```
*DECK DLSODES
      SUBROUTINE DLSODES (F, NEQ, Y, T, TOUT, ITOL, RTOL, ATOL, ITASK,
     1            ISTATE, IOPT, RWORK, LRW, IWORK, LIW, JAC, MF)
      EXTERNAL F, JAC
      INTEGER NEQ, ITOL, ITASK, ISTATE, IOPT, LRW, IWORK, LIW, MF
      DOUBLE PRECISION Y, T, TOUT, RTOL, ATOL, RWORK
      DIMENSION NEQ(*), Y(*), RTOL(*), ATOL(*), RWORK(LRW), IWORK(LIW)
C-----------------------------------------------------------------------
C This is the 12 November 2003 version of
C DLSODES: Livermore Solver for Ordinary Differential Equations
C          with general Sparse Jacobian matrix.
C
C This version is in double precision.
C
C DLSODES solves the initial value problem for stiff or nonstiff
C systems of first order ODEs,
C     dy/dt = f(t,y) ,  or, in component form,
C     dy(i)/dt = f(i) = f(i,t,y(1),y(2),...,y(NEQ)) (i = 1,...,NEQ).
C DLSODES is a variant of the DLSODE package, and is intended for
C problems in which the Jacobian matrix df/dy has an arbitrary
C sparse structure (when the problem is stiff).
C
C Authors:       Alan C. Hindmarsh
C                Center for Applied Scientific Computing, L-561
C                Lawrence Livermore National Laboratory
C                Livermore, CA 94551
C and
C                Andrew H. Sherman
C                J. S. Nolen and Associates
C                Houston, TX 77084
C-----------------------------------------------------------------------
C References:
C 1.  Alan C. Hindmarsh,  ODEPACK, A Systematized Collection of ODE
C     Solvers, in Scientific Computing, R. S. Stepleman et al. (Eds.),
C     North-Holland, Amsterdam, 1983, pp. 55-64.
C
C 2.  S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman,
C     Yale Sparse Matrix Package: I. The Symmetric Codes,
C     Int. J. Num. Meth. Eng., 18 (1982), pp. 1145-1151.
C
C 3.  S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman,
C     Yale Sparse Matrix Package: II. The Nonsymmetric Codes,
C     Research Report No. 114, Dept. of Computer Sciences, Yale
C     University, 1977.
C-----------------------------------------------------------------------
C Summary of Usage.
C
C Communication between the user and the DLSODES package, for normal
C situations, is summarized here.  This summary describes only a subset
C of the full set of options available.  See the full description for
C details, including optional communication, nonstandard options,
C and instructions for special situations.  See also the example
C problem (with program and output) following this summary.
C
C A. First provide a subroutine of the form:
C               SUBROUTINE F (NEQ, T, Y, YDOT)
C               DOUBLE PRECISION T, Y(*), YDOT(*)
C which supplies the vector function f by loading YDOT(i) with f(i).
C
C B. Next determine (or guess) whether or not the problem is stiff.
C Stiffness occurs when the Jacobian matrix df/dy has an eigenvalue
C whose real part is negative and large in magnitude, compared to the
C reciprocal of the t span of interest.  If the problem is nonstiff,
C use a method flag MF = 10.  If it is stiff, there are two standard
```

```
C choices for the method flag, MF = 121 and MF = 222.  In both cases,
C DLSODES requires the Jacobian matrix in some form, and it treats this
C matrix in general sparse form, with sparsity structure determined
C internally.  (For options where the user supplies the sparsity
C structure, see the full description of MF below.)
C
C C. If the problem is stiff, you are encouraged to supply the Jacobian
C directly (MF = 121), but if this is not feasible, DLSODES will
C compute it internally by difference quotients (MF = 222).
C If you are supplying the Jacobian, provide a subroutine of the form:
C               SUBROUTINE JAC (NEQ, T, Y, J, IAN, JAN, PDJ)
C               DOUBLE PRECISION T, Y(*), IAN(*), JAN(*), PDJ(*)
C Here NEQ, T, Y, and J are input arguments, and the JAC routine is to
C load the array PDJ (of length NEQ) with the J-th column of df/dy.
C I.e., load PDJ(i) with df(i)/dy(J) for all relevant values of i.
C The arguments IAN and JAN should be ignored for normal situations.
C DLSODES will call the JAC routine with J = 1,2,...,NEQ.
C Only nonzero elements need be loaded.  Usually, a crude approximation
C to df/dy, possibly with fewer nonzero elements, will suffice.
C
C D. Write a main program which calls Subroutine DLSODES once for
C each point at which answers are desired.  This should also provide
C for possible use of logical unit 6 for output of error messages by
C DLSODES.  On the first call to DLSODES, supply arguments as follows:
C F      = name of subroutine for right-hand side vector f.
C          This name must be declared External in calling program.
C NEQ    = number of first order ODEs.
C Y      = array of initial values, of length NEQ.
C T      = the initial value of the independent variable t.
C TOUT   = first point where output is desired (.ne. T).
C ITOL   = 1 or 2 according as ATOL (below) is a scalar or array.
C RTOL   = relative tolerance parameter (scalar).
C ATOL   = absolute tolerance parameter (scalar or array).
C          The estimated local error in Y(i) will be controlled so as
C          to be roughly less (in magnitude) than
C             EWT(i) = RTOL*ABS(Y(i)) + ATOL     if ITOL = 1, or
C             EWT(i) = RTOL*ABS(Y(i)) + ATOL(i)  if ITOL = 2.
C          Thus the local error test passes if, in each component,
C          either the absolute error is less than ATOL (or ATOL(i)),
C          or the relative error is less than RTOL.
C          Use RTOL = 0.0 for pure absolute error control, and
C          use ATOL = 0.0 (or ATOL(i) = 0.0) for pure relative error
C          control.  Caution: actual (global) errors may exceed these
C          local tolerances, so choose them conservatively.
C ITASK  = 1 for normal computation of output values of Y at t = TOUT.
C ISTATE = integer flag (input and output).  Set ISTATE = 1.
C IOPT   = 0 to indicate no optional inputs used.
C RWORK  = real work array of length at least:
C             20 + 16*NEQ           for MF = 10,
C             20 + (2 + 1./LENRAT)*NNZ + (11 + 9./LENRAT)*NEQ
C                                   for MF = 121 or 222,
C          where:
C          NNZ    = the number of nonzero elements in the sparse
C                   Jacobian (if this is unknown, use an estimate), and
C          LENRAT = the real to integer wordlength ratio (usually 1 in
C                   single precision and 2 in double precision).
C          In any case, the required size of RWORK cannot generally
C          be predicted in advance if MF = 121 or 222, and the value
C          above is a rough estimate of a crude lower bound.  Some
C          experimentation with this size may be necessary.
C          (When known, the correct required length is an optional
C          output, available in IWORK(17).)
C LRW    = declared length of RWORK (in user dimension).
C IWORK  = integer work array of length at least 30.
```

```
C LIW    = declared length of IWORK (in user dimension).
C JAC    = name of subroutine for Jacobian matrix (MF = 121).
C          If used, this name must be declared External in calling
C          program.  If not used, pass a dummy name.
C MF     = method flag.  Standard values are:
C          10  for nonstiff (Adams) method, no Jacobian used
C          121 for stiff (BDF) method, user-supplied sparse Jacobian
C          222 for stiff method, internally generated sparse Jacobian
C Note that the main program must declare arrays Y, RWORK, IWORK,
C and possibly ATOL.
C
C E. The output from the first call (or any call) is:
C      Y = array of computed values of y(t) vector.
C      T = corresponding value of independent variable (normally TOUT).
C ISTATE = 2  if DLSODES was successful, negative otherwise.
C          -1 means excess work done on this call (perhaps wrong MF).
C          -2 means excess accuracy requested (tolerances too small).
C          -3 means illegal input detected (see printed message).
C          -4 means repeated error test failures (check all inputs).
C          -5 means repeated convergence failures (perhaps bad Jacobian
C             supplied or wrong choice of MF or tolerances).
C          -6 means error weight became zero during problem. (Solution
C             component i vanished, and ATOL or ATOL(i) = 0.)
C          -7 means a fatal error return flag came from sparse solver
C             CDRV by way of DPRJS or DSOLSS.  Should never happen.
C          A return with ISTATE = -1, -4, or -5 may result from using
C          an inappropriate sparsity structure, one that is quite
C          different from the initial structure.  Consider calling
C          DLSODES again with ISTATE = 3 to force the structure to be
C          reevaluated.  See the full description of ISTATE below.
C
C F. To continue the integration after a successful return, simply
C reset TOUT and call DLSODES again.  No other parameters need be reset.
C
C-----------------------------------------------------------------------
C Example Problem.
C
C The following is a simple example problem, with the coding
C needed for its solution by DLSODES.  The problem is from chemical
C kinetics, and consists of the following 12 rate equations:
C    dy1/dt  = -rk1*y1
C    dy2/dt  = rk1*y1 + rk11*rk14*y4 + rk19*rk14*y5
C                - rk3*y2*y3 - rk15*y2*y12 - rk2*y2
C    dy3/dt  = rk2*y2 - rk5*y3 - rk3*y2*y3 - rk7*y10*y3
C                + rk11*rk14*y4 + rk12*rk14*y6
C    dy4/dt  = rk3*y2*y3 - rk11*rk14*y4 - rk4*y4
C    dy5/dt  = rk15*y2*y12 - rk19*rk14*y5 - rk16*y5
C    dy6/dt  = rk7*y10*y3 - rk12*rk14*y6 - rk8*y6
C    dy7/dt  = rk17*y10*y12 - rk20*rk14*y7 - rk18*y7
C    dy8/dt  = rk9*y10 - rk13*rk14*y8 - rk10*y8
C    dy9/dt  = rk4*y4 + rk16*y5 + rk8*y6 + rk18*y7
C    dy10/dt = rk5*y3 + rk12*rk14*y6 + rk20*rk14*y7
C                + rk13*rk14*y8 - rk7*y10*y3 - rk17*y10*y12
C                - rk6*y10 - rk9*y10
C    dy11/dt = rk10*y8
C    dy12/dt = rk6*y10 + rk19*rk14*y5 + rk20*rk14*y7
C                - rk15*y2*y12 - rk17*y10*y12
C
C with rk1 = rk5 = 0.1,  rk4 = rk8 = rk16 = rk18 = 2.5,
C      rk10 = 5.0,  rk2 = rk6 = 10.0,  rk14 = 30.0,
C      rk3 = rk7 = rk9 = rk11 = rk12 = rk13 = rk19 = rk20 = 50.0,
C      rk15 = rk17 = 100.0.
C
C The t interval is from 0 to 1000, and the initial conditions
```

```
C are y1 = 1, y2 = y3 = ... = y12 = 0.  The problem is stiff.
C
C The following coding solves this problem with DLSODES, using MF = 121
C and printing results at t = .1, 1., 10., 100., 1000.  It uses
C ITOL = 1 and mixed relative/absolute tolerance controls.
C During the run and at the end, statistical quantities of interest
C are printed (see optional outputs in the full description below).
C
C     EXTERNAL FEX, JEX
C     DOUBLE PRECISION ATOL, RTOL, RWORK, T, TOUT, Y
C     DIMENSION Y(12), RWORK(500), IWORK(30)
C     DATA LRW/500/, LIW/30/
C     NEQ = 12
C     DO 10 I = 1,NEQ
C 10    Y(I) = 0.0D0
C     Y(1) = 1.0D0
C     T = 0.0D0
C     TOUT = 0.1D0
C     ITOL = 1
C     RTOL = 1.0D-4
C     ATOL = 1.0D-6
C     ITASK = 1
C     ISTATE = 1
C     IOPT = 0
C     MF = 121
C     DO 40 IOUT = 1,5
C       CALL DLSODES (FEX, NEQ, Y, T, TOUT, ITOL, RTOL, ATOL,
C    1     ITASK, ISTATE, IOPT, RWORK, LRW, IWORK, LIW, JEX, MF)
C       WRITE(6,30)T,IWORK(11),RWORK(11),(Y(I),I=1,NEQ)
C 30    FORMAT(//' At t =',D11.3,4X,
C    1    ' No. steps =',I5,4X,' Last step =',D11.3/
C    2    '  Y array =  ',4D14.5/13X,4D14.5/13X,4D14.5)
C       IF (ISTATE .LT. 0) GO TO 80
C       TOUT = TOUT*10.0D0
C 40    CONTINUE
C     LENRW = IWORK(17)
C     LENIW = IWORK(18)
C     NST = IWORK(11)
C     NFE = IWORK(12)
C     NJE = IWORK(13)
C     NLU = IWORK(21)
C     NNZ = IWORK(19)
C     NNZLU = IWORK(25) + IWORK(26) + NEQ
C     WRITE (6,70) LENRW,LENIW,NST,NFE,NJE,NLU,NNZ,NNZLU
C 70  FORMAT(//' Required RWORK size =',I4,'   IWORK size =',I4/
C    1   ' No. steps =',I4,'   No. f-s =',I4,'   No. J-s =',I4,
C    2   '   No. LU-s =',I4/' No. of nonzeros in J =',I5,
C    3   '   No. of nonzeros in LU =',I5)
C     STOP
C 80  WRITE(6,90)ISTATE
C 90  FORMAT(///' Error halt.. ISTATE =',I3)
C     STOP
C     END
C
C     SUBROUTINE FEX (NEQ, T, Y, YDOT)
C     DOUBLE PRECISION T, Y, YDOT
C     DOUBLE PRECISION RK1, RK2, RK3, RK4, RK5, RK6, RK7, RK8, RK9,
C    1   RK10, RK11, RK12, RK13, RK14, RK15, RK16, RK17
C     DIMENSION Y(12), YDOT(12)
C     DATA RK1/0.1D0/, RK2/10.0D0/, RK3/50.0D0/, RK4/2.5D0/, RK5/0.1D0/,
C    1   RK6/10.0D0/, RK7/50.0D0/, RK8/2.5D0/, RK9/50.0D0/, RK10/5.0D0/,
C    2   RK11/50.0D0/, RK12/50.0D0/, RK13/50.0D0/, RK14/30.0D0/,
C    3   RK15/100.0D0/, RK16/2.5D0/, RK17/100.0D0/, RK18/2.5D0/,
C    4   RK19/50.0D0/, RK20/50.0D0/
```

```
C      YDOT(1)  = -RK1*Y(1)
C      YDOT(2)  = RK1*Y(1) + RK11*RK14*Y(4) + RK19*RK14*Y(5)
C     1           - RK3*Y(2)*Y(3) - RK15*Y(2)*Y(12) - RK2*Y(2)
C      YDOT(3)  = RK2*Y(2) - RK5*Y(3) - RK3*Y(2)*Y(3) - RK7*Y(10)*Y(3)
C     1           + RK11*RK14*Y(4) + RK12*RK14*Y(6)
C      YDOT(4)  = RK3*Y(2)*Y(3) - RK11*RK14*Y(4) - RK4*Y(4)
C      YDOT(5)  = RK15*Y(2)*Y(12) - RK19*RK14*Y(5) - RK16*Y(5)
C      YDOT(6)  = RK7*Y(10)*Y(3) - RK12*RK14*Y(6) - RK8*Y(6)
C      YDOT(7)  = RK17*Y(10)*Y(12) - RK20*RK14*Y(7) - RK18*Y(7)
C      YDOT(8)  = RK9*Y(10) - RK13*RK14*Y(8) - RK10*Y(8)
C      YDOT(9)  = RK4*Y(4) + RK16*Y(5) + RK8*Y(6) + RK18*Y(7)
C      YDOT(10) = RK5*Y(3) + RK12*RK14*Y(6) + RK20*RK14*Y(7)
C     1           + RK13*RK14*Y(8) - RK7*Y(10)*Y(3) - RK17*Y(10)*Y(12)
C     2           - RK6*Y(10) - RK9*Y(10)
C      YDOT(11) = RK10*Y(8)
C      YDOT(12) = RK6*Y(10) + RK19*RK14*Y(5) + RK20*RK14*Y(7)
C     1           - RK15*Y(2)*Y(12) - RK17*Y(10)*Y(12)
C      RETURN
C      END
C
C      SUBROUTINE JEX (NEQ, T, Y, J, IA, JA, PDJ)
C      DOUBLE PRECISION T, Y, PDJ
C      DOUBLE PRECISION RK1, RK2, RK3, RK4, RK5, RK6, RK7, RK8, RK9,
C     1   RK10, RK11, RK12, RK13, RK14, RK15, RK16, RK17
C      DIMENSION Y(12), IA(*), JA(*), PDJ(12)
C      DATA RK1/0.1D0/, RK2/10.0D0/, RK3/50.0D0/, RK4/2.5D0/, RK5/0.1D0/,
C     1   RK6/10.0D0/, RK7/50.0D0/, RK8/2.5D0/, RK9/50.0D0/, RK10/5.0D0/,
C     2   RK11/50.0D0/, RK12/50.0D0/, RK13/50.0D0/, RK14/30.0D0/,
C     3   RK15/100.0D0/, RK16/2.5D0/, RK17/100.0D0/, RK18/2.5D0/,
C     4   RK19/50.0D0/, RK20/50.0D0/
C      GO TO (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12), J
C 1    PDJ(1) = -RK1
C      PDJ(2) = RK1
C      RETURN
C 2    PDJ(2) = -RK3*Y(3) - RK15*Y(12) - RK2
C      PDJ(3) = RK2 - RK3*Y(3)
C      PDJ(4) = RK3*Y(3)
C      PDJ(5) = RK15*Y(12)
C      PDJ(12) = -RK15*Y(12)
C      RETURN
C 3    PDJ(2) = -RK3*Y(2)
C      PDJ(3) = -RK5 - RK3*Y(2) - RK7*Y(10)
C      PDJ(4) = RK3*Y(2)
C      PDJ(6) = RK7*Y(10)
C      PDJ(10) = RK5 - RK7*Y(10)
C      RETURN
C 4    PDJ(2) = RK11*RK14
C      PDJ(3) = RK11*RK14
C      PDJ(4) = -RK11*RK14 - RK4
C      PDJ(9) = RK4
C      RETURN
C 5    PDJ(2) = RK19*RK14
C      PDJ(5) = -RK19*RK14 - RK16
C      PDJ(9) = RK16
C      PDJ(12) = RK19*RK14
C      RETURN
C 6    PDJ(3) = RK12*RK14
C      PDJ(6) = -RK12*RK14 - RK8
C      PDJ(9) = RK8
C      PDJ(10) = RK12*RK14
C      RETURN
C 7    PDJ(7) = -RK20*RK14 - RK18
C      PDJ(9) = RK18
C      PDJ(10) = RK20*RK14
```

```
C     PDJ(12) = RK20*RK14
C     RETURN
C 8   PDJ(8) = -RK13*RK14 - RK10
C     PDJ(10) = RK13*RK14
C     PDJ(11) = RK10
C 9   RETURN
C 10  PDJ(3) = -RK7*Y(3)
C     PDJ(6) = RK7*Y(3)
C     PDJ(7) = RK17*Y(12)
C     PDJ(8) = RK9
C     PDJ(10) = -RK7*Y(3) - RK17*Y(12) - RK6 - RK9
C     PDJ(12) = RK6 - RK17*Y(12)
C 11  RETURN
C 12  PDJ(2) = -RK15*Y(2)
C     PDJ(5) = RK15*Y(2)
C     PDJ(7) = RK17*Y(10)
C     PDJ(10) = -RK17*Y(10)
C     PDJ(12) = -RK15*Y(2) - RK17*Y(10)
C     RETURN
C     END
C
C The output of this program (on a Cray-1 in single precision)
C is as follows:
C
C
C At t =  1.000e-01     No. steps =   12     Last step =  1.515e-02
C  Y array =     9.90050e-01   6.28228e-03   3.65313e-03   7.51934e-07
C                1.12167e-09   1.18458e-09   1.77291e-12   3.26476e-07
C                5.46720e-08   9.99500e-06   4.48483e-08   2.76398e-06
C
C
C At t =  1.000e+00     No. steps =   33     Last step =  7.880e-02
C  Y array =     9.04837e-01   9.13105e-03   8.20622e-02   2.49177e-05
C                1.85055e-06   1.96797e-06   1.46157e-07   2.39557e-05
C                3.26306e-05   7.21621e-04   5.06433e-05   3.05010e-03
C
C
C At t =  1.000e+01     No. steps =   48     Last step =  1.239e+00
C  Y array =     3.67876e-01   3.68958e-03   3.65133e-01   4.48325e-05
C                6.10798e-05   4.33148e-05   5.90211e-05   1.18449e-04
C                3.15235e-03   3.56531e-03   4.15520e-03   2.48741e-01
C
C
C At t =  1.000e+02     No. steps =   91     Last step =  3.764e+00
C  Y array =     4.44981e-05   4.42666e-07   4.47273e-04  -3.53257e-11
C                2.81577e-08  -9.67741e-11   2.77615e-07   1.45322e-07
C                1.56230e-02   4.37394e-06   1.60104e-02   9.52246e-01
C
C
C At t =  1.000e+03     No. steps =  111     Last step =  4.156e+02
C  Y array =    -2.65492e-13   2.60539e-14  -8.59563e-12   6.29355e-14
C               -1.78066e-13   5.71471e-13  -1.47561e-12   4.58078e-15
C                1.56314e-02   1.37878e-13   1.60184e-02   9.52719e-01
C
C
C Required RWORK size = 442   IWORK size =  30
C No. steps = 111   No. f-s = 142   No. J-s =   2   No. LU-s =  20
C No. of nonzeros in J =   44   No. of nonzeros in LU =   50
C
C-----------------------------------------------------------------------
C Full Description of User Interface to DLSODES.
C
C The user interface to DLSODES consists of the following parts.
C
```

```
C 1.   The call sequence to Subroutine DLSODES, which is a driver
C      routine for the solver.  This includes descriptions of both
C      the call sequence arguments and of user-supplied routines.
C      Following these descriptions is a description of
C      optional inputs available through the call sequence, and then
C      a description of optional outputs (in the work arrays).
C
C 2.   Descriptions of other routines in the DLSODES package that may be
C      (optionally) called by the user.  These provide the ability to
C      alter error message handling, save and restore the internal
C      Common, and obtain specified derivatives of the solution y(t).
C
C 3.   Descriptions of Common blocks to be declared in overlay
C      or similar environments, or to be saved when doing an interrupt
C      of the problem and continued solution later.
C
C 4.   Description of two routines in the DLSODES package, either of
C      which the user may replace with his/her own version, if desired.
C      These relate to the measurement of errors.
C
C-----------------------------------------------------------------------
C Part 1.  Call Sequence.
C
C The call sequence parameters used for input only are
C     F, NEQ, TOUT, ITOL, RTOL, ATOL, ITASK, IOPT, LRW, LIW, JAC, MF,
C and those used for both input and output are
C     Y, T, ISTATE.
C The work arrays RWORK and IWORK are also used for conditional and
C optional inputs and optional outputs.  (The term output here refers
C to the return from Subroutine DLSODES to the user's calling program.)
C
C The legality of input parameters will be thoroughly checked on the
C initial call for the problem, but not checked thereafter unless a
C change in input parameters is flagged by ISTATE = 3 on input.
C
C The descriptions of the call arguments are as follows.
C
C F      = the name of the user-supplied subroutine defining the
C          ODE system.  The system must be put in the first-order
C          form dy/dt = f(t,y), where f is a vector-valued function
C          of the scalar t and the vector y.  Subroutine F is to
C          compute the function f.  It is to have the form
C               SUBROUTINE F (NEQ, T, Y, YDOT)
C               DOUBLE PRECISION T, Y(*), YDOT(*)
C          where NEQ, T, and Y are input, and the array YDOT = f(t,y)
C          is output.  Y and YDOT are arrays of length NEQ.
C          Subroutine F should not alter y(1),...,y(NEQ).
C          F must be declared External in the calling program.
C
C          Subroutine F may access user-defined quantities in
C          NEQ(2),... and/or in Y(NEQ(1)+1),... if NEQ is an array
C          (dimensioned in F) and/or Y has length exceeding NEQ(1).
C          See the descriptions of NEQ and Y below.
C
C          If quantities computed in the F routine are needed
C          externally to DLSODES, an extra call to F should be made
C          for this purpose, for consistent and accurate results.
C          If only the derivative dy/dt is needed, use DINTDY instead.
C
C NEQ    = the size of the ODE system (number of first order
C          ordinary differential equations).  Used only for input.
C          NEQ may be decreased, but not increased, during the problem.
C          If NEQ is decreased (with ISTATE = 3 on input), the
C          remaining components of Y should be left undisturbed, if
```

```
C          these are to be accessed in F and/or JAC.
C
C          Normally, NEQ is a scalar, and it is generally referred to
C          as a scalar in this user interface description.  However,
C          NEQ may be an array, with NEQ(1) set to the system size.
C          (The DLSODES package accesses only NEQ(1).)  In either case,
C          this parameter is passed as the NEQ argument in all calls
C          to F and JAC.  Hence, if it is an array, locations
C          NEQ(2),... may be used to store other integer data and pass
C          it to F and/or JAC.  Subroutines F and/or JAC must include
C          NEQ in a Dimension statement in that case.
C
C Y      = a real array for the vector of dependent variables, of
C          length NEQ or more.  Used for both input and output on the
C          first call (ISTATE = 1), and only for output on other calls.
C          on the first call, Y must contain the vector of initial
C          values.  On output, Y contains the computed solution vector,
C          evaluated at T.  If desired, the Y array may be used
C          for other purposes between calls to the solver.
C
C          This array is passed as the Y argument in all calls to
C          F and JAC.  Hence its length may exceed NEQ, and locations
C          Y(NEQ+1),... may be used to store other real data and
C          pass it to F and/or JAC.  (The DLSODES package accesses only
C          Y(1),...,Y(NEQ).)
C
C T      = the independent variable.  On input, T is used only on the
C          first call, as the initial point of the integration.
C          on output, after each call, T is the value at which a
C          computed solution Y is evaluated (usually the same as TOUT).
C          On an error return, T is the farthest point reached.
C
C TOUT   = the next value of t at which a computed solution is desired.
C          Used only for input.
C
C          When starting the problem (ISTATE = 1), TOUT may be equal
C          to T for one call, then should .ne. T for the next call.
C          For the initial T, an input value of TOUT .ne. T is used
C          in order to determine the direction of the integration
C          (i.e. the algebraic sign of the step sizes) and the rough
C          scale of the problem.  Integration in either direction
C          (forward or backward in t) is permitted.
C
C          If ITASK = 2 or 5 (one-step modes), TOUT is ignored after
C          the first call (i.e. the first call with TOUT .ne. T).
C          Otherwise, TOUT is required on every call.
C
C          If ITASK = 1, 3, or 4, the values of TOUT need not be
C          monotone, but a value of TOUT which backs up is limited
C          to the current internal T interval, whose endpoints are
C          TCUR - HU and TCUR (see optional outputs, below, for
C          TCUR and HU).
C
C ITOL   = an indicator for the type of error control.  See
C          description below under ATOL.  Used only for input.
C
C RTOL   = a relative error tolerance parameter, either a scalar or
C          an array of length NEQ.  See description below under ATOL.
C          Input only.
C
C ATOL   = an absolute error tolerance parameter, either a scalar or
C          an array of length NEQ.  Input only.
C
C             The input parameters ITOL, RTOL, and ATOL determine
```

```
C          the error control performed by the solver.  The solver will
C          control the vector E = (E(i)) of estimated local errors
C          in y, according to an inequality of the form
C                      RMS-norm of ( E(i)/EWT(i) )   .le.   1,
C          where       EWT(i) = RTOL(i)*ABS(Y(i)) + ATOL(i),
C          and the RMS-norm (root-mean-square norm) here is
C          RMS-norm(v) = SQRT(sum v(i)**2 / NEQ).  Here EWT = (EWT(i))
C          is a vector of weights which must always be positive, and
C          the values of RTOL and ATOL should all be non-negative.
C          The following table gives the types (scalar/array) of
C          RTOL and ATOL, and the corresponding form of EWT(i).
C
C             ITOL    RTOL       ATOL          EWT(i)
C              1     scalar     scalar     RTOL*ABS(Y(i)) + ATOL
C              2     scalar     array      RTOL*ABS(Y(i)) + ATOL(i)
C              3     array      scalar     RTOL(i)*ABS(Y(i)) + ATOL
C              4     array      array      RTOL(i)*ABS(Y(i)) + ATOL(i)
C
C          When either of these parameters is a scalar, it need not
C          be dimensioned in the user's calling program.
C
C          If none of the above choices (with ITOL, RTOL, and ATOL
C          fixed throughout the problem) is suitable, more general
C          error controls can be obtained by substituting
C          user-supplied routines for the setting of EWT and/or for
C          the norm calculation.  See Part 4 below.
C
C          If global errors are to be estimated by making a repeated
C          run on the same problem with smaller tolerances, then all
C          components of RTOL and ATOL (i.e. of EWT) should be scaled
C          down uniformly.
C
C ITASK  = an index specifying the task to be performed.
C          Input only.  ITASK has the following values and meanings.
C          1  means normal computation of output values of y(t) at
C             t = TOUT (by overshooting and interpolating).
C          2  means take one step only and return.
C          3  means stop at the first internal mesh point at or
C             beyond t = TOUT and return.
C          4  means normal computation of output values of y(t) at
C             t = TOUT but without overshooting t = TCRIT.
C             TCRIT must be input as RWORK(1).  TCRIT may be equal to
C             or beyond TOUT, but not behind it in the direction of
C             integration.  This option is useful if the problem
C             has a singularity at or beyond t = TCRIT.
C          5  means take one step, without passing TCRIT, and return.
C             TCRIT must be input as RWORK(1).
C
C          Note:  If ITASK = 4 or 5 and the solver reaches TCRIT
C          (within roundoff), it will return T = TCRIT (exactly) to
C          indicate this (unless ITASK = 4 and TOUT comes before TCRIT,
C          in which case answers at t = TOUT are returned first).
C
C ISTATE = an index used for input and output to specify the
C          the state of the calculation.
C
C          On input, the values of ISTATE are as follows.
C          1  means this is the first call for the problem
C             (initializations will be done).  See note below.
C          2  means this is not the first call, and the calculation
C             is to continue normally, with no change in any input
C             parameters except possibly TOUT and ITASK.
C             (If ITOL, RTOL, and/or ATOL are changed between calls
C             with ISTATE = 2, the new values will be used but not
C             tested for legality.)
C          3  means this is not the first call, and the
C             calculation is to continue normally, but with
C             a change in input parameters other than
C             TOUT and ITASK.  Changes are allowed in
C             NEQ, ITOL, RTOL, ATOL, IOPT, LRW, LIW, MF,
C             the conditional inputs IA and JA,
C             and any of the optional inputs except H0.
C             In particular, if MITER = 1 or 2, a call with ISTATE = 3
C             will cause the sparsity structure of the problem to be
C             recomputed (or reread from IA and JA if MOSS = 0).
C          Note:  a preliminary call with TOUT = T is not counted
C          as a first call here, as no initialization or checking of
C          input is done.  (Such a call is sometimes useful for the
C          purpose of outputting the initial conditions.)
C          Thus the first call for which TOUT .ne. T requires
C          ISTATE = 1 on input.
C
C          On output, ISTATE has the following values and meanings.
C           1  means nothing was done; TOUT = T and ISTATE = 1 on input.
C           2  means the integration was performed successfully.
C          -1  means an excessive amount of work (more than MXSTEP
C              steps) was done on this call, before completing the
C              requested task, but the integration was otherwise
C              successful as far as T.  (MXSTEP is an optional input
C              and is normally 500.)  To continue, the user may
C              simply reset ISTATE to a value .gt. 1 and call again
C              (the excess work step counter will be reset to 0).
C              In addition, the user may increase MXSTEP to avoid
C              this error return (see below on optional inputs).
C          -2  means too much accuracy was requested for the precision
C              of the machine being used.  This was detected before
C              completing the requested task, but the integration
C              was successful as far as T.  To continue, the tolerance
C              parameters must be reset, and ISTATE must be set
C              to 3.  The optional output TOLSF may be used for this
C              purpose.  (Note: If this condition is detected before
C              taking any steps, then an illegal input return
C              (ISTATE = -3) occurs instead.)
C          -3  means illegal input was detected, before taking any
C              integration steps.  See written message for details.
C              Note:  If the solver detects an infinite loop of calls
C              to the solver with illegal input, it will cause
C              the run to stop.
C          -4  means there were repeated error test failures on
C              one attempted step, before completing the requested
C              task, but the integration was successful as far as T.
C              The problem may have a singularity, or the input
C              may be inappropriate.
C          -5  means there were repeated convergence test failures on
C              one attempted step, before completing the requested
C              task, but the integration was successful as far as T.
C              This may be caused by an inaccurate Jacobian matrix,
C              if one is being used.
C          -6  means EWT(i) became zero for some i during the
C              integration.  Pure relative error control (ATOL(i)=0.0)
C              was requested on a variable which has now vanished.
C              The integration was successful as far as T.
C          -7  means a fatal error return flag came from the sparse
C              solver CDRV by way of DPRJS or DSOLSS (numerical
C              factorization or backsolve).  This should never happen.
C              The integration was successful as far as T.
C
C          Note: an error return with ISTATE = -1, -4, or -5 and with
```

```
C          MITER = 1 or 2 may mean that the sparsity structure of the
C          problem has changed significantly since it was last
C          determined (or input).  In that case, one can attempt to
C          complete the integration by setting ISTATE = 3 on the next
C          call, so that a new structure determination is done.
C
C          Note:  since the normal output value of ISTATE is 2,
C          it does not need to be reset for normal continuation.
C          Also, since a negative input value of ISTATE will be
C          regarded as illegal, a negative output value requires the
C          user to change it, and possibly other inputs, before
C          calling the solver again.
C
C IOPT   = an integer flag to specify whether or not any optional
C          inputs are being used on this call.  Input only.
C          The optional inputs are listed separately below.
C          IOPT = 0 means no optional inputs are being used.
C                   Default values will be used in all cases.
C          IOPT = 1 means one or more optional inputs are being used.
C
C RWORK  = a work array used for a mixture of real (double precision)
C          and integer work space.
C          The length of RWORK (in real words) must be at least
C             20 + NYH*(MAXORD + 1) + 3*NEQ + LWM    where
C          NYH    = the initial value of NEQ,
C          MAXORD = 12 (if METH = 1) or 5 (if METH = 2) (unless a
C                   smaller value is given as an optional input),
C          LWM = 0                                  if MITER = 0,
C          LWM = 2*NNZ + 2*NEQ + (NNZ+9*NEQ)/LENRAT   if MITER = 1,
C          LWM = 2*NNZ + 2*NEQ + (NNZ+10*NEQ)/LENRAT  if MITER = 2,
C          LWM = NEQ + 2                            if MITER = 3.
C          In the above formulas,
C          NNZ    = number of nonzero elements in the Jacobian matrix.
C          LENRAT = the real to integer wordlength ratio (usually 1 in
C                   single precision and 2 in double precision).
C          (See the MF description for METH and MITER.)
C          Thus if MAXORD has its default value and NEQ is constant,
C          the minimum length of RWORK is:
C             20 + 16*NEQ        for MF = 10,
C             20 + 16*NEQ + LWM  for MF = 11, 111, 211, 12, 112, 212,
C             22 + 17*NEQ        for MF = 13,
C             20 +  9*NEQ        for MF = 20,
C             20 +  9*NEQ + LWM  for MF = 21, 121, 221, 22, 122, 222,
C             22 + 10*NEQ        for MF = 23.
C          If MITER = 1 or 2, the above formula for LWM is only a
C          crude lower bound.  The required length of RWORK cannot
C          be readily predicted in general, as it depends on the
C          sparsity structure of the problem.  Some experimentation
C          may be necessary.
C
C          The first 20 words of RWORK are reserved for conditional
C          and optional inputs and optional outputs.
C
C          The following word in RWORK is a conditional input:
C            RWORK(1) = TCRIT = critical value of t which the solver
C                       is not to overshoot.  Required if ITASK is
C                       4 or 5, and ignored otherwise.  (See ITASK.)
C
C LRW    = the length of the array RWORK, as declared by the user.
C          (This will be checked by the solver.)
C
C IWORK  = an integer work array.  The length of IWORK must be at least
C             31 + NEQ + NNZ   if MOSS = 0 and MITER = 1 or 2, or
C             30               otherwise.
```

```
C          (NNZ is the number of nonzero elements in df/dy.)
C
C          In DLSODES, IWORK is used only for conditional and
C          optional inputs and optional outputs.
C
C          The following two blocks of words in IWORK are conditional
C          inputs, required if MOSS = 0 and MITER = 1 or 2, but not
C          otherwise (see the description of MF for MOSS).
C            IWORK(30+j) = IA(j)     (j=1,...,NEQ+1)
C            IWORK(31+NEQ+k) = JA(k) (k=1,...,NNZ)
C          The two arrays IA and JA describe the sparsity structure
C          to be assumed for the Jacobian matrix.  JA contains the row
C          indices where nonzero elements occur, reading in columnwise
C          order, and IA contains the starting locations in JA of the
C          descriptions of columns 1,...,NEQ, in that order, with
C          IA(1) = 1.  Thus, for each column index j = 1,...,NEQ, the
C          values of the row index i in column j where a nonzero
C          element may occur are given by
C            i = JA(k),  where   IA(j) .le. k .lt. IA(j+1).
C          If NNZ is the total number of nonzero locations assumed,
C          then the length of the JA array is NNZ, and IA(NEQ+1) must
C          be NNZ + 1.  Duplicate entries are not allowed.
C
C LIW    = the length of the array IWORK, as declared by the user.
C          (This will be checked by the solver.)
C
C Note:  The work arrays must not be altered between calls to DLSODES
C for the same problem, except possibly for the conditional and
C optional inputs, and except for the last 3*NEQ words of RWORK.
C The latter space is used for internal scratch space, and so is
C available for use by the user outside DLSODES between calls, if
C desired (but not for use by F or JAC).
C
C JAC    = name of user-supplied routine (MITER = 1 or MOSS = 1) to
C          compute the Jacobian matrix, df/dy, as a function of
C          the scalar t and the vector y.  It is to have the form
C               SUBROUTINE JAC (NEQ, T, Y, J, IAN, JAN, PDJ)
C               DOUBLE PRECISION T, Y(*), IAN(*), JAN(*), PDJ(*)
C          where NEQ, T, Y, J, IAN, and JAN are input, and the array
C          PDJ, of length NEQ, is to be loaded with column J
C          of the Jacobian on output.  Thus df(i)/dy(J) is to be
C          loaded into PDJ(i) for all relevant values of i.
C          Here T and Y have the same meaning as in Subroutine F,
C          and J is a column index (1 to NEQ).  IAN and JAN are
C          undefined in calls to JAC for structure determination
C          (MOSS = 1).  otherwise, IAN and JAN are structure
C          descriptors, as defined under optional outputs below, and
C          so can be used to determine the relevant row indices i, if
C          desired.
C               JAC need not provide df/dy exactly.  A crude
C          approximation (possibly with greater sparsity) will do.
C               In any case, PDJ is preset to zero by the solver,
C          so that only the nonzero elements need be loaded by JAC.
C          Calls to JAC are made with J = 1,...,NEQ, in that order, and
C          each such set of calls is preceded by a call to F with the
C          same arguments NEQ, T, and Y.  Thus to gain some efficiency,
C          intermediate quantities shared by both calculations may be
C          saved in a user Common block by F and not recomputed by JAC,
C          if desired.  JAC must not alter its input arguments.
C          JAC must be declared External in the calling program.
C               Subroutine JAC may access user-defined quantities in
C          NEQ(2),... and/or in Y(NEQ(1)+1),... if NEQ is an array
C          (dimensioned in JAC) and/or Y has length exceeding NEQ(1).
C          See the descriptions of NEQ and Y above.
```

```
C
C MF     = the method flag.  Used only for input.
C          MF has three decimal digits-- MOSS, METH, MITER--
C             MF = 100*MOSS + 10*METH + MITER.
C          MOSS indicates the method to be used to obtain the sparsity
C          structure of the Jacobian matrix if MITER = 1 or 2:
C            MOSS = 0 means the user has supplied IA and JA
C                     (see descriptions under IWORK above).
C            MOSS = 1 means the user has supplied JAC (see below)
C                     and the structure will be obtained from NEQ
C                     initial calls to JAC.
C            MOSS = 2 means the structure will be obtained from NEQ+1
C                     initial calls to F.
C          METH indicates the basic linear multistep method:
C            METH = 1 means the implicit Adams method.
C            METH = 2 means the method based on Backward
C                     Differentiation Formulas (BDFs).
C          MITER indicates the corrector iteration method:
C            MITER = 0 means functional iteration (no Jacobian matrix
C                      is involved).
C            MITER = 1 means chord iteration with a user-supplied
C                      sparse Jacobian, given by Subroutine JAC.
C            MITER = 2 means chord iteration with an internally
C                      generated (difference quotient) sparse Jacobian
C                      (using NGP extra calls to F per df/dy value,
C                      where NGP is an optional output described below.)
C            MITER = 3 means chord iteration with an internally
C                      generated diagonal Jacobian approximation
C                      (using 1 extra call to F per df/dy evaluation).
C          If MITER = 1 or MOSS = 1, the user must supply a Subroutine
C          JAC (the name is arbitrary) as described above under JAC.
C          Otherwise, a dummy argument can be used.
C
C          The standard choices for MF are:
C            MF = 10  for a nonstiff problem,
C            MF = 21 or 22 for a stiff problem with IA/JA supplied
C                     (21 if JAC is supplied, 22 if not),
C            MF = 121 for a stiff problem with JAC supplied,
C                     but not IA/JA,
C            MF = 222 for a stiff problem with neither IA/JA nor
C                     JAC supplied.
C          The sparseness structure can be changed during the
C          problem by making a call to DLSODES with ISTATE = 3.
C-----------------------------------------------------------------------
C Optional Inputs.
C
C The following is a list of the optional inputs provided for in the
C call sequence.  (See also Part 2.)  For each such input variable,
C this table lists its name as used in this documentation, its
C location in the call sequence, its meaning, and the default value.
C The use of any of these inputs requires IOPT = 1, and in that
C case all of these inputs are examined.  A value of zero for any
C of these optional inputs will cause the default value to be used.
C Thus to use a subset of the optional inputs, simply preload
C locations 5 to 10 in RWORK and IWORK to 0.0 and 0 respectively, and
C then set those of interest to nonzero values.
C
C Name    Location      Meaning and Default Value
C
C H0      RWORK(5)  the step size to be attempted on the first step.
C                   The default value is determined by the solver.
C
C HMAX    RWORK(6)  the maximum absolute step size allowed.
C                   The default value is infinite.
```

```
C
C HMIN    RWORK(7)  the minimum absolute step size allowed.
C                   The default value is 0.  (This lower bound is not
C                   enforced on the final step before reaching TCRIT
C                   when ITASK = 4 or 5.)
C
C SETH    RWORK(8)  the element threshhold for sparsity determination
C                   when MOSS = 1 or 2.  If the absolute value of
C                   an estimated Jacobian element is .le. SETH, it
C                   will be assumed to be absent in the structure.
C                   The default value of SETH is 0.
C
C MAXORD  IWORK(5)  the maximum order to be allowed.  The default
C                   value is 12 if METH = 1, and 5 if METH = 2.
C                   If MAXORD exceeds the default value, it will
C                   be reduced to the default value.
C                   If MAXORD is changed during the problem, it may
C                   cause the current order to be reduced.
C
C MXSTEP  IWORK(6)  maximum number of (internally defined) steps
C                   allowed during one call to the solver.
C                   The default value is 500.
C
C MXHNIL  IWORK(7)  maximum number of messages printed (per problem)
C                   warning that T + H = T on a step (H = step size).
C                   This must be positive to result in a non-default
C                   value.  The default value is 10.
C-----------------------------------------------------------------------
C Optional Outputs.
C
C As optional additional output from DLSODES, the variables listed
C below are quantities related to the performance of DLSODES
C which are available to the user.  These are communicated by way of
C the work arrays, but also have internal mnemonic names as shown.
C Except where stated otherwise, all of these outputs are defined
C on any successful return from DLSODES, and on any return with
C ISTATE = -1, -2, -4, -5, or -6.  On an illegal input return
C (ISTATE = -3), they will be unchanged from their existing values
C (if any), except possibly for TOLSF, LENRW, and LENIW.
C On any error return, outputs relevant to the error will be defined,
C as noted below.
C
C Name    Location      Meaning
C
C HU      RWORK(11) the step size in t last used (successfully).
C
C HCUR    RWORK(12) the step size to be attempted on the next step.
C
C TCUR    RWORK(13) the current value of the independent variable
C                   which the solver has actually reached, i.e. the
C                   current internal mesh point in t.  On output, TCUR
C                   will always be at least as far as the argument
C                   T, but may be farther (if interpolation was done).
C
C TOLSF   RWORK(14) a tolerance scale factor, greater than 1.0,
C                   computed when a request for too much accuracy was
C                   detected (ISTATE = -3 if detected at the start of
C                   the problem, ISTATE = -2 otherwise).  If ITOL is
C                   left unaltered but RTOL and ATOL are uniformly
C                   scaled up by a factor of TOLSF for the next call,
C                   then the solver is deemed likely to succeed.
C                   (The user may also ignore TOLSF and alter the
C                   tolerance parameters in any other way appropriate.)
C
```

```
C NST     IWORK(11) the number of steps taken for the problem so far.
C
C NFE     IWORK(12) the number of f evaluations for the problem so far,
C                   excluding those for structure determination
C                   (MOSS = 2).
C
C NJE     IWORK(13) the number of Jacobian evaluations for the problem
C                   so far, excluding those for structure determination
C                   (MOSS = 1).
C
C NQU     IWORK(14) the method order last used (successfully).
C
C NQCUR   IWORK(15) the order to be attempted on the next step.
C
C IMXER   IWORK(16) the index of the component of largest magnitude in
C                   the weighted local error vector ( E(i)/EWT(i) ),
C                   on an error return with ISTATE = -4 or -5.
C
C LENRW   IWORK(17) the length of RWORK actually required.
C                   This is defined on normal returns and on an illegal
C                   input return for insufficient storage.
C
C LENIW   IWORK(18) the length of IWORK actually required.
C                   This is defined on normal returns and on an illegal
C                   input return for insufficient storage.
C
C NNZ     IWORK(19) the number of nonzero elements in the Jacobian
C                   matrix, including the diagonal (MITER = 1 or 2).
C                   (This may differ from that given by IA(NEQ+1)-1
C                   if MOSS = 0, because of added diagonal entries.)
C
C NGP     IWORK(20) the number of groups of column indices, used in
C                   difference quotient Jacobian aproximations if
C                   MITER = 2.  This is also the number of extra f
C                   evaluations needed for each Jacobian evaluation.
C
C NLU     IWORK(21) the number of sparse LU decompositions for the
C                   problem so far.
C
C LYH     IWORK(22) the base address in RWORK of the history array YH,
C                   described below in this list.
C
C IPIAN   IWORK(23) the base address of the structure descriptor array
C                   IAN, described below in this list.
C
C IPJAN   IWORK(24) the base address of the structure descriptor array
C                   JAN, described below in this list.
C
C NZL     IWORK(25) the number of nonzero elements in the strict lower
C                   triangle of the LU factorization used in the chord
C                   iteration (MITER = 1 or 2).
C
C NZU     IWORK(26) the number of nonzero elements in the strict upper
C                   triangle of the LU factorization used in the chord
C                   iteration (MITER = 1 or 2).
C                   The total number of nonzeros in the factorization
C                   is therefore NZL + NZU + NEQ.
C
C The following four arrays are segments of the RWORK array which
C may also be of interest to the user as optional outputs.
C For each array, the table below gives its internal name,
C its base address, and its description.
C For YH and ACOR, the base addresses are in RWORK (a real array).
C The integer arrays IAN and JAN are to be obtained by declaring an
```

```
C integer array IWK and identifying IWK(1) with RWORK(21), using either
C an equivalence statement or a subroutine call.  Then the base
C addresses IPIAN (of IAN) and IPJAN (of JAN) in IWK are to be obtained
C as optional outputs IWORK(23) and IWORK(24), respectively.
C Thus IAN(1) is IWK(IPIAN), etc.
C
C Name    Base Address      Description
C
C IAN    IPIAN (in IWK)  structure descriptor array of size NEQ + 1.
C JAN    IPJAN (in IWK)  structure descriptor array of size NNZ.
C         (see above)    IAN and JAN together describe the sparsity
C                        structure of the Jacobian matrix, as used by
C                        DLSODES when MITER = 1 or 2.
C                        JAN contains the row indices of the nonzero
C                        locations, reading in columnwise order, and
C                        IAN contains the starting locations in JAN of
C                        the descriptions of columns 1,...,NEQ, in
C                        that order, with IAN(1) = 1.  Thus for each
C                        j = 1,...,NEQ, the row indices i of the
C                        nonzero locations in column j are
C                        i = JAN(k),  IAN(j) .le. k .lt. IAN(j+1).
C                        Note that IAN(NEQ+1) = NNZ + 1.
C                        (If MOSS = 0, IAN/JAN may differ from the
C                        input IA/JA because of a different ordering
C                        in each column, and added diagonal entries.)
C
C YH      LYH            the Nordsieck history array, of size NYH by
C          (optional     (NQCUR + 1), where NYH is the initial value
C           output)      of NEQ.  For j = 0,1,...,NQCUR, column j+1
C                        of YH contains HCUR**j/factorial(j) times
C                        the j-th derivative of the interpolating
C                        polynomial currently representing the solution,
C                        evaluated at t = TCUR.  The base address LYH
C                        is another optional output, listed above.
C
C ACOR     LENRW-NEQ+1   array of size NEQ used for the accumulated
C                        corrections on each step, scaled on output
C                        to represent the estimated local error in y
C                        on the last step.  This is the vector E  in
C                        the description of the error control.  It is
C                        defined only on a successful return from
C                        DLSODES.
C
C-----------------------------------------------------------------------
C Part 2.  Other Routines Callable.
C
C The following are optional calls which the user may make to
C gain additional capabilities in conjunction with DLSODES.
C (The routines XSETUN and XSETF are designed to conform to the
C SLATEC error handling package.)
C
C     Form of Call                  Function
C   CALL XSETUN(LUN)          Set the logical unit number, LUN, for
C                             output of messages from DLSODES, if
C                             the default is not desired.
C                             The default value of LUN is 6.
C
C   CALL XSETF(MFLAG)         Set a flag to control the printing of
C                             messages by DLSODES.
C                             MFLAG = 0 means do not print. (Danger:
C                             This risks losing valuable information.)
C                             MFLAG = 1 means print (the default).
C
C                             Either of the above calls may be made at
```

```
C                             any time and will take effect immediately.
C
C   CALL DSRCMS(RSAV,ISAV,JOB) saves and restores the contents of
C                             the internal Common blocks used by
C                             DLSODES (see Part 3 below).
C                             RSAV must be a real array of length 224
C                             or more, and ISAV must be an integer
C                             array of length 71 or more.
C                             JOB=1 means save Common into RSAV/ISAV.
C                             JOB=2 means restore Common from RSAV/ISAV.
C                                DSRCMS is useful if one is
C                             interrupting a run and restarting
C                             later, or alternating between two or
C                             more problems solved with DLSODES.
C
C   CALL DINTDY(,,,,,)        Provide derivatives of y, of various
C        (see below)          orders, at a specified point t, if
C                             desired.  It may be called only after
C                             a successful return from DLSODES.
C
C The detailed instructions for using DINTDY are as follows.
C The form of the call is:
C
C   LYH = IWORK(22)
C   CALL DINTDY (T, K, RWORK(LYH), NYH, DKY, IFLAG)
C
C The input parameters are:
C
C T         = value of independent variable where answers are desired
C             (normally the same as the T last returned by DLSODES).
C             For valid results, T must lie between TCUR - HU and TCUR.
C             (See optional outputs for TCUR and HU.)
C K         = integer order of the derivative desired.  K must satisfy
C             0 .le. K .le. NQCUR, where NQCUR is the current order
C             (See optional outputs).  The capability corresponding
C             to K = 0, i.e. computing y(T), is already provided
C             by DLSODES directly.  Since NQCUR .ge. 1, the first
C             derivative dy/dt is always available with DINTDY.
C LYH       = the base address of the history array YH, obtained
C             as an optional output as shown above.
C NYH       = column length of YH, equal to the initial value of NEQ.
C
C The output parameters are:
C
C DKY       = a real array of length NEQ containing the computed value
C             of the K-th derivative of y(t).
C IFLAG     = integer flag, returned as 0 if K and T were legal,
C             -1 if K was illegal, and -2 if T was illegal.
C             On an error return, a message is also written.
C-----------------------------------------------------------------------
C Part 3.  Common Blocks.
C
C If DLSODES is to be used in an overlay situation, the user
C must declare, in the primary overlay, the variables in:
C   (1) the call sequence to DLSODES, and
C   (2) the two internal Common blocks
C         /DLS001/  of length  255  (218 double precision words
C                      followed by 37 integer words),
C         /DLSS01/  of length  40  (6 double precision words
C                      followed by 34 integer words),
C
C If DLSODES is used on a system in which the contents of internal
C Common blocks are not preserved between calls, the user should
C declare the above Common blocks in the calling program to insure
```

```
C that their contents are preserved.
C
C If the solution of a given problem by DLSODES is to be interrupted
C and then later continued, such as when restarting an interrupted run
C or alternating between two or more problems, the user should save,
C following the return from the last DLSODES call prior to the
C interruption, the contents of the call sequence variables and the
C internal Common blocks, and later restore these values before the
C next DLSODES call for that problem.  To save and restore the Common
C blocks, use Subroutine DSRCMS (see Part 2 above).
C
C-----------------------------------------------------------------------
C Part 4.  Optionally Replaceable Solver Routines.
C
C Below are descriptions of two routines in the DLSODES package which
C relate to the measurement of errors.  Either routine can be
C replaced by a user-supplied version, if desired.  However, since such
C a replacement may have a major impact on performance, it should be
C done only when absolutely necessary, and only with great caution.
C (Note: The means by which the package version of a routine is
C superseded by the user's version may be system-dependent.)
C
C (a) DEWSET.
C The following subroutine is called just before each internal
C integration step, and sets the array of error weights, EWT, as
C described under ITOL/RTOL/ATOL above:
C     Subroutine DEWSET (NEQ, ITOL, RTOL, ATOL, YCUR, EWT)
C where NEQ, ITOL, RTOL, and ATOL are as in the DLSODES call sequence,
C YCUR contains the current dependent variable vector, and
C EWT is the array of weights set by DEWSET.
C
C If the user supplies this subroutine, it must return in EWT(i)
C (i = 1,...,NEQ) a positive quantity suitable for comparing errors
C in y(i) to.  The EWT array returned by DEWSET is passed to the DVNORM
C routine (see below), and also used by DLSODES in the computation
C of the optional output IMXER, the diagonal Jacobian approximation,
C and the increments for difference quotient Jacobians.
C
C In the user-supplied version of DEWSET, it may be desirable to use
C the current values of derivatives of y.  Derivatives up to order NQ
C are available from the history array YH, described above under
C optional outputs.  In DEWSET, YH is identical to the YCUR array,
C extended to NQ + 1 columns with a column length of NYH and scale
C factors of H**j/factorial(j).  On the first call for the problem,
C given by NST = 0, NQ is 1 and H is temporarily set to 1.0.
C NYH is the initial value of NEQ.  The quantities NQ, H, and NST
C can be obtained by including in DEWSET the statements:
C     DOUBLE PRECISION RLS
C     COMMON /DLS001/ RLS(218),ILS(37)
C     NQ = ILS(33)
C     NST = ILS(34)
C     H = RLS(212)
C Thus, for example, the current value of dy/dt can be obtained as
C YCUR(NYH+i)/H  (i=1,...,NEQ)  (and the division by H is
C unnecessary when NST = 0).
C
C (b) DVNORM.
C The following is a real function routine which computes the weighted
C root-mean-square norm of a vector v:
C     D = DVNORM (N, V, W)
C where
C   N = the length of the vector,
C   V = real array of length N containing the vector,
C   W = real array of length N containing weights,
C   D = SQRT( (1/N) * sum(V(i)*W(i))**2 ).
C DVNORM is called with N = NEQ and with W(i) = 1.0/EWT(i), where
C EWT is as set by Subroutine DEWSET.
C
C If the user supplies this function, it should return a non-negative
C value of DVNORM suitable for use in the error control in DLSODES.
C None of the arguments should be altered by DVNORM.
C For example, a user-supplied DVNORM routine might:
C   -substitute a max-norm of (V(i)*W(i)) for the RMS-norm, or
C   -ignore some components of V in the norm, with the effect of
C    suppressing the error control on those components of y.
C-----------------------------------------------------------------------
C
C***REVISION HISTORY  (YYYYMMDD)
C 19810120  DATE WRITTEN
C 19820315  Upgraded MDI in ODRV package: operates on M + M-transpose.
C 19820426  Numerous revisions in use of work arrays;
C           use wordlength ratio LENRAT; added IPISP & LRAT to Common;
C           added optional outputs IPIAN/IPJAN;
C           numerous corrections to comments.
C 19830503  Added routine CNTNZU; added NZL and NZU to /LSS001/;
C           changed ADJLR call logic; added optional outputs NZL & NZU;
C           revised counter initializations; revised PREP stmt. numbers;
C           corrections to comments throughout.
C 19870320  Corrected jump on test of umax in CDRV routine;
C           added ISTATE = -7 return.
C 19870330  Major update: corrected comments throughout;
C           removed TRET from Common; rewrote EWSET with 4 loops;
C           fixed t test in INTDY; added Cray directives in STODE;
C           in STODE, fixed DELP init. and logic around PJAC call;
C           combined routines to save/restore Common;
C           passed LEVEL = 0 in error message calls (except run abort).
C 20010425  Major update: convert source lines to upper case;
C           added *DECK lines; changed from 1 to * in dummy dimensions;
C           changed names R1MACH/D1MACH to RUMACH/DUMACH;
C           renamed routines for uniqueness across single/double prec.;
C           converted intrinsic names to generic form;
C           removed ILLIN and NTREP (data loaded) from Common;
C           removed all 'own' variables from Common;
C           changed error messages to quoted strings;
C           replaced XERRWV/XERRWD with 1993 revised version;
C           converted prologues, comments, error messages to mixed case;
C           converted arithmetic IF statements to logical IF statements;
C           numerous corrections to prologues and internal comments.
C 20010507  Converted single precision source to double precision.
C 20020502  Corrected declarations in descriptions of user routines.
C 20031105  Restored 'own' variables to Common blocks, to enable
C           interrupt/restart feature.
C 20031112  Added SAVE statements for data-loaded constants.
C
C-----------------------------------------------------------------------
C Other routines in the DLSODES package.
C
C In addition to Subroutine DLSODES, the DLSODES package includes the
C following subroutines and function routines:
C  DIPREP   acts as an iterface between DLSODES and DPREP, and also does
C           adjusting of work space pointers and work arrays.
C  DPREP    is called by DIPREP to compute sparsity and do sparse matrix
C           preprocessing if MITER = 1 or 2.
C  JGROUP   is called by DPREP to compute groups of Jacobian column
C           indices for use when MITER = 2.
C  ADJLR    adjusts the length of required sparse matrix work space.
C           It is called by DPREP.
C  CNTNZU   is called by DPREP and counts the nonzero elements in the
```

```
C           strict upper triangle of J + J-transpose, where J = df/dy.
C  DINTDY   computes an interpolated value of the y vector at t = TOUT.
C  DSTODE   is the core integrator, which does one step of the
C           integration and the associated error control.
C  DCFODE   sets all method coefficients and test constants.
C  DPRJS    computes and preprocesses the Jacobian matrix J = df/dy
C           and the Newton iteration matrix P = I - h*l0*J.
C  DSOLSS   manages solution of linear system in chord iteration.
C  DEWSET   sets the error weight vector EWT before each step.
C  DVNORM   computes the weighted RMS-norm of a vector.
C  DSRCMS   is a user-callable routine to save and restore
C           the contents of the internal Common blocks.
C  ODRV     constructs a reordering of the rows and columns of
C           a matrix by the minimum degree algorithm.  ODRV is a
C           driver routine which calls Subroutines MD, MDI, MDM,
C           MDP, MDU, and SRO.  See Ref. 2 for details.  (The ODRV
C           module has been modified since Ref. 2, however.)
C  CDRV     performs reordering, symbolic factorization, numerical
C           factorization, or linear system solution operations,
C           depending on a path argument ipath.  CDRV is a
C           driver routine which calls Subroutines NROC, NSFC,
C           NNFC, NNSC, and NNTC.  See Ref. 3 for details.
C           DLSODES uses CDRV to solve linear systems in which the
C           coefficient matrix is  P = I - con*J, where I is the
C           identity, con is a scalar, and J is an approximation to
C           the Jacobian df/dy.  Because CDRV deals with rowwise
C           sparsity descriptions, CDRV works with P-transpose, not P.
C  DUMACH   computes the unit roundoff in a machine-independent manner.
C  XERRWD, XSETUN, XSETF, IXSAV, and IUMACH  handle the printing of all
C           error messages and warnings.  XERRWD is machine-dependent.
C Note:  DVNORM, DUMACH, IXSAV, and IUMACH are function routines.
C All the others are subroutines.
C
C-----------------------------------------------------------------------
      EXTERNAL DPRJS, DSOLSS
      DOUBLE PRECISION DUMACH, DVNORM
      INTEGER INIT, MXSTEP, MXHNIL, NHNIL, NSLAST, NYH, IOWNS,
     1   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L,
     2   LYH, LEWT, LACOR, LSAVF, LWM, LIWM, METH, MITER,
     3   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      INTEGER IPLOST, IESP, ISTATC, IYS, IBA, IBIAN, IBJAN, IBJGP,
     1   IPIAN, IPJAN, IPJGP, IPIGP, IPR, IPC, IPIC, IPISP, IPRSP, IPA,
     2   LENYH, LENYHM, LENWK, LREQ, LRAT, LREST, LWMIN, MOSS, MSBJ,
     3   NSLJ, NGP, NLU, NNZ, NSP, NZL, NZU
      INTEGER I, I1, I2, IFLAG, IMAX, IMUL, IMXER, IPFLAG, IPGO, IREM,
     1   J, KGO, LENRAT, LENYHT, LENIW, LENRW, LF0, LIA, LJA,
     2   LRTEM, LWTEM, LYHD, LYHN, MF1, MORD, MXHNL0, MXSTP0, NCOLM
      DOUBLE PRECISION ROWNS,
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND
      DOUBLE PRECISION CON0, CONMIN, CCMXJ, PSMALL, RBIG, SETH
      DOUBLE PRECISION ATOLI, AYI, BIG, EWTI, H0, HMAX, HMX, RH, RTOLI,
     1   TCRIT, TDIST, TNEXT, TOL, TOLSF, TP, SIZE, SUM, W0
      DIMENSION MORD(2)
      LOGICAL IHIT
      CHARACTER*60 MSG
      SAVE LENRAT, MORD, MXSTP0, MXHNL0
C-----------------------------------------------------------------------
C The following two internal Common blocks contain
C (a) variables which are local to any subroutine but whose values must
C     be preserved between calls to the routine ("own" variables), and
C (b) variables which are communicated between subroutines.
C The block DLS001 is declared in subroutines DLSODES, DIPREP, DPREP,
C DINTDY, DSTODE, DPRJS, and DSOLSS.
C The block DLSS01 is declared in subroutines DLSODES, DIPREP, DPREP,
C DPRJS, and DSOLSS.
C Groups of variables are replaced by dummy arrays in the Common
C declarations in routines where those variables are not used.
C-----------------------------------------------------------------------
      COMMON /DLS001/ ROWNS(209),
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND,
     2   INIT, MXSTEP, MXHNIL, NHNIL, NSLAST, NYH, IOWNS(6),
     3   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L,
     4   LYH, LEWT, LACOR, LSAVF, LWM, LIWM, METH, MITER,
     5   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
C
      COMMON /DLSS01/ CON0, CONMIN, CCMXJ, PSMALL, RBIG, SETH,
     1   IPLOST, IESP, ISTATC, IYS, IBA, IBIAN, IBJAN, IBJGP,
     2   IPIAN, IPJAN, IPJGP, IPIGP, IPR, IPC, IPIC, IPISP, IPRSP, IPA,
     3   LENYH, LENYHM, LENWK, LREQ, LRAT, LREST, LWMIN, MOSS, MSBJ,
     4   NSLJ, NGP, NLU, NNZ, NSP, NZL, NZU
C
      DATA MORD(1),MORD(2)/12,5/, MXSTP0/500/, MXHNL0/10/
C-----------------------------------------------------------------------
C In the Data statement below, set LENRAT equal to the ratio of
C the wordlength for a real number to that for an integer.  Usually,
C LENRAT = 1 for single precision and 2 for double precision.  If the
C true ratio is not an integer, use the next smaller integer (.ge. 1).
C-----------------------------------------------------------------------
      DATA LENRAT/2/
C-----------------------------------------------------------------------
C Block A.
C This code block is executed on every call.
C It tests ISTATE and ITASK for legality and branches appropriately.
C If ISTATE .gt. 1 but the flag INIT shows that initialization has
C not yet been done, an error return occurs.
C If ISTATE = 1 and TOUT = T, return immediately.
C-----------------------------------------------------------------------
      IF (ISTATE .LT. 1 .OR. ISTATE .GT. 3) GO TO 601
      IF (ITASK .LT. 1 .OR. ITASK .GT. 5) GO TO 602
      IF (ISTATE .EQ. 1) GO TO 10
      IF (INIT .EQ. 0) GO TO 603
      IF (ISTATE .EQ. 2) GO TO 200
      GO TO 20
 10   INIT = 0
      IF (TOUT .EQ. T) RETURN
C-----------------------------------------------------------------------
C Block B.
C The next code block is executed for the initial call (ISTATE = 1),
C or for a continuation call with parameter changes (ISTATE = 3).
C It contains checking of all inputs and various initializations.
C If ISTATE = 1, the final setting of work space pointers, the matrix
C preprocessing, and other initializations are done in Block C.
C
C First check legality of the non-optional inputs NEQ, ITOL, IOPT,
C MF, ML, and MU.
C-----------------------------------------------------------------------
 20   IF (NEQ(1) .LE. 0) GO TO 604
      IF (ISTATE .EQ. 1) GO TO 25
      IF (NEQ(1) .GT. N) GO TO 605
 25   N = NEQ(1)
      IF (ITOL .LT. 1 .OR. ITOL .GT. 4) GO TO 606
      IF (IOPT .LT. 0 .OR. IOPT .GT. 1) GO TO 607
      MOSS = MF/100
      MF1 = MF - 100*MOSS
      METH = MF1/10
      MITER = MF1 - 10*METH
      IF (MOSS .LT. 0 .OR. MOSS .GT. 2) GO TO 608
      IF (METH .LT. 1 .OR. METH .GT. 2) GO TO 608
      IF (MITER .LT. 0 .OR. MITER .GT. 3) GO TO 608
      IF (MITER .EQ. 0 .OR. MITER .EQ. 3) MOSS = 0
C Next process and check the optional inputs. -------------------------
      IF (IOPT .EQ. 1) GO TO 40
      MAXORD = MORD(METH)
      MXSTEP = MXSTP0
      MXHNIL = MXHNL0
      IF (ISTATE .EQ. 1) H0 = 0.0D0
      HMXI = 0.0D0
      HMIN = 0.0D0
      SETH = 0.0D0
      GO TO 60
 40   MAXORD = IWORK(5)
      IF (MAXORD .LT. 0) GO TO 611
      IF (MAXORD .EQ. 0) MAXORD = 100
      MAXORD = MIN(MAXORD,MORD(METH))
      MXSTEP = IWORK(6)
      IF (MXSTEP .LT. 0) GO TO 612
      IF (MXSTEP .EQ. 0) MXSTEP = MXSTP0
      MXHNIL = IWORK(7)
      IF (MXHNIL .LT. 0) GO TO 613
      IF (MXHNIL .EQ. 0) MXHNIL = MXHNL0
      IF (ISTATE .NE. 1) GO TO 50
      H0 = RWORK(5)
      IF ((TOUT - T)*H0 .LT. 0.0D0) GO TO 614
 50   HMAX = RWORK(6)
      IF (HMAX .LT. 0.0D0) GO TO 615
      HMXI = 0.0D0
      IF (HMAX .GT. 0.0D0) HMXI = 1.0D0/HMAX
      HMIN = RWORK(7)
      IF (HMIN .LT. 0.0D0) GO TO 616
      SETH = RWORK(8)
      IF (SETH .LT. 0.0D0) GO TO 609
C Check RTOL and ATOL for legality. ------------------------------------
 60   RTOLI = RTOL(1)
      ATOLI = ATOL(1)
      DO 65 I = 1,N
        IF (ITOL .GE. 