        elements actually in the WORK ARRAYS.

            WORK ARRAYS
               IWORK  - Integer
                        Vector of dimension LIMIT, the first K
                        elements of which contain pointers
                        to the error estimates over the subintervals,
                        such that WORK(LIMIT*3+IWORK(1)), ...,
                        WORK(LIMIT*3+IWORK(K)) form a decreasing
                        sequence with K = LAST if LAST.LE.(LIMIT/2+2),
                        and K = LIMIT+1-LAST otherwise

               WORK   - Double precision
                        Vector of dimension LENW
                        On return
                        WORK(1), ..., WORK(LAST) contain the left
                         end points of the subintervals in the
                         partition of (A,B),
                        WORK(LIMIT+1), ..., WORK(LIMIT+LAST) contain
                         the right end points,
                        WORK(LIMIT*2+1), ..., WORK(LIMIT*2+LAST)
                         contain the integral approximations over
                         the subintervals,
                        WORK(LIMIT*3+1), ..., WORK(LIMIT*3+LAST)
                         contain the error estimates.

***REFERENCES  (NONE)
***ROUTINES CALLED  DQAWSE, XERMSG
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQAWSE

```
      SUBROUTINE DQAWSE (F, A, B, ALFA, BETA, INTEGR, EPSABS, EPSREL,
     +   LIMIT, RESULT, ABSERR, NEVAL, IER, ALIST, BLIST, RLIST, ELIST,
     +   IORD, LAST)
***BEGIN PROLOGUE  DQAWSE
***PURPOSE  The routine calculates an approximation result to a given
            definite integral I = Integral of F*W over (A,B),
            (where W shows a singular behaviour at the end points,
            see parameter INTEGR).
            Hopefully satisfying following claim for accuracy
            ABS(I-RESULT).LE.MAX(EPSABS,EPSREL*ABS(I)).
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A2A1
***TYPE      DOUBLE PRECISION (QAWSE-S, DQAWSE-D)
***KEYWORDS  ALGEBRAIC-LOGARITHMIC END POINT SINGULARITIES,
             AUTOMATIC INTEGRATOR, CLENSHAW-CURTIS METHOD, QUADPACK,
             QUADRATURE, SPECIAL-PURPOSE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Integration of functions having algebraico-logarithmic
        end point singularities
        Standard fortran subroutine
        Double precision version

        PARAMETERS
         ON ENTRY
            F      - Double precision
                     Function subprogram defining the integrand
                     function F(X). The actual name for F needs to be
                     declared E X T E R N A L in the driver program.

            A      - Double precision
                     Lower limit of integration

            B      - Double precision
                     Upper limit of integration, B.GT.A
                     If B.LE.A, the routine will end with IER = 6.

            ALFA   - Double precision
                     Parameter in the WEIGHT function, ALFA.GT.(-1)
                     If ALFA.LE.(-1), the routine will end with
                     IER = 6.

            BETA   - Double precision
                     Parameter in the WEIGHT function, BETA.GT.(-1)
                     If BETA.LE.(-1), the routine will end with
                     IER = 6.

            INTEGR - Integer
                     Indicates which WEIGHT function is to be used
                     = 1  (X-A)**ALFA*(B-X)**BETA
```

```
            = 2  (X-A)**ALFA*(B-X)**BETA*LOG(X-A)
            = 3  (X-A)**ALFA*(B-X)**BETA*LOG(B-X)
            = 4  (X-A)**ALFA*(B-X)**BETA*LOG(X-A)*LOG(B-X)
            If INTEGR.LT.1 or INTEGR.GT.4, the routine
            will end with IER = 6.

   EPSABS - Double precision
            Absolute accuracy requested
   EPSREL - Double precision
            Relative accuracy requested
            If  EPSABS.LE.0
            and EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
            the routine will end with IER = 6.

   LIMIT  - Integer
            Gives an upper bound on the number of subintervals
            in the partition of (A,B), LIMIT.GE.2
            If LIMIT.LT.2, the routine will end with IER = 6.

ON RETURN
   RESULT - Double precision
            Approximation to the integral

   ABSERR - Double precision
            Estimate of the modulus of the absolute error,
            which should equal or exceed ABS(I-RESULT)

   NEVAL  - Integer
            Number of integrand evaluations

   IER    - Integer
            IER = 0 Normal and reliable termination of the
                    routine. It is assumed that the requested
                    accuracy has been achieved.
            IER.GT.0 Abnormal termination of the routine
                    the estimates for the integral and error
                    are less reliable. It is assumed that the
                    requested accuracy has not been achieved.
   ERROR MESSAGES
                = 1 Maximum number of subdivisions allowed
                    has been achieved. One can allow more
                    subdivisions by increasing the value of
                    LIMIT. However, if this yields no
                    improvement, it is advised to analyze the
                    integrand in order to determine the
                    integration difficulties which prevent the
                    requested tolerance from being achieved.
                    In case of a jump DISCONTINUITY or a local
                    SINGULARITY of algebraico-logarithmic type
                    at one or more interior points of the
                    integration range, one should proceed by
                    splitting up the interval at these
                    points and calling the integrator on the
                    subranges.
                = 2 The occurrence of roundoff error is
                    detected, which prevents the requested
                    tolerance from being achieved.
                = 3 Extremely bad integrand behaviour occurs
                    at some points of the integration
                    interval.
```

```
                    = 6 The input is invalid, because
                        B.LE.A or ALFA.LE.(-1) or BETA.LE.(-1), or
                        INTEGR.LT.1 or INTEGR.GT.4, or
                        (EPSABS.LE.0 and
                         EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
                        or LIMIT.LT.2.
                        RESULT, ABSERR, NEVAL, RLIST(1), ELIST(1),
                        IORD(1) and LAST are set to zero. ALIST(1)
                        and BLIST(1) are set to A and B
                        respectively.

        ALIST  - Double precision
                 Vector of dimension at least LIMIT, the first
                  LAST  elements of which are the left
                 end points of the subintervals in the partition
                 of the given integration range (A,B)

        BLIST  - Double precision
                 Vector of dimension at least LIMIT, the first
                  LAST  elements of which are the right
                 end points of the subintervals in the partition
                 of the given integration range (A,B)

        RLIST  - Double precision
                 Vector of dimension at least LIMIT, the first
                  LAST  elements of which are the integral
                 approximations on the subintervals

        ELIST  - Double precision
                 Vector of dimension at least LIMIT, the first
                  LAST  elements of which are the moduli of the
                 absolute error estimates on the subintervals

        IORD   - Integer
                 Vector of dimension at least LIMIT, the first K
                 of which are pointers to the error
                 estimates over the subintervals, so that
                 ELIST(IORD(1)), ..., ELIST(IORD(K)) with K = LAST
                 If LAST.LE.(LIMIT/2+2), and K = LIMIT+1-LAST
                 otherwise form a decreasing sequence

        LAST   - Integer
                 Number of subintervals actually produced in
                 the subdivision process

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DQC25S, DQMOMO, DQPSRT
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQC25C

```
      SUBROUTINE DQC25C (F, A, B, C, RESULT, ABSERR, KRUL, NEVAL)
***BEGIN PROLOGUE  DQC25C
***PURPOSE  To compute I = Integral of F*W over (A,B) with
            error estimate, where W(X) = 1/(X-C)
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A2A2, J4
***TYPE      DOUBLE PRECISION (QC25C-S, DQC25C-D)
***KEYWORDS  25-POINT CLENSHAW-CURTIS INTEGRATION, QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Integration rules for the computation of CAUCHY
        PRINCIPAL VALUE integrals
        Standard fortran subroutine
        Double precision version

        PARAMETERS
           F      - Double precision
                    Function subprogram defining the integrand function
                    F(X). The actual name for F needs to be declared
                    E X T E R N A L  in the driver program.

           A      - Double precision
                    Left end point of the integration interval

           B      - Double precision
                    Right end point of the integration interval, B.GT.A

           C      - Double precision
                    Parameter in the WEIGHT function

           RESULT - Double precision
                    Approximation to the integral
                    result is computed by using a generalized
                    Clenshaw-Curtis method if C lies within ten percent
                    of the integration interval. In the other case the
                    15-point Kronrod rule obtained by optimal addition
                    of abscissae to the 7-point Gauss rule, is applied.

           ABSERR - Double precision
                    Estimate of the modulus of the absolute error,
                    which should equal or exceed ABS(I-RESULT)

           KRUL   - Integer
                    Key which is decreased by 1 if the 15-point
                    Gauss-Kronrod scheme has been used

           NEVAL  - Integer
                    Number of integrand evaluations

 ......................................................................
```

```
***REFERENCES  (NONE)
***ROUTINES CALLED  DQCHEB, DQK15W, DQWGTC
***REVISION HISTORY  (YYMMDD)
   810101  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
```

DQC25F

```
      SUBROUTINE DQC25F (F, A, B, OMEGA, INTEGR, NRMOM, MAXP1, KSAVE,
     +   RESULT, ABSERR, NEVAL, RESABS, RESASC, MOMCOM, CHEBMO)
***BEGIN PROLOGUE  DQC25F
***PURPOSE  To compute the integral I=Integral of F(X) over (A,B)
            Where W(X) = COS(OMEGA*X) or W(X)=SIN(OMEGA*X) and to
            compute J = Integral of ABS(F) over (A,B). For small value
            of OMEGA or small intervals (A,B) the 15-point GAUSS-KRONRO
            Rule is used. Otherwise a generalized CLENSHAW-CURTIS
            method is used.
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A2A2
***TYPE      DOUBLE PRECISION (QC25F-S, DQC25F-D)
***KEYWORDS  CLENSHAW-CURTIS METHOD, GAUSS-KRONROD RULES,
             INTEGRATION RULES FOR FUNCTIONS WITH COS OR SIN FACTOR,
             QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Integration rules for functions with COS or SIN factor
        Standard fortran subroutine
        Double precision version

        PARAMETERS
         ON ENTRY
           F      - Double precision
                    Function subprogram defining the integrand
                    function F(X). The actual name for F needs to
                    be declared E X T E R N A L in the calling program.

           A      - Double precision
                    Lower limit of integration

           B      - Double precision
                    Upper limit of integration

           OMEGA  - Double precision
                    Parameter in the WEIGHT function

           INTEGR - Integer
                    Indicates which WEIGHT function is to be used
                       INTEGR = 1   W(X) = COS(OMEGA*X)
                       INTEGR = 2   W(X) = SIN(OMEGA*X)

           NRMOM  - Integer
                    The length of interval (A,B) is equal to the length
                    of the original integration interval divided by
                    2**NRMOM (we suppose that the routine is used in an
                    adaptive integration process, otherwise set
                    NRMOM = 0). NRMOM must be zero at the first call.

           MAXP1  - Integer
```

```
                 Gives an upper bound on the number of Chebyshev
                 moments which can be stored, i.e. for the
                 intervals of lengths ABS(BB-AA)*2**(-L),
                 L = 0,1,2, ..., MAXP1-2.

      KSAVE  - Integer
                 Key which is one when the moments for the
                 current interval have been computed

    ON RETURN
      RESULT - Double precision
                 Approximation to the integral I

      ABSERR - Double precision
                 Estimate of the modulus of the absolute
                 error, which should equal or exceed ABS(I-RESULT)

      NEVAL  - Integer
                 Number of integrand evaluations

      RESABS - Double precision
                 Approximation to the integral J

      RESASC - Double precision
                 Approximation to the integral of ABS(F-I/(B-A))

    ON ENTRY AND RETURN
      MOMCOM - Integer
                 For each interval length we need to compute the
                 Chebyshev moments. MOMCOM counts the number of
                 intervals for which these moments have already been
                 computed. If NRMOM.LT.MOMCOM or KSAVE = 1, the
                 Chebyshev moments for the interval (A,B) have
                 already been computed and stored, otherwise we
                 compute them and we increase MOMCOM.

      CHEBMO - Double precision
                 Array of dimension at least (MAXP1,25) containing
                 the modified Chebyshev moments for the first MOMCOM
                 MOMCOM interval lengths

 ......................................................................

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, DGTSL, DQCHEB, DQK15W, DQWGTF
***REVISION HISTORY  (YYMMDD)
   810101  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
```

DQC25S

```
      SUBROUTINE DQC25S (F, A, B, BL, BR, ALFA, BETA, RI, RJ, RG, RH,
     +   RESULT, ABSERR, RESASC, INTEGR, NEV)
***BEGIN PROLOGUE  DQC25S
***PURPOSE  To compute I = Integral of F*W over (BL,BR), with error
            estimate, where the weight function W has a singular
            behaviour of ALGEBRAICO-LOGARITHMIC type at the points
            A and/or B. (BL,BR) is a part of (A,B).
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A2A2
***TYPE      DOUBLE PRECISION (QC25S-S, DQC25S-D)
***KEYWORDS  25-POINT CLENSHAW-CURTIS INTEGRATION, QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Integration rules for integrands having ALGEBRAICO-LOGARITHMIC
        end point singularities
        Standard fortran subroutine
        Double precision version

        PARAMETERS
           F      - Double precision
                    Function subprogram defining the integrand
                    F(X). The actual name for F needs to be declared
                    E X T E R N A L  in the driver program.

           A      - Double precision
                    Left end point of the original interval

           B      - Double precision
                    Right end point of the original interval, B.GT.A

           BL     - Double precision
                    Lower limit of integration, BL.GE.A

           BR     - Double precision
                    Upper limit of integration, BR.LE.B

           ALFA   - Double precision
                    PARAMETER IN THE WEIGHT FUNCTION

           BETA   - Double precision
                    Parameter in the weight function

           RI,RJ,RG,RH - Double precision
                    Modified CHEBYSHEV moments for the application
                    of the generalized CLENSHAW-CURTIS
                    method (computed in subroutine DQMOMO)

           RESULT - Double precision
                    Approximation to the integral
                    RESULT is computed by using a generalized
```

```
                 CLENSHAW-CURTIS method if B1 = A or BR = B.
                 in all other cases the 15-POINT KRONROD
                 RULE is applied, obtained by optimal addition of
                 Abscissae to the 7-POINT GAUSS RULE.

        ABSERR - Double precision
                 Estimate of the modulus of the absolute error,
                 which should equal or exceed ABS(I-RESULT)

        RESASC - Double precision
                 Approximation to the integral of ABS(F*W-I/(B-A))

        INTEGR - Integer
                 Which determines the weight function
                 = 1   W(X) = (X-A)**ALFA*(B-X)**BETA
                 = 2   W(X) = (X-A)**ALFA*(B-X)**BETA*LOG(X-A)
                 = 3   W(X) = (X-A)**ALFA*(B-X)**BETA*LOG(B-X)
                 = 4   W(X) = (X-A)**ALFA*(B-X)**BETA*LOG(X-A)*
                              LOG(B-X)

        NEV    - Integer
                 Number of integrand evaluations

***REFERENCES  (NONE)
***ROUTINES CALLED  DQCHEB, DQK15W, DQWGTS
***REVISION HISTORY  (YYMMDD)
   810101  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
```

DQDOTA

```
      DOUBLE PRECISION FUNCTION DQDOTA (N, DB, QC, DX, INCX, DY, INCY)
***BEGIN PROLOGUE  DQDOTA
***PURPOSE  Compute the inner product of two vectors with extended
            precision accumulation and result.
***LIBRARY   SLATEC
***CATEGORY  D1A4
***TYPE      DOUBLE PRECISION (DQDOTA-D)
***KEYWORDS  DOT PRODUCT, INNER PRODUCT
***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)
      DB  double precision scalar to be added to inner product
      QC  extended precision scalar to be added to inner product
      DX  double precision vector with N elements
    INCX  storage spacing between elements of DX
      DY  double precision vector with N elements
    INCY  storage spacing between elements of DY

    --Output--
  DQDOTA  double precision result
      QC  extended precision result

    D.P. dot product with extended precision accumulation (and result)
    QC and DQDOTA are set = DB + QC + sum for I = 0 to N-1 of
      DX(LX+I*INCX) * DY(LY+I*INCY),  where QC is an extended
      precision result previously computed by DQDOTI or DQDOTA
      and LX = 1 if INCX .GE. 0, else LX = (-INCX)*N, and LY is
      defined in a similar way using INCY.  The MP package by
      Richard P. Brent is used for the extended precision arithmetic.

    Fred T. Krogh,  JPL,  1977,  June 1

    The common block for the MP package is name MPCOM.  If local
    variable I1 is zero, DQDOTA calls MPBLAS to initialize
    the MP package and reset I1 to 1.

    The argument QC(*) and the local variables QX and QY are INTEGER
    arrays of size 30.  See the comments in the routine MPBLAS for the
    reason for this choice.

***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  MPADD, MPBLAS, MPCDM, MPCMD, MPMUL
***COMMON BLOCKS    MPCOM
***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)
920501  Reformatted the REFERENCES section.  (WRB)
930124  Increased Array sizes for SUN -r8.  (RWC)
END PROLOGUE
```

DQDOTI

```
      DOUBLE PRECISION FUNCTION DQDOTI (N, DB, QC, DX, INCX, DY, INCY)
***BEGIN PROLOGUE  DQDOTI
***PURPOSE  Compute the inner product of two vectors with extended
            precision accumulation and result.
***LIBRARY   SLATEC
***CATEGORY  D1A4
***TYPE      DOUBLE PRECISION (DQDOTI-D)
***KEYWORDS  DOT PRODUCT, INNER PRODUCT
***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)
       DB  double precision scalar to be added to inner product
       QC  extended precision scalar to be added
       DX  double precision vector with N elements
     INCX  storage spacing between elements of DX
       DY  double precision vector with N elements
     INCY  storage spacing between elements of DY

     --Output--
   DQDOTI  double precision result
       QC  extended precision result

     D.P. dot product with extended precision accumulation (and result)
     QC and DQDOTI are set = DB + sum for I = 0 to N-1 of
       DX(LX+I*INCX) * DY(LY+I*INCY),  where QC is an extended
       precision result which can be used as input to DQDOTA,
       and LX = 1 if INCX .GE. 0, else LX = (-INCX)*N, and LY is
       defined in a similar way using INCY.  The MP package by
       Richard P. Brent is used for the extended precision arithmetic.

     Fred T. Krogh,  JPL,  1977,  June 1

     The common block for the MP package is named MPCOM.  If local
     variable I1 is zero, DQDOTI calls MPBLAS to initialize the MP
     package and reset I1 to 1.

    The argument QC(*), and the local variables QX and QY are INTEGER
    arrays of size 30.  See the comments in the routine MPBLAS for the
    reason for this choice.

***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  MPADD, MPBLAS, MPCDM, MPCMD, MPMUL
***COMMON BLOCKS    MPCOM
***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)
920501  Reformatted the REFERENCES section.  (WRB)
930124  Increased Array sizes for SUN -r8.  (RWC)
END PROLOGUE
```

DQK15

```
      SUBROUTINE DQK15 (F, A, B, RESULT, ABSERR, RESABS, RESASC)
***BEGIN PROLOGUE  DQK15
***PURPOSE  To compute I = Integral of F over (A,B), with error
                         estimate
                       J = integral of ABS(F) over (A,B)
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A1A2
***TYPE      DOUBLE PRECISION (QK15-S, DQK15-D)
***KEYWORDS  15-POINT GAUSS-KRONROD RULES, QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

           Integration rules
           Standard fortran subroutine
           Double precision version

           PARAMETERS
            ON ENTRY
              F      - Double precision
                       Function subprogram defining the integrand
                       FUNCTION F(X). The actual name for F needs to be
                       Declared E X T E R N A L in the calling program.

              A      - Double precision
                       Lower limit of integration

              B      - Double precision
                       Upper limit of integration

            ON RETURN
              RESULT - Double precision
                       Approximation to the integral I
                       Result is computed by applying the 15-POINT
                       KRONROD RULE (RESK) obtained by optimal addition
                       of abscissae to the 7-POINT GAUSS RULE(RESG).

              ABSERR - Double precision
                       Estimate of the modulus of the absolute error,
                       which should not exceed ABS(I-RESULT)

              RESABS - Double precision
                       Approximation to the integral J

              RESASC - Double precision
                       Approximation to the integral of ABS(F-I/(B-A))
                       over (A,B)

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH
***REVISION HISTORY  (YYMMDD)
   800101  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
```

DQK15I

```
      SUBROUTINE DQK15I (F, BOUN, INF, A, B, RESULT, ABSERR, RESABS,
     +   RESASC)
***BEGIN PROLOGUE  DQK15I
***PURPOSE  The original (infinite integration range is mapped
            onto the interval (0,1) and (A,B) is a part of (0,1).
            it is the purpose to compute
            I = Integral of transformed integrand over (A,B),
            J = Integral of ABS(Transformed Integrand) over (A,B).
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A3A2, H2A4A2
***TYPE      DOUBLE PRECISION (QK15I-S, DQK15I-D)
***KEYWORDS  15-POINT GAUSS-KRONROD RULES, QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

           Integration Rule
           Standard Fortran subroutine
           Double precision version

           PARAMETERS
            ON ENTRY
              F      - Double precision
                       Function subprogram defining the integrand
                       FUNCTION F(X). The actual name for F needs to be
                       Declared E X T E R N A L in the calling program.

              BOUN   - Double precision
                       Finite bound of original integration
                       Range (SET TO ZERO IF INF = +2)

              INF    - Integer
                       If INF = -1, the original interval is
                                   (-INFINITY,BOUND),
                       If INF = +1, the original interval is
                                   (BOUND,+INFINITY),
                       If INF = +2, the original interval is
                                   (-INFINITY,+INFINITY) AND
                       The integral is computed as the sum of two
                       integrals, one over (-INFINITY,0) and one over
                       (0,+INFINITY).

              A      - Double precision
                       Lower limit for integration over subrange
                       of (0,1)

              B      - Double precision
                       Upper limit for integration over subrange
                       of (0,1)

            ON RETURN
              RESULT - Double precision
```

```
                 Approximation to the integral I
                 Result is computed by applying the 15-POINT
                 KRONROD RULE(RESK) obtained by optimal addition
                 of abscissae to the 7-POINT GAUSS RULE(RESG).

        ABSERR - Double precision
                 Estimate of the modulus of the absolute error,
                 WHICH SHOULD EQUAL or EXCEED ABS(I-RESULT)

        RESABS - Double precision
                 Approximation to the integral J

        RESASC - Double precision
                 Approximation to the integral of
                 ABS((TRANSFORMED INTEGRAND)-I/(B-A)) over (A,B)

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH
***REVISION HISTORY  (YYMMDD)
   800101  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
```

DQK15W

```
      SUBROUTINE DQK15W (F, W, P1, P2, P3, P4, KP, A, B, RESULT, ABSERR,
     +   RESABS, RESASC)
***BEGIN PROLOGUE  DQK15W
***PURPOSE  To compute I = Integral of F*W over (A,B), with error
                         estimate
                       J = Integral of ABS(F*W) over (A,B)
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A2A2
***TYPE      DOUBLE PRECISION (QK15W-S, DQK15W-D)
***KEYWORDS  15-POINT GAUSS-KRONROD RULES, QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

           Integration rules
           Standard fortran subroutine
           Double precision version

           PARAMETERS
             ON ENTRY
              F      - Double precision
                       Function subprogram defining the integrand
                       function F(X). The actual name for F needs to be
                       declared E X T E R N A L in the driver program.

              W      - Double precision
                       Function subprogram defining the integrand
                       WEIGHT function W(X). The actual name for W
                       needs to be declared E X T E R N A L in the
                       calling program.

              P1, P2, P3, P4 - Double precision
                       Parameters in the WEIGHT function

              KP     - Integer
                       Key for indicating the type of WEIGHT function

              A      - Double precision
                       Lower limit of integration

              B      - Double precision
                       Upper limit of integration

            ON RETURN
              RESULT - Double precision
                       Approximation to the integral I
                       RESULT is computed by applying the 15-point
                       Kronrod rule (RESK) obtained by optimal addition
                       of abscissae to the 7-point Gauss rule (RESG).

              ABSERR - Double precision
                       Estimate of the modulus of the absolute error,
```

```
                         which should equal or exceed ABS(I-RESULT)

              RESABS - Double precision
                       Approximation to the integral of ABS(F)

              RESASC - Double precision
                       Approximation to the integral of ABS(F-I/(B-A))

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH
***REVISION HISTORY  (YYMMDD)
   810101  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
```

DQK21

```
      SUBROUTINE DQK21 (F, A, B, RESULT, ABSERR, RESABS, RESASC)
***BEGIN PROLOGUE  DQK21
***PURPOSE  To compute I = Integral of F over (A,B), with error
                         estimate
                       J = Integral of ABS(F) over (A,B)
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A1A2
***TYPE      DOUBLE PRECISION (QK21-S, DQK21-D)
***KEYWORDS  21-POINT GAUSS-KRONROD RULES, QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

           Integration rules
           Standard fortran subroutine
           Double precision version

           PARAMETERS
            ON ENTRY
              F      - Double precision
                       Function subprogram defining the integrand
                       FUNCTION F(X). The actual name for F needs to be
                       Declared E X T E R N A L in the driver program.

              A      - Double precision
                       Lower limit of integration

              B      - Double precision
                       Upper limit of integration

            ON RETURN
              RESULT - Double precision
                        Approximation to the integral I
                       RESULT is computed by applying the 21-POINT
                       KRONROD RULE (RESK) obtained by optimal addition
                       of abscissae to the 10-POINT GAUSS RULE (RESG).

              ABSERR - Double precision
                       Estimate of the modulus of the absolute error,
                       which should not exceed ABS(I-RESULT)

              RESABS - Double precision
                       Approximation to the integral J

              RESASC - Double precision
                       Approximation to the integral of ABS(F-I/(B-A))
                       over (A,B)

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH
***REVISION HISTORY  (YYMMDD)
   800101  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
```

DQK31

```
      SUBROUTINE DQK31 (F, A, B, RESULT, ABSERR, RESABS, RESASC)
***BEGIN PROLOGUE  DQK31
***PURPOSE  To compute I = Integral of F over (A,B) with error
                         estimate
                       J = Integral of ABS(F) over (A,B)
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A1A2
***TYPE      DOUBLE PRECISION (QK31-S, DQK31-D)
***KEYWORDS  31-POINT GAUSS-KRONROD RULES, QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

           Integration rules
           Standard fortran subroutine
           Double precision version

           PARAMETERS
            ON ENTRY
              F      - Double precision
                       Function subprogram defining the integrand
                       FUNCTION F(X). The actual name for F needs to be
                       Declared E X T E R N A L in the calling program.

              A      - Double precision
                       Lower limit of integration

              B      - Double precision
                       Upper limit of integration

            ON RETURN
              RESULT - Double precision
                       Approximation to the integral I
                       RESULT is computed by applying the 31-POINT
                       GAUSS-KRONROD RULE (RESK), obtained by optimal
                       addition of abscissae to the 15-POINT GAUSS
                       RULE (RESG).

              ABSERR - Double precision
                       Estimate of the modulus of the modulus,
                       which should not exceed ABS(I-RESULT)

              RESABS - Double precision
                       Approximation to the integral J

              RESASC - Double precision
                       Approximation to the integral of ABS(F-I/(B-A))
                       over (A,B)

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH
***REVISION HISTORY  (YYMMDD)
```

```
800101  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
```

DQK41

```
      SUBROUTINE DQK41 (F, A, B, RESULT, ABSERR, RESABS, RESASC)
***BEGIN PROLOGUE  DQK41
***PURPOSE  To compute I = Integral of F over (A,B), with error
                         estimate
                       J = Integral of ABS(F) over (A,B)
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A1A2
***TYPE      DOUBLE PRECISION (QK41-S, DQK41-D)
***KEYWORDS  41-POINT GAUSS-KRONROD RULES, QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

           Integration rules
           Standard fortran subroutine
           Double precision version

           PARAMETERS
            ON ENTRY
              F      - Double precision
                       Function subprogram defining the integrand
                       FUNCTION F(X). The actual name for F needs to be
                       declared E X T E R N A L in the calling program.

              A      - Double precision
                       Lower limit of integration

              B      - Double precision
                       Upper limit of integration

            ON RETURN
              RESULT - Double precision
                       Approximation to the integral I
                       RESULT is computed by applying the 41-POINT
                       GAUSS-KRONROD RULE (RESK) obtained by optimal
                       addition of abscissae to the 20-POINT GAUSS
                       RULE (RESG).

              ABSERR - Double precision
                       Estimate of the modulus of the absolute error,
                       which should not exceed ABS(I-RESULT)

              RESABS - Double precision
                       Approximation to the integral J

              RESASC - Double precision
                       Approximation to the integral of ABS(F-I/(B-A))
                       over (A,B)

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH
***REVISION HISTORY  (YYMMDD)
```

```
800101  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
```

DQK51

```
      SUBROUTINE DQK51 (F, A, B, RESULT, ABSERR, RESABS, RESASC)
***BEGIN PROLOGUE  DQK51
***PURPOSE  To compute I = Integral of F over (A,B) with error
                         estimate
                       J = Integral of ABS(F) over (A,B)
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A1A2
***TYPE      DOUBLE PRECISION (QK51-S, DQK51-D)
***KEYWORDS  51-POINT GAUSS-KRONROD RULES, QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

           Integration rules
           Standard fortran subroutine
           Double precision version

           PARAMETERS
            ON ENTRY
              F      - Double precision
                       Function subroutine defining the integrand
                       function F(X). The actual name for F needs to be
                       declared E X T E R N A L in the calling program.

              A      - Double precision
                       Lower limit of integration

              B      - Double precision
                       Upper limit of integration

            ON RETURN
              RESULT - Double precision
                       Approximation to the integral I
                       RESULT is computed by applying the 51-point
                       Kronrod rule (RESK) obtained by optimal addition
                       of abscissae to the 25-point Gauss rule (RESG).

              ABSERR - Double precision
                       Estimate of the modulus of the absolute error,
                       which should not exceed ABS(I-RESULT)

              RESABS - Double precision
                       Approximation to the integral J

              RESASC - Double precision
                       Approximation to the integral of ABS(F-I/(B-A))
                       over (A,B)

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH
***REVISION HISTORY  (YYMMDD)
   800101  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 WGK(26) to code.  (WRB)
END PROLOGUE
```

DQK61

```
      SUBROUTINE DQK61 (F, A, B, RESULT, ABSERR, RESABS, RESASC)
***BEGIN PROLOGUE  DQK61
***PURPOSE  To compute I = Integral of F over (A,B) with error
                         estimate
                     J = Integral of ABS(F) over (A,B)
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A1A2
***TYPE      DOUBLE PRECISION (QK61-S, DQK61-D)
***KEYWORDS  61-POINT GAUSS-KRONROD RULES, QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        Integration rule
        Standard fortran subroutine
        Double precision version


        PARAMETERS
         ON ENTRY
           F      - Double precision
                    Function subprogram defining the integrand
                    function F(X). The actual name for F needs to be
                    declared E X T E R N A L in the calling program.

           A      - Double precision
                    Lower limit of integration

           B      - Double precision
                    Upper limit of integration

         ON RETURN
           RESULT - Double precision
                    Approximation to the integral I
                    RESULT is computed by applying the 61-point
                    Kronrod rule (RESK) obtained by optimal addition of
                    abscissae to the 30-point Gauss rule (RESG).

           ABSERR - Double precision
                    Estimate of the modulus of the absolute error,
                    which should equal or exceed ABS(I-RESULT)

           RESABS - Double precision
                    Approximation to the integral J

           RESASC - Double precision
                    Approximation to the integral of ABS(F-I/(B-A))

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH
***REVISION HISTORY  (YYMMDD)
   800101  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
```

DQMOMO

```
      SUBROUTINE DQMOMO (ALFA, BETA, RI, RJ, RG, RH, INTEGR)
***BEGIN PROLOGUE  DQMOMO
***PURPOSE  This routine computes modified Chebyshev moments.  The K-th
            modified Chebyshev moment is defined as the integral over
            (-1,1) of W(X)*T(K,X), where T(K,X) is the Chebyshev
            polynomial of degree K.
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A2A1, C3A2
***TYPE      DOUBLE PRECISION (QMOMO-S, DQMOMO-D)
***KEYWORDS  MODIFIED CHEBYSHEV MOMENTS, QUADPACK, QUADRATURE
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

        MODIFIED CHEBYSHEV MOMENTS
        STANDARD FORTRAN SUBROUTINE
        DOUBLE PRECISION VERSION

        PARAMETERS
           ALFA   - Double precision
                    Parameter in the weight function W(X), ALFA.GT.(-1)

           BETA   - Double precision
                    Parameter in the weight function W(X), BETA.GT.(-1)

           RI     - Double precision
                    Vector of dimension 25
                    RI(K) is the integral over (-1,1) of
                    (1+X)**ALFA*T(K-1,X), K = 1, ..., 25.

           RJ     - Double precision
                    Vector of dimension 25
                    RJ(K) is the integral over (-1,1) of
                    (1-X)**BETA*T(K-1,X), K = 1, ..., 25.

           RG     - Double precision
                    Vector of dimension 25
                    RG(K) is the integral over (-1,1) of
                    (1+X)**ALFA*LOG((1+X)/2)*T(K-1,X), K = 1, ..., 25.

           RH     - Double precision
                    Vector of dimension 25
                    RH(K) is the integral over (-1,1) of
                    (1-X)**BETA*LOG((1-X)/2)*T(K-1,X), K = 1, ..., 25.

           INTEGR - Integer
                    Input parameter indicating the modified
                    Moments to be computed
                    INTEGR = 1 compute RI, RJ
                           = 2 compute RI, RJ, RG
                           = 3 compute RI, RJ, RH
                           = 4 compute RI, RJ, RG, RH
```

```
***REFERENCES  (NONE)
***ROUTINES CALLED  (NONE)
***REVISION HISTORY  (YYMMDD)
   820101  DATE WRITTEN
   891009  Removed unreferenced statement label.  (WRB)
   891009  REVISION DATE from Version 3.2
   891214  Prologue converted to Version 4.0 format.  (BAB)
   END PROLOGUE
```

DQNC79

```
      SUBROUTINE DQNC79 (FUN, A, B, ERR, ANS, IERR, K)
***BEGIN PROLOGUE  DQNC79
***PURPOSE  Integrate a function using a 7-point adaptive Newton-Cotes
            quadrature rule.
***LIBRARY   SLATEC
***CATEGORY  H2A1A1
***TYPE      DOUBLE PRECISION (QNC79-S, DQNC79-D)
***KEYWORDS  ADAPTIVE QUADRATURE, INTEGRATION, NEWTON-COTES
***AUTHOR  Kahaner, D. K., (NBS)
           Jones, R. E., (SNLA)
***DESCRIPTION

      Abstract  *** a DOUBLE PRECISION routine ***
        DQNC79 is a general purpose program for evaluation of
        one dimensional integrals of user defined functions.
        DQNC79 will pick its own points for evaluation of the
        integrand and these will vary from problem to problem.
        Thus, DQNC79 is not designed to integrate over data sets.
        Moderately smooth integrands will be integrated efficiently
        and reliably.  For problems with strong singularities,
        oscillations etc., the user may wish to use more sophis-
        ticated routines such as those in QUADPACK.  One measure
        of the reliability of DQNC79 is the output parameter K,
        giving the number of integrand evaluations that were needed.

      Description of Arguments

      --Input--* FUN, A, B, ERR are DOUBLE PRECISION *
        FUN  - name of external function to be integrated.  This name
               must be in an EXTERNAL statement in your calling
               program.  You must write a Fortran function to evaluate
               FUN.  This should be of the form
                     DOUBLE PRECISION FUNCTION FUN (X)
               C
               C     X can vary from A to B
               C     FUN(X) should be finite for all X on interval.
               C
                     FUN = ...
                     RETURN
                     END
        A    - lower limit of integration
        B    - upper limit of integration (may be less than A)
        ERR  - is a requested error tolerance.  Normally, pick a value
               0 .LT. ERR .LT. 1.0D-8.

      --Output--
        ANS  - computed value of the integral.  Hopefully, ANS is
               accurate to within ERR * integral of ABS(FUN(X)).
        IERR - a status code
             - Normal codes
                1  ANS most likely meets requested error tolerance.
               -1  A equals B, or A and B are too nearly equal to
                   allow normal integration.  ANS is set to zero.
             - Abnormal code
                2  ANS probably does not meet requested error tolerance.
        K    - the number of function evaluations actually used to do
```

```
              the integration.  A value of K .GT. 1000 indicates a
              difficult problem; other programs may be more efficient.
              DQNC79 will gracefully give up if K exceeds 2000.

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, I1MACH, XERMSG
***REVISION HISTORY  (YYMMDD)
   790601  DATE WRITTEN
   890531  Changed all specific intrinsics to generic.  (WRB)
   890911  Removed unnecessary intrinsics.  (WRB)
   890911  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)
   920218  Code redone to parallel QNC79.  (WRB)
   930120  Increase array size 80->99, and KMX 2000->5000 for SUN -r8
           wordlength.  (RWC)
   END PROLOGUE
```

DQNG

```
      SUBROUTINE DQNG (F, A, B, EPSABS, EPSREL, RESULT, ABSERR, NEVAL,
     +   IER)
***BEGIN PROLOGUE  DQNG
***PURPOSE  The routine calculates an approximation result to a
            given definite integral I = integral of F over (A,B),
            hopefully satisfying following claim for accuracy
            ABS(I-RESULT).LE.MAX(EPSABS,EPSREL*ABS(I)).
***LIBRARY   SLATEC (QUADPACK)
***CATEGORY  H2A1A1
***TYPE      DOUBLE PRECISION (QNG-S, DQNG-D)
***KEYWORDS  AUTOMATIC INTEGRATOR, GAUSS-KRONROD(PATTERSON) RULES,
             NONADAPTIVE, QUADPACK, QUADRATURE, SMOOTH INTEGRAND
***AUTHOR  Piessens, Robert
             Applied Mathematics and Programming Division
             K. U. Leuven
           de Doncker, Elise
             Applied Mathematics and Programming Division
             K. U. Leuven
***DESCRIPTION

 NON-ADAPTIVE INTEGRATION
 STANDARD FORTRAN SUBROUTINE
 DOUBLE PRECISION VERSION

           F      - Double precision
                    Function subprogram defining the integrand function
                    F(X). The actual name for F needs to be declared
                    E X T E R N A L in the driver program.

           A      - Double precision
                    Lower limit of integration

           B      - Double precision
                    Upper limit of integration

           EPSABS - Double precision
                    Absolute accuracy requested
           EPSREL - Double precision
                    Relative accuracy requested
                    If  EPSABS.LE.0
                    And EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28),
                    The routine will end with IER = 6.

         ON RETURN
           RESULT - Double precision
                    Approximation to the integral I
                    Result is obtained by applying the 21-POINT
                    GAUSS-KRONROD RULE (RES21) obtained by optimal
                    addition of abscissae to the 10-POINT GAUSS RULE
                    (RES10), or by applying the 43-POINT RULE (RES43)
                    obtained by optimal addition of abscissae to the
                    21-POINT GAUSS-KRONROD RULE, or by applying the
                    87-POINT RULE (RES87) obtained by optimal addition
                    of abscissae to the 43-POINT RULE.

           ABSERR - Double precision
```

```
                 Estimate of the modulus of the absolute error,
                 which should EQUAL or EXCEED ABS(I-RESULT)

        NEVAL  - Integer
                 Number of integrand evaluations

        IER    - IER = 0 normal and reliable termination of the
                         routine. It is assumed that the requested
                         accuracy has been achieved.
                 IER.GT.0 Abnormal termination of the routine. It is
                         assumed that the requested accuracy has
                         not been achieved.
        ERROR MESSAGES
                 IER = 1 The maximum number of steps has been
                         executed. The integral is probably too
                         difficult to be calculated by DQNG.
                     = 6 The input is invalid, because
                         EPSABS.LE.0 AND
                         EPSREL.LT.MAX(50*REL.MACH.ACC.,0.5D-28).
                         RESULT, ABSERR and NEVAL are set to zero.

***REFERENCES  (NONE)
***ROUTINES CALLED  D1MACH, XERMSG
***REVISION HISTORY  (YYMMDD)
   800101  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)
   END PROLOGUE
```

DQRDC

```
      SUBROUTINE DQRDC (X, LDX, N, P, QRAUX, JPVT, WORK, JOB)
***BEGIN PROLOGUE  DQRDC
***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      DOUBLE PRECISION (SQRDC-S, DQRDC-D, CQRDC-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX, ORTHOGONAL TRIANGULAR,
             QR DECOMPOSITION
***AUTHOR  Stewart, G. W., (U. of Maryland)
***DESCRIPTION

      DQRDC 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       DOUBLE PRECISION(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    DOUBLE PRECISION(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   DOUBLE PRECISION(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  DAXPY, DDOT, DNRM2, DSCAL, DSWAP
***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
```

DQRSL

```
      SUBROUTINE DQRSL (X, LDX, N, K, QRAUX, Y, QY, QTY, B, RSD, XB,
     +   JOB, INFO)
***BEGIN PROLOGUE  DQRSL
***PURPOSE  Apply the output of DQRDC to compute coordinate transfor-
            mations, projections, and least squares solutions.
***LIBRARY   SLATEC (LINPACK)
***CATEGORY  D9, D2A1
***TYPE      DOUBLE PRECISION (SQRSL-S, DQRSL-D, CQRSL-C)
***KEYWORDS  LINEAR ALGEBRA, LINPACK, MATRIX, ORTHOGONAL TRIANGULAR,
             SOLVE
***AUTHOR  Stewart, G. W., (U. of Maryland)
***DESCRIPTION

      DQRSL applies the output of DQRDC 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 DQRDC (if no pivoting was
      done, XK consists of the first K columns of X in their
      original order).  DQRDC 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      DOUBLE PRECISION(LDX,P).
                X contains the output of DQRDC.

         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 DQRDC.

         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 DQRDC.

         QRAUX  DOUBLE PRECISION(P).
                QRAUX contains the auxiliary output from DQRDC.

         Y      DOUBLE PRECISION(N)
                Y contains an N-vector that is to be manipulated
                by DQRSL.

         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 DOUBLE PRECISION(N).
QY contains Q*Y, if its computation has been requested.

QTY DOUBLE PRECISION(N).
QTY contains TRANS(Q)*Y, if its computation has been requested. Here TRANS(Q) is the transpose of the matrix Q.

B DOUBLE PRECISION(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 DQRDC, the J-th component of B will be associated with column JPVT(J) of the original matrix X that was input into DQRDC.)

RSD DOUBLE PRECISION(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 DOUBLE PRECISION(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 DQRSL(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  DAXPY, DCOPY, DDOT
***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
```

DRC

```
      DOUBLE PRECISION FUNCTION DRC (X, Y, IER)
***BEGIN PROLOGUE  DRC
***PURPOSE  Calculate a double precision approximation to
             DRC(X,Y) = Integral from zero to infinity of
                                -1/2     -1
                    (1/2)(t+X)    (t+Y)  dt,
            where X is nonnegative and Y is positive.
***LIBRARY   SLATEC
***CATEGORY  C14
***TYPE      DOUBLE PRECISION (RC-S, DRC-D)
***KEYWORDS  DUPLICATION THEOREM, ELEMENTARY FUNCTIONS,
             ELLIPTIC INTEGRAL, 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.     DRC
          Standard FORTRAN function routine
          Double precision version
          The routine calculates an approximation result to
          DRC(X,Y) = integral from zero to infinity of

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

          where X is nonnegative and Y is positive.  The duplication
          theorem is iterated until the variables are nearly equal,
          and the function is then expanded in Taylor series to fifth
          order.  Logarithmic, inverse circular, and inverse hyper-
          bolic functions can be expressed in terms of DRC.

   2.     Calling Sequence
          DRC( X, Y, IER )

          Parameters On Entry
          Values assigned by the calling routine

          X      - Double precision, nonnegative variable

          Y      - Double precision, positive variable


          On Return  (values assigned by the DRC routine)

          DRC    - Double precision approximation to the integral
```

IER - Integer to indicate normal or abnormal termination.

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 and Y are unaltered.

3. Error messages

Value of IER assigned by the DRC routine

Value assigned	Error message printed
IER = 1	X.LT.0.0D0.OR.Y.LE.0.0D0
= 2	X+Y.LT.LOLIM
= 3	MAX(X,Y) .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 and Y

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 :	3.0D-78	1.0D+75
CDC 6000/7000 SERIES :	1.0D-292	1.0D+321
UNIVAC 1100 SERIES :	1.0D-307	1.0D+307
CRAY :	2.3D-2466	1.0D+2465
VAX 11 SERIES :	1.5D-38	3.0D+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 16 * ERRTOL ** 6 / (1 - 2 * 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 prac-

tice the total error from both sources is usually less than the amount given in the table.

Sample choices:

ERRTOL	Relative truncation error less than
1.0D-3	2.0D-17
3.0D-3	2.0D-14
1.0D-2	2.0D-11
3.0D-2	2.0D-8
1.0D-1	2.0D-5

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:

DRC special comments

Check: DRC(X,X+Z) + DRC(Y,Y+Z) = DRC(0,Z)

where X, Y, and Z are positive and X * Y = Z * Z

On Input:

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

On Output:

X and Y are unaltered.

DRC(0,1/4)=DRC(1/16,1/8)=PI=3.14159...

DRC(9/4,2)=LN(2)

****

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

Special functions via DRC

LN X X .GT. 0

```
                                        2
LN(X) = (X-1) DRC(((1+X)/2)  , X )

-----------------------------------------------------------------

ARCSIN X            -1 .LE. X .LE. 1

                                 2
ARCSIN X = X DRC (1-X  ,1 )

-----------------------------------------------------------------

ARCCOS X            0 .LE. X .LE. 1

                            2       2
ARCCOS X = SQRT(1-X ) DRC(X  ,1 )

-----------------------------------------------------------------

ARCTAN X            -INF .LT. X .LT. +INF

                                   2
ARCTAN X = X DRC(1,1+X  )

-----------------------------------------------------------------

ARCCOT X            0 .LE. X .LT. INF

                        2   2
ARCCOT X = DRC(X  ,X +1 )

-----------------------------------------------------------------

ARCSINH X           -INF .LT. X .LT. +INF

                                 2
ARCSINH X = X DRC(1+X  ,1 )

-----------------------------------------------------------------

ARCCOSH X           X .GE. 1

                           2         2
ARCCOSH X = SQRT(X -1) DRC(X  ,1 )

-----------------------------------------------------------------

ARCTANH X           -1 .LT. X .LT. 1

                                   2
ARCTANH X = X DRC(1,1-X  )

-----------------------------------------------------------------

ARCCOTH X           X .GT. 1

                        2   2
ARCCOTH X = DRC(X  ,X -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  D1MACH, 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
```

DRC3JJ

```
      SUBROUTINE DRC3JJ (L2, L3, M2, M3, L1MIN, L1MAX, THRCOF, NDIM,
     +   IER)
***BEGIN PROLOGUE  DRC3JJ
***PURPOSE  Evaluate the 3j symbol f(L1) = (  L1   L2 L3)
                                           (-M2-M3 M2 M3)
            for all allowed values of L1, the other parameters
            being held fixed.
***LIBRARY   SLATEC
***CATEGORY  C19
***TYPE      DOUBLE PRECISION (RC3JJ-S, DRC3JJ-D)
***KEYWORDS  3J COEFFICIENTS, 3J SYMBOLS, CLEBSCH-GORDAN COEFFICIENTS,
             RACAH COEFFICIENTS, VECTOR ADDITION COEFFICIENTS,
             WIGNER COEFFICIENTS
***AUTHOR  Gordon, R. G., Harvard University
           Schulten, K., Max Planck Institute
***DESCRIPTION

 *Usage:

        DOUBLE PRECISION L2, L3, M2, M3, L1MIN, L1MAX, THRCOF(NDIM)
        INTEGER NDIM, IER

        CALL DRC3JJ (L2, L3, M2, M3, L1MIN, L1MAX, THRCOF, NDIM, IER)

 *Arguments:

     L2 :IN      Parameter in 3j symbol.

     L3 :IN      Parameter in 3j symbol.

     M2 :IN      Parameter in 3j symbol.

     M3 :IN      Parameter in 3j symbol.

     L1MIN :OUT  Smallest allowable L1 in 3j symbol.

     L1MAX :OUT  Largest allowable L1 in 3j symbol.

     THRCOF :OUT Set of 3j coefficients generated by evaluating the
                 3j symbol for all allowed values of L1.  THRCOF(I)
                 will contain f(L1MIN+I-1), I=1,2,...,L1MAX+L1MIN+1.

     NDIM :IN    Declared length of THRCOF in calling program.

     IER :OUT    Error flag.
                 IER=0 No errors.
                 IER=1 Either L2.LT.ABS(M2) or L3.LT.ABS(M3).
                 IER=2 Either L2+ABS(M2) or L3+ABS(M3) non-integer.
                 IER=3 L1MAX-L1MIN not an integer.
                 IER=4 L1MAX less than L1MIN.
                 IER=5 NDIM less than L1MAX-L1MIN+1.

 *Description:

    Although conventionally the parameters of the vector addition
  coefficients satisfy certain restrictions, such as being integers
```

or integers plus 1/2, the restrictions imposed on input to this subroutine are somewhat weaker. See, for example, Section 27.9 of Abramowitz and Stegun or Appendix C of Volume II of A. Messiah. The restrictions imposed by this subroutine are

1. L2 .GE. ABS(M2) and L3 .GE. ABS(M3);
2. L2+ABS(M2) and L3+ABS(M3) must be integers;
3. L1MAX-L1MIN must be a non-negative integer, where L1MAX=L2+L3 and L1MIN=MAX(ABS(L2-L3),ABS(M2+M3)).

If the conventional restrictions are satisfied, then these restrictions are met.

The user should be cautious in using input parameters that do not satisfy the conventional restrictions. For example, the the subroutine produces values of

```
     f(L1) = ( L1  2.5  5.8)
             (-0.3 1.5 -1.2)
```

for L1=3.3,4.3,...,8.3 but none of the symmetry properties of the 3j symbol, set forth on page 1056 of Messiah, is satisfied.

The subroutine generates f(L1MIN), f(L1MIN+1), ..., f(L1MAX) where L1MIN and L1MAX are defined above. The sequence f(L1) is generated by a three-term recurrence algorithm with scaling to control overflow. Both backward and forward recurrence are used to maintain numerical stability. The two recurrence sequences are matched at an interior point and are normalized from the unitary property of 3j coefficients and Wigner's phase convention.

The algorithm is suited to applications in which large quantum numbers arise, such as in molecular dynamics.

***REFERENCES

1. Abramowitz, M., and Stegun, I. A., Eds., Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables, NBS Applied Mathematics Series 55, June 1964 and subsequent printings.
2. Messiah, Albert., Quantum Mechanics, Volume II, North-Holland Publishing Company, 1963.
3. Schulten, Klaus and Gordon, Roy G., Exact recursive evaluation of 3j and 6j coefficients for quantum-mechanical coupling of angular momenta, J Math Phys, v 16, no. 10, October 1975, pp. 1961-1970.
4. Schulten, Klaus and Gordon, Roy G., Semiclassical approximations to 3j and 6j coefficients for quantum-mechanical coupling of angular momenta, J Math Phys, v 16, no. 10, October 1975, pp. 1971-1988.
5. Schulten, Klaus and Gordon, Roy G., Recursive evaluation of 3j and 6j coefficients, Computer Phys Comm, v 11, 1976, pp. 269-278.

***ROUTINES CALLED D1MACH, XERMSG

***REVISION HISTORY (YYMMDD)

750101 DATE WRITTEN

880515 SLATEC prologue added by G. C. Nielson, NBS; parameters HUGE and TINY revised to depend on D1MACH.

891229 Prologue description rewritten; other prologue sections revised; LMATCH (location of match point for recurrences) removed from argument list; argument IER changed to serve only as an error flag (previously, in cases without error, it returned the number of scalings); number of error codes increased to provide more precise error information; program comments revised; SLATEC error handler calls

```
        introduced to enable printing of error messages to meet
        SLATEC standards. These changes were done by D. W. Lozier,
        M. A. McClain and J. M. Smith of the National Institute
        of Standards and Technology, formerly NBS.
910415  Mixed type expressions eliminated; variable C1 initialized;
        description of THRCOF expanded. These changes were done by
        D. W. Lozier.
END PROLOGUE
```

DRC3JM

```
      SUBROUTINE DRC3JM (L1, L2, L3, M1, M2MIN, M2MAX, THRCOF, NDIM,
     +   IER)
***BEGIN PROLOGUE  DRC3JM
***PURPOSE  Evaluate the 3j symbol g(M2) = (L1 L2   L3  )
                                           (M1 M2 -M1-M2)
            for all allowed values of M2, the other parameters
            being held fixed.
***LIBRARY   SLATEC
***CATEGORY  C19
***TYPE      DOUBLE PRECISION (RC3JM-S, DRC3JM-D)
***KEYWORDS  3J COEFFICIENTS, 3J SYMBOLS, CLEBSCH-GORDAN COEFFICIENTS,
             RACAH COEFFICIENTS, VECTOR ADDITION COEFFICIENTS,
             WIGNER COEFFICIENTS
***AUTHOR  Gordon, R. G., Harvard University
           Schulten, K., Max Planck Institute
***DESCRIPTION

 *Usage:

        DOUBLE PRECISION L1, L2, L3, M1, M2MIN, M2MAX, THRCOF(NDIM)
        INTEGER NDIM, IER

        CALL DRC3JM (L1, L2, L3, M1, M2MIN, M2MAX, THRCOF, NDIM, IER)

 *Arguments:

     L1 :IN      Parameter in 3j symbol.

     L2 :IN      Parameter in 3j symbol.

     L3 :IN      Parameter in 3j symbol.

     M1 :IN      Parameter in 3j symbol.

     M2MIN :OUT  Smallest allowable M2 in 3j symbol.

     M2MAX :OUT  Largest allowable M2 in 3j symbol.

     THRCOF :OUT Set of 3j coefficients generated by evaluating the
                 3j symbol for all allowed values of M2.  THRCOF(I)
                 will contain g(M2MIN+I-1), I=1,2,...,M2MAX-M2MIN+1.

     NDIM :IN    Declared length of THRCOF in calling program.

     IER :OUT    Error flag.
                 IER=0 No errors.
                 IER=1 Either L1.LT.ABS(M1) or L1+ABS(M1) non-integer.
                 IER=2 ABS(L1-L2).LE.L3.LE.L1+L2 not satisfied.
                 IER=3 L1+L2+L3 not an integer.
                 IER=4 M2MAX-M2MIN not an integer.
                 IER=5 M2MAX less than M2MIN.
                 IER=6 NDIM less than M2MAX-M2MIN+1.

 *Description:

     Although conventionally the parameters of the vector addition
```

coefficients satisfy certain restrictions, such as being integers or integers plus 1/2, the restrictions imposed on input to this subroutine are somewhat weaker. See, for example, Section 27.9 of Abramowitz and Stegun or Appendix C of Volume II of A. Messiah. The restrictions imposed by this subroutine are

1. L1.GE.ABS(M1) and L1+ABS(M1) must be an integer;
2. ABS(L1-L2).LE.L3.LE.L1+L2;
3. L1+L2+L3 must be an integer;
4. M2MAX-M2MIN must be an integer, where
   M2MAX=MIN(L2,L3-M1) and M2MIN=MAX(-L2,-L3-M1).

If the conventional restrictions are satisfied, then these restrictions are met.

The user should be cautious in using input parameters that do not satisfy the conventional restrictions. For example, the the subroutine produces values of

```
     g(M2) = (0.75 1.50   1.75  )
             (0.25  M2  -0.25-M2)
```

for M2=-1.5,-0.5,0.5,1.5 but none of the symmetry properties of the 3j symbol, set forth on page 1056 of Messiah, is satisfied.

The subroutine generates g(M2MIN), g(M2MIN+1), ..., g(M2MAX) where M2MIN and M2MAX are defined above. The sequence g(M2) is generated by a three-term recurrence algorithm with scaling to control overflow. Both backward and forward recurrence are used to maintain numerical stability. The two recurrence sequences are matched at an interior point and are normalized from the unitary property of 3j coefficients and Wigner's phase convention.

The algorithm is suited to applications in which large quantum numbers arise, such as in molecular dynamics.

***REFERENCES

1. Abramowitz, M., and Stegun, I. A., Eds., Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables, NBS Applied Mathematics Series 55, June 1964 and subsequent printings.
2. Messiah, Albert., Quantum Mechanics, Volume II, North-Holland Publishing Company, 1963.
3. Schulten, Klaus and Gordon, Roy G., Exact recursive evaluation of 3j and 6j coefficients for quantum-mechanical coupling of angular momenta, J Math Phys, v 16, no. 10, October 1975, pp. 1961-1970.
4. Schulten, Klaus and Gordon, Roy G., Semiclassical approximations to 3j and 6j coefficients for quantum-mechanical coupling of angular momenta, J Math Phys, v 16, no. 10, October 1975, pp. 1971-1988.
5. Schulten, Klaus and Gordon, Roy G., Recursive evaluation of 3j and 6j coefficients, Computer Phys Comm, v 11, 1976, pp. 269-278.

***ROUTINES CALLED D1MACH, XERMSG

***REVISION HISTORY (YYMMDD)

750101 DATE WRITTEN

880515 SLATEC prologue added by G. C. Nielson, NBS; parameters HUGE and TINY revised to depend on D1MACH.

891229 Prologue description rewritten; other prologue sections revised; MMATCH (location of match point for recurrences) removed from argument list; argument IER changed to serve only as an error flag (previously, in cases without error, it returned the number of scalings); number of error codes

```
        increased to provide more precise error information;
        program comments revised; SLATEC error handler calls
        introduced to enable printing of error messages to meet
        SLATEC standards. These changes were done by D. W. Lozier,
        M. A. McClain and J. M. Smith of the National Institute
        of Standards and Technology, formerly NBS.
910415  Mixed type expressions eliminated; variable C1 initialized;
        description of THRCOF expanded. These changes were done by
        D. W. Lozier.
END PROLOGUE
```

DRC6J

```
      SUBROUTINE DRC6J(L2, L3, L4, L5, L6, L1MIN, L1MAX, SIXCOF, NDIM,
     +   IER)
***BEGIN PROLOGUE  DRC6J
***PURPOSE  Evaluate the 6j symbol h(L1) = {L1 L2 L3}
                                           {L4 L5 L6}
            for all allowed values of L1, the other parameters
            being held fixed.
***LIBRARY   SLATEC
***CATEGORY  C19
***TYPE      DOUBLE PRECISION (RC6J-S, DRC6J-D)
***KEYWORDS  6J COEFFICIENTS, 6J SYMBOLS, CLEBSCH-GORDAN COEFFICIENTS,
             RACAH COEFFICIENTS, VECTOR ADDITION COEFFICIENTS,
             WIGNER COEFFICIENTS
***AUTHOR  Gordon, R. G., Harvard University
           Schulten, K., Max Planck Institute
***DESCRIPTION

 *Usage:

        DOUBLE PRECISION L2, L3, L4, L5, L6, L1MIN, L1MAX, SIXCOF(NDIM)
        INTEGER NDIM, IER

        CALL DRC6J(L2, L3, L4, L5, L6, L1MIN, L1MAX, SIXCOF, NDIM, IER)

 *Arguments:

     L2 :IN      Parameter in 6j symbol.

     L3 :IN      Parameter in 6j symbol.

     L4 :IN      Parameter in 6j symbol.

     L5 :IN      Parameter in 6j symbol.

     L6 :IN      Parameter in 6j symbol.

     L1MIN :OUT  Smallest allowable L1 in 6j symbol.

     L1MAX :OUT  Largest allowable L1 in 6j symbol.

     SIXCOF :OUT Set of 6j coefficients generated by evaluating the
                 6j symbol for all allowed values of L1.  SIXCOF(I)
                 will contain h(L1MIN+I-1), I=1,2,...,L1MAX-L1MIN+1.

     NDIM :IN    Declared length of SIXCOF in calling program.

     IER :OUT    Error flag.
                 IER=0 No errors.
                 IER=1 L2+L3+L5+L6 or L4+L2+L6 not an integer.
                 IER=2 L4, L2, L6 triangular condition not satisfied.
                 IER=3 L4, L5, L3 triangular condition not satisfied.
                 IER=4 L1MAX-L1MIN not an integer.
                 IER=5 L1MAX less than L1MIN.
                 IER=6 NDIM less than L1MAX-L1MIN+1.

 *Description:
```

The definition and properties of 6j symbols can be found, for example, in Appendix C of Volume II of A. Messiah. Although the parameters of the vector addition coefficients satisfy certain conventional restrictions, the restriction that they be non-negative integers or non-negative integers plus 1/2 is not imposed on input to this subroutine. The restrictions imposed are

1. L2+L3+L5+L6 and L2+L4+L6 must be integers;
2. ABS(L2-L4).LE.L6.LE.L2+L4 must be satisfied;
3. ABS(L4-L5).LE.L3.LE.L4+L5 must be satisfied;
4. L1MAX-L1MIN must be a non-negative integer, where L1MAX=MIN(L2+L3,L5+L6) and L1MIN=MAX(ABS(L2-L3),ABS(L5-L6)).

If all the conventional restrictions are satisfied, then these restrictions are met. Conversely, if input to this subroutine meets all of these restrictions and the conventional restriction stated above, then all the conventional restrictions are satisfied.

The user should be cautious in using input parameters that do not satisfy the conventional restrictions. For example, the the subroutine produces values of

```
h(L1) = { L1 2/3  1 }
        {2/3 2/3 2/3}
```

for L1=1/3 and 4/3 but none of the symmetry properties of the 6j symbol, set forth on pages 1063 and 1064 of Messiah, is satisfied.

The subroutine generates h(L1MIN), h(L1MIN+1), ..., h(L1MAX) where L1MIN and L1MAX are defined above. The sequence h(L1) is generated by a three-term recurrence algorithm with scaling to control overflow. Both backward and forward recurrence are used to maintain numerical stability. The two recurrence sequences are matched at an interior point and are normalized from the unitary property of 6j coefficients and Wigner's phase convention.

The algorithm is suited to applications in which large quantum numbers arise, such as in molecular dynamics.

***REFERENCES

1. Messiah, Albert., Quantum Mechanics, Volume II, North-Holland Publishing Company, 1963.
2. Schulten, Klaus and Gordon, Roy G., Exact recursive evaluation of 3j and 6j coefficients for quantum-mechanical coupling of angular momenta, J Math Phys, v 16, no. 10, October 1975, pp. 1961-1970.
3. Schulten, Klaus and Gordon, Roy G., Semiclassical approximations to 3j and 6j coefficients for quantum-mechanical coupling of angular momenta, J Math Phys, v 16, no. 10, October 1975, pp. 1971-1988.
4. Schulten, Klaus and Gordon, Roy G., Recursive evaluation of 3j and 6j coefficients, Computer Phys Comm, v 11, 1976, pp. 269-278.

***ROUTINES CALLED D1MACH, XERMSG

***REVISION HISTORY (YYMMDD)

750101 DATE WRITTEN

880515 SLATEC prologue added by G. C. Nielson, NBS; parameters HUGE and TINY revised to depend on D1MACH.

891229 Prologue description rewritten; other prologue sections revised; LMATCH (location of match point for recurrences) removed from argument list; argument IER changed to serve only as an error flag (previously, in cases without error, it returned the number of scalings); number of error codes

```
        increased to provide more precise error information;
        program comments revised; SLATEC error handler calls
        introduced to enable printing of error messages to meet
        SLATEC standards. These changes were done by D. W. Lozier,
        M. A. McClain and J. M. Smith of the National Institute
        of Standards and Technology, formerly NBS.
910415  Mixed type expressions eliminated; variable C1 initialized;
        description of SIXCOF expanded. These changes were done by
        D. W. Lozier.
END PROLOGUE
```

DRD

```
      DOUBLE PRECISION FUNCTION DRD (X, Y, Z, IER)
***BEGIN PROLOGUE  DRD
***PURPOSE  Compute the incomplete or complete elliptic integral of
            the 2nd kind. For X and Y nonnegative, X+Y and Z positive,
            DRD(X,Y,Z) = Integral from zero to infinity of
                                    -1/2     -1/2     -3/2
                      (3/2)(t+X)    (t+Y)    (t+Z)    dt.
            If X or Y is zero, the integral is complete.
***LIBRARY   SLATEC
***CATEGORY  C14
***TYPE      DOUBLE PRECISION (RD-S, DRD-D)
***KEYWORDS  COMPLETE ELLIPTIC INTEGRAL, DUPLICATION THEOREM,
             INCOMPLETE ELLIPTIC INTEGRAL, INTEGRAL OF THE SECOND 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.     DRD
          Evaluate an INCOMPLETE (or COMPLETE) ELLIPTIC INTEGRAL
          of the second kind
          Standard FORTRAN function routine
          Double precision version
          The routine calculates an approximation result to
          DRD(X,Y,Z) = Integral from zero to infinity of
                                    -1/2     -1/2     -3/2
                      (3/2)(t+X)    (t+Y)    (t+Z)    dt,
          where X and Y are nonnegative, X + Y is positive, and Z is
          positive.  If X or Y 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

          DRD( X, Y, Z, IER )

          Parameters On Entry
          Values assigned by the calling routine

          X      - Double precision, nonnegative variable

          Y      - Double precision, nonnegative variable

                   X + Y is positive

          Z      - Double precision, positive variable
```

On Return (values assigned by the DRD routine)

DRD - Double 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 DRD routine

Value assigned	Error message printed
IER = 1	MIN(X,Y) .LT. 0.0D0
= 2	MIN(X + 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 2 / (machine maximum) ** (2/3).

UPLIM - Upper limit of valid arguments

Not greater than (0.1D0 * ERRTOL / machine minimum) ** (2/3), where ERRTOL is described below. In the following table it is assumed that ERRTOL will never be chosen smaller than 1.0D-5.

Acceptable values for:	LOLIM	UPLIM
IBM 360/370 SERIES :	6.0D-51	1.0D+48
CDC 6000/7000 SERIES :	5.0D-215	2.0D+191
UNIVAC 1100 SERIES :	1.0D-205	2.0D+201
CRAY :	3.0D-1644	1.69D+1640
VAX 11 SERIES :	1.0D-25	4.5D+21

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
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
1.0D-3	4.0D-18
3.0D-3	3.0D-15
1.0D-2	4.0D-12
3.0D-2	3.0D-9
1.0D-1	4.0D-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:

DRD Special Comments

Check: DRD(X,Y,Z) + DRD(Y,Z,X) + DRD(Z,X,Y)
= 3 / SQRT(X * Y * Z), where X, Y, and Z are positive.

On Input:

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

On Output:

X, Y, Z are unaltered.

****

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

Special double precision functions via DRD and DRF

Legendre form of ELLIPTIC INTEGRAL of 2nd kind

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

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

$$E(K) = DRF(0,1-K^2,1) - (K^2/3)\ DRD(0,1-K^2,1)$$

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

Bulirsch form of ELLIPTIC INTEGRAL of 2nd kind

$$EL2(X,KC,A,B) = AX\ DRF(1,1+KC^2 X^2,1+X^2) +$$

$$+(1/3)(B-A)\ X^3\ DRD(1,1+KC^2 X^2,1+X^2)$$

Legendre form of alternative ELLIPTIC INTEGRAL of 2nd kind

$$D(Q,K) = \int_0^Q SIN^2 P\ (1-K^2 SIN^2 P)^{-1/2}\ DP$$

$$D(Q,K) = (1/3)\ (SIN^3 Q)\ DRD(COS^2 Q,1-K^2 SIN^2 Q,1)$$

Lemniscate constant B

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

B = (1/3) DRD (0,2,1)

Heuman's LAMBDA function

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

$$= SIN(B) \, (DRF(0,COS^2(A),1)-(1/3) \, SIN^2(A) \, *$$

$$*DRD(0,COS^2(A),1)) \, DRF(COS^2(B),1-COS^2(A) \, SIN^2(B),1)$$

$$-(1/3) \, COS^2(A) \, SIN^3(B) \, DRF(0,COS^2(A),1) \, *$$

$$*DRD(COS^2(B),1-COS^2(A) \, SIN^2(B),1)$$

Jacobi ZETA function

$$Z(B,K) = (K^2/3) \, SIN(B) \, DRF(COS^2(B),1-K^2 \, SIN^2(B),1)$$

$$*DRD(0,1-K^2,1)/DRF(0,1-K^2,1)$$

$$-(K^2/3) \, SIN^3(B) \, DRD(COS^2(B),1-K^2 \, SIN^2(B),1)$$

***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  D1MACH, 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)
   900315  CALLs to XERROR changed to CALLs to XERMSG.  (THJ)
   900326  Removed duplicate information from DESCRIPTION section.
           (WRB)
   900510  Modify calls to XERMSG to put in standard form.  (RWC)
   920501  Reformatted the REFERENCES section.  (WRB)
   END PROLOGUE
```

DRF

```
      DOUBLE PRECISION FUNCTION DRF (X, Y, Z, IER)
***BEGIN PROLOGUE  DRF
***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      DOUBLE 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.     DRF
          Evaluate an INCOMPLETE (or COMPLETE) ELLIPTIC INTEGRAL
          of the first kind
          Standard FORTRAN function routine
          Double precision version
          The routine calculates an approximation result to
          DRF(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
          DRF( X, Y, Z, IER )

          Parameters On entry
          Values assigned by the calling routine

          X      - Double precision, nonnegative variable

          Y      - Double precision, nonnegative variable

          Z      - Double precision, nonnegative variable
```

On Return (values assigned by the DRF routine)

DRF - Double 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 DRF routine

Value assigned	Error Message Printed
IER = 1	MIN(X,Y,Z) .LT. 0.0D0
= 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 :	3.0D-78	1.0D+75
CDC 6000/7000 SERIES :	1.0D-292	1.0D+321
UNIVAC 1100 SERIES :	1.0D-307	1.0D+307
CRAY :	2.3D-2466	1.09D+2465
VAX 11 SERIES :	1.5D-38	3.0D+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
1.0D-3	3.0D-19
3.0D-3	2.0D-16
1.0D-2	3.0D-13
3.0D-2	2.0D-10
1.0D-1	3.0D-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:

DRF Special Comments

Check by addition theorem: DRF(X,X+Z,X+W) + DRF(Y,Y+Z,Y+W) = DRF(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 DRF(X,Y,Z).

On Output:

X, Y, Z are unaltered.

****

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

Special double precision functions via DRF

Legendre form of ELLIPTIC INTEGRAL of 1st kind

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

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

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

Bulirsch form of ELLIPTIC INTEGRAL of 1st kind

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

Lemniscate constant A

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

***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 D1MACH, 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
```

DRJ

```
     DOUBLE PRECISION FUNCTION DRJ (X, Y, Z, P, IER)
***BEGIN PROLOGUE  DRJ
***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      DOUBLE 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.     DRJ
          Standard FORTRAN function routine
          Double precision version
          The routine calculates an approximation result to
          DRJ(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
          DRJ( X, Y, Z, P, IER )

          Parameters on Entry
          Values assigned by the calling routine

          X      - Double precision, nonnegative variable

          Y      - Double precision, nonnegative variable

          Z      - Double precision, nonnegative variable

          P      - Double precision, positive variable
```

On Return (values assigned by the DRJ routine)

DRJ - Double 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 DRJ routine

Value assigned	Error Message printed
IER = 1	MIN(X,Y,Z) .LT. 0.0D0
= 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 DRC.

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

Acceptable values for:	LOLIM	UPLIM
IBM 360/370 SERIES :	2.0D-26	3.0D+24
CDC 6000/7000 SERIES :	5.0D-98	3.0D+106
UNIVAC 1100 SERIES :	5.0D-103	6.0D+101
CRAY :	1.32D-822	1.4D+821
VAX 11 SERIES :	2.5D-13	9.0D+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 DRJ 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
1.0D-3	4.0D-18
3.0D-3	3.0D-15
1.0D-2	4.0D-12
3.0D-2	3.0D-9
1.0D-1	4.0D-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:

DRJ Special Comments

Check by addition theorem: DRJ(X,X+Z,X+W,X+P) + DRJ(Y,Y+Z,Y+W,Y+P) + (A-B) * DRJ(A,B,B,A) + 3.0D0 / SQRT(A) = DRJ(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 3.0D0 * DRC(A,B).

On Input:

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

On Output:

X, Y, Z, P are unaltered.

****

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

Special double precision functions via DRJ and DRF

Legendre form of ELLIPTIC INTEGRAL of 3rd kind

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

$$*(1-K^2\,SIN^2(THETA))^{-1/2}\ D\,THETA$$

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

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

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

Bulirsch form of ELLIPTIC INTEGRAL of 3rd kind

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

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

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

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

Heuman's LAMBDA function

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

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

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

```
                                  2         2       2
                          *DRJ(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
               *DRF(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
               *DRJ(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
                      *DRJ(0,1-K ,1,1-K SIN (B)) / DRF (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  D1MACH, DRC, 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
```

DROT

```
      SUBROUTINE DROT (N, DX, INCX, DY, INCY, DC, DS)
***BEGIN PROLOGUE  DROT
***PURPOSE  Apply a plane Givens rotation.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A8
***TYPE      DOUBLE 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)
       DX  double precision vector with N elements
     INCX  storage spacing between elements of DX
       DY  double precision vector with N elements
     INCY  storage spacing between elements of DY
       DC  D.P. element of rotation matrix
       DS  D.P. element of rotation matrix

     --Output--
       DX  rotated vector DX (unchanged if N .LE. 0)
       DY  rotated vector DY (unchanged if N .LE. 0)

     Multiply the 2 x 2 matrix  ( DC DS) times the 2 x N matrix (DX**T)
                                (-DS DC)                        (DY**T)
     where **T indicates transpose.  The elements of DX are in
     DX(LX+I*INCX), I = 0 to N-1, where LX = 1 if INCX .GE. 0, else
     LX = 1+(1-N)*INCX, and similarly for DY 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
```

DROTG

```
      SUBROUTINE DROTG (DA, DB, DC, DS)
***BEGIN PROLOGUE  DROTG
***PURPOSE  Construct a plane Givens rotation.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B10
***TYPE      DOUBLE 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--
       DA  double precision scalar
       DB  double precision scalar

     --Output--
       DA  double precision result R
       DB  double precision result Z
       DC  double precision result
       DS  double precision result

     Construct the Givens transformation

         ( DC  DS )
     G = (        ) ,    DC**2 + DS**2 = 1 ,
         (-DS  DC )

     which zeros the second entry of the 2-vector  (DA,DB)**T .

     The quantity R = (+/-)SQRT(DA**2 + DB**2) overwrites DA in
     storage.  The value of DB is overwritten by a value Z which
     allows DC and DS to be recovered by the following algorithm.

           If Z=1  set  DC=0.0  and  DS=1.0
           If ABS(Z) .LT. 1  set  DC=SQRT(1-Z**2)  and  DS=Z
           If ABS(Z) .GT. 1  set  DC=1/Z  and  DS=SQRT(1-DC**2)

     Normally, the subprogram DROT(N,DX,INCX,DY,INCY,DC,DS) 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
   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
```

DROTM

```
      SUBROUTINE DROTM (N, DX, INCX, DY, INCY, DPARAM)
***BEGIN PROLOGUE  DROTM
***PURPOSE  Apply a modified Givens transformation.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1A8
***TYPE      DOUBLE 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)
       DX  double precision vector with N elements
     INCX  storage spacing between elements of DX
       DY  double precision vector with N elements
     INCY  storage spacing between elements of DY
   DPARAM  5-element D.P. vector.  DPARAM(1) is DFLAG described below.
           Locations 2-5 of SPARAM contain elements of the
           transformation matrix H described below.

     --Output--
       DX  rotated vector (unchanged if N .LE. 0)
       DY  rotated vector (unchanged if N .LE. 0)

     Apply the modified Givens transformation, H, to the 2 by N matrix
     (DX**T)
     (DY**T) , where **T indicates transpose.  The elements of DX are
     in DX(LX+I*INCX), I = 0 to N-1, where LX = 1 if INCX .GE. 0, else
     LX = 1+(1-N)*INCX, and similarly for DY using LY and INCY.

     With DPARAM(1)=DFLAG, H has one of the following forms:

     DFLAG=-1.D0     DFLAG=0.D0        DFLAG=1.D0     DFLAG=-2.D0

       (DH11  DH12)    (1.D0  DH12)    (DH11  1.D0)    (1.D0  0.D0)
     H=(          )    (          )    (          )    (         )
       (DH21  DH22),   (DH21  1.D0),   (-1.D0 DH22),   (0.D0  1.D0).

     See DROTMG for a description of data storage in DPARAM.

***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
```

DROTMG

```
      SUBROUTINE DROTMG (DD1, DD2, DX1, DY1, DPARAM)
***BEGIN PROLOGUE  DROTMG
***PURPOSE  Construct a modified Givens transformation.
***LIBRARY   SLATEC (BLAS)
***CATEGORY  D1B10
***TYPE      DOUBLE 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--
       DD1  double precision scalar
       DD2  double precision scalar
       DX1  double precision scalar
       DX2  double precision scalar
   DPARAM  D.P. 5-vector. DPARAM(1)=DFLAG defined below.
           Locations 2-5 contain the rotation matrix.

      --Output--
       DD1  changed to represent the effect of the transformation
       DD2  changed to represent the effect of the transformation
       DX1  changed to represent the effect of the transformation
       DX2  unchanged

      Construct the modified Givens transformation matrix H which zeros
      the second component of the 2-vector  (SQRT(DD1)*DX1,SQRT(DD2)*
      DY2)**T.
      With DPARAM(1)=DFLAG, H has one of the following forms:

      DFLAG=-1.D0     DFLAG=0.D0        DFLAG=1.D0     DFLAG=-2.D0

        (DH11  DH12)    (1.D0  DH12)    (DH11  1.D0)    (1.D0  0.D0)
      H=(          )    (          )    (          )    (          )
        (DH21  DH22),   (DH21  1.D0),   (-1.D0 DH22),   (0.D0  1.D0).

      Locations 2-5 of DPARAM contain DH11, DH21, DH12, and DH22,
      respectively.  (Values of 1.D0, -1.D0, or 0.D0 implied by the
      value of DPARAM(1) are not stored in DPARAM.)

***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
   890531  Changed all specific intrinsics to generic.  (WRB)
   890531  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
```

DS2LT

```
      SUBROUTINE DS2LT (N, NELT, IA, JA, A, ISYM, NEL, IEL, JEL, EL)
***BEGIN PROLOGUE  DS2LT
***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      DOUBLE 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)
     DOUBLE PRECISION A(NELT), EL(NEL)

     CALL DS2LT( 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       Double Precision 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      Double Precision     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
       double precision 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)
   890404  DATE WRITTEN
   890404  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
```

DS2Y

```
      SUBROUTINE DS2Y (N, NELT, IA, JA, A, ISYM)
***BEGIN PROLOGUE  DS2Y
***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      DOUBLE 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
     DOUBLE PRECISION A(NELT)

     CALL DS2Y( 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    Double Precision 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 double precision 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  QS2I1D
***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
```

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 SLATEC3.
availability	Machines on which these routines run.
who	Who to contact for assistance.
introduction	Brief overview of SLATEC3; and other SLATEC documentation.
index	This structural keyword index.
date	The latest revision date for SLATEC3.
revisions	Revision history of this document.

In addition, the name of every subroutine described in SLATEC3 is the keyword and link for retrieving its description. Included are:

Routine Name	Gams Cat.	Function Performed
DACOSH	c	elementary-functions, special-functions
DAI	c	elementary-functions, special-functions
DAIE	c	elementary-functions, special-functions
DASINH	c	elementary-functions, special-functions
DASUM	d1a	vector-operations
DATANH	c	elementary-functions, special-functions
DAVINT	h2	quadrature, definite-integrals
DAWS	c	elementary-functions, special-functions
DAXPY	d1a	vector-operations
DBCG	d2	linear-equations
DBESI	c	elementary-functions, special-functions
DBESI0	c	elementary-functions, special-functions
DBESI1	c	elementary-functions, special-functions
DBESJ	c	elementary-functions, special-functions
DBESJ0	c	elementary-functions, special-functions
DBESJ1	c	elementary-functions, special-functions
DBESK	c	elementary-functions, special-functions
DBESK0	c	elementary-functions, special-functions
DBESK1	c	elementary-functions, special-functions
DBESKS	c	elementary-functions, special-functions
DBESY	c	elementary-functions, special-functions
DBESY0	c	elementary-functions, special-functions
DBESY1	c	elementary-functions, special-functions
DBETA	c	elementary-functions, special-functions
DBETAI	c	elementary-functions, special-functions
DBFQAD	e	interpolation
DBHIN	n	data-handling
DBHIN	n	data-handling
DBI	c	elementary-functions, special-functions
DBIE	c	elementary-functions, special-functions
DBINOM	c	elementary-functions, special-functions

DBINT4	e	interpolation
DBINTK	e	interpolation
DBNDAC	d9	overdetermined-systems, least-squares
DBNDSL	d9	overdetermined-systems, least-squares
DBOCLS	k	approximation
DBOLS	k	approximation
DBSI0E	c	elementary-functions, special-functions
DBSI1E	c	elementary-functions, special-functions
DBSK0E	c	elementary-functions, special-functions
DBSK1E	c	elementary-functions, special-functions
DBSKES	c	elementary-functions, special-functions
DBSKIN	c	elementary-functions, special-functions
DBSPDR	e	interpolation
DBSPEV	e	interpolation
DBSPPP	e	interpolation
DBSPVD	e	interpolation
DBSPVN	e	interpolation
DBSQAD	e	interpolation
DBVALU	e	interpolation
DBVSUP	i1	ordinary-differential-equations
DCBRT	c	elementary-functions, special-functions
DCDOT	d1a	vector-operations
DCG	d2	linear-equations
DCGN	d2	linear-equations
DCGS	d2	linear-equations
DCHDC	linpack	cholesky-operations
DCHDD	linpack	cholesky-operations
DCHEX	linpack	cholesky-operations
DCHFDV	e	interpolation
DCHFEV	e	interpolation
DCHU	c	elementary-functions, special-functions
DCHUD	linpack	cholesky-operations
DCKDER	f	nonlinear-equations
DCOPY	d1a	vector-operations
DCOPYM	d1a	vector-operations
DCOSDG	c	elementary-functions, special-functions
DCOT	c	elementary-functions, special-functions
DCOV	k	approximation
DCPPLT	n	data-handling
DCPPLT	n	data-handling
DCSEVL	c	elementary-functions, special-functions
DCV	l	statistics
DDASSL	i1	ordinary-differential-equations
DDAWS	c	elementary-functions, special-functions
DDEABM	i1	ordinary-differential-equations
DDEBDF	i1	ordinary-differential-equations
DDERKF	i1	ordinary-differential-equations
DDOT	d1a	vector-operations
DDRIV1	i1	ordinary-differential-equations
DDRIV2	i1	ordinary-differential-equations
DDRIV3	i1	ordinary-differential-equations
DE1	c	elementary-functions, special-functions
DEABM	i1	ordinary-differential-equations
DEBDF	i1	ordinary-differential-equations
DEFC	k	approximation
DEI	c	elementary-functions, special-functions
DERF	c	elementary-functions, special-functions
DERFC	c	elementary-functions, special-functions
DERKF	i1	ordinary-differential-equations
DEXINT	c	elementary-functions, special-functions

DEXPRL	c	elementary-functions, special-functions
DFAC	c	elementary-functions, special-functions
DFC	k	approximation
DFZERO	f	nonlinear-equations
DGAMI	c	elementary-functions, special-functions
DGAMIC	c	elementary-functions, special-functions
DGAMIT	c	elementary-functions, special-functions
DGAMLM	c	elementary-functions, special-functions
DGAMMA	c	elementary-functions, special-functions
DGAMR	c	elementary-functions, special-functions
DGAUS8	h2	quadrature, definite-integrals
DGBCO	linpack	general-band
DGBDI	linpack	general-band
DGBFA	linpack	general-band
DGBSL	linpack	general-band
DGECO	linpack	general
DGEDI	linpack	general
DGEFA	linpack	general
DGEFS	d2	linear-equations
DGEMM	d1b	matrix-operations
DGEMV	d1b	matrix-operations
DGER	d1b	matrix-operations
DGESL	linpack	general
DGLSS	d9	overdetermined-systems, least-squares
DGMRES	d2	linear-equations
DGTSL	linpack	general-tridiagonal
DHFTI	d9	overdetermined-systems, least-squares
DINTP	i1	ordinary-differential-equations
DINTRV	e	interpolation
DIR	d2	linear-equations
DLBETA	c	elementary-functions, special-functions
DLGAMS	c	elementary-functions, special-functions
DLI	c	elementary-functions, special-functions
DLLSIA	d9	overdetermined-systems, least-squares
DLLTI2	d2	linear-equations
DLNGAM	c	elementary-functions, special-functions
DLNREL	c	elementary-functions, special-functions
DLPDOC	d2	linear-equations
DLSEI	k	approximation
DNBCO	d2	linear-equations
DNBDI	d3	determinants
DNBFA	d2	linear-equations
DNBFS	d2	linear-equations
DNBSL	d2	linear-equations
DNLS1	k	approximation
DNLS1E	k	approximation
DNRM2	d1a	vector-operations
DNSQ	f	nonlinear-equations
DNSQE	f	nonlinear-equations
DOMN	d2	linear-equations
DP1VLU	k	approximation
DPBCO	linpack	hermitian-positive-definite-band
DPBDI	linpack	hermitian-positive-definite-band
DPBFA	linpack	hermitian-positive-definite-band
DPBSL	linpack	hermitian-positive-definite-band
DPCHBS	e	interpolation
DPCHCM	e	interpolation
DPCHFD	e	interpolation
DPCHFE	e	interpolation
DPCHIA	e	interpolation

DPCHIC	e	interpolation
DPCHID	e	interpolation
DPCHIM	e	interpolation
DPCHSP	e	interpolation
DPCOEF	k	approximation
DPFQAD	e	interpolation
DPLINT	e	interpolation
DPOCH	c	elementary-functions, special-functions
DPOCH1	c	elementary-functions, special-functions
DPOCO	linpack	hermitian-positive-definite
DPODI	linpack	hermitian-positive-definite
DPOFA	linpack	hermitian-positive-definite
DPOFS	d2	linear-equations
DPOLCF	e	interpolation
DPOLFT	k	approximation
DPOLVL	e	interpolation
DPOSL	linpack	hermitian-positive-definite
DPPCO	linpack	hermitian-positive-definite
DPPDI	linpack	hermitian-positive-definite
DPPERM	n	data-handling
DPPFA	linpack	hermitian-positive-definite
DPPQAD	e	interpolation
DPPSL	linpack	hermitian-positive-definite
DPPVAL	e	interpolation
DPSI	c	elementary-functions, special-functions
DPSIFN	c	elementary-functions, special-functions
DPSORT	n	data-handling
DPTSL	linpack	positive-definite-tridiagonal
DQAG	h2	quadrature, definite-integrals
DQAGE	h2	quadrature, definite-integrals
DQAGI	h2	quadrature, definite-integrals
DQAGIE	h2	quadrature, definite-integrals
DQAGP	h2	quadrature, definite-integrals
DQAGPE	h2	quadrature, definite-integrals
DQAGS	h2	quadrature, definite-integrals
DQAGSE	h2	quadrature, definite-integrals
DQAWC	h2	quadrature, definite-integrals
DQAWCE	h2	quadrature, definite-integrals
DQAWF	h2	quadrature, definite-integrals
DQAWFE	h2	quadrature, definite-integrals
DQAWO	h2	quadrature, definite-integrals
DQAWOE	h2	quadrature, definite-integrals
DQAWS	h2	quadrature, definite-integrals
DQAWSE	h2	quadrature, definite-integrals
DQC25C	h2	quadrature, definite-integrals
DQC25F	h2	quadrature, definite-integrals
DQC25S	h2	quadrature, definite-integrals
DQDOTA	d1a	vector-operations
DQDOTI	d1a	vector-operations
DQK15	h2	quadrature, definite-integrals
DQK15I	h2	quadrature, definite-integrals
DQK15W	h2	quadrature, definite-integrals
DQK21	h2	quadrature, definite-integrals
DQK31	h2	quadrature, definite-integrals
DQK41	h2	quadrature, definite-integrals
DQK51	h2	quadrature, definite-integrals
DQK61	h2	quadrature, definite-integrals
DQMOMO	h2	quadrature, definite-integrals
DQNC79	h2	quadrature, definite-integrals
DQNG	h2	quadrature, definite-integrals

DQRDC	d5	qr-decomposition
DQRSL	d5	qr-decomposition
DRC	c	elementary-functions, special-functions
DRC3JJ	c	elementary-functions, special-functions
DRC3JM	c	elementary-functions, special-functions
DRC6J	c	elementary-functions, special-functions
DRD	c	elementary-functions, special-functions
DRF	c	elementary-functions, special-functions
DRJ	c	elementary-functions, special-functions
DROT	d1a	vector-operations
DROTG	d1a	vector-operations
DROTM	d1a	vector-operations
DROTMG	d1a	vector-operations
DS2LT	d2	linear-equations
DS2Y	d1b	matrix-operations

Date and Revisions

Revision date	Keyword affected	Description of changes
03Apr96	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 (03Apr96)

UCID-19631,19632,19633

Privacy and Legal Notice (URL: http://www.llnl.gov/disclaimer.html)

TRG (03Apr96) Contact on the OCF: lc-hotline@llnl.gov, on the SCF: lc-hotline@pop.llnl.gov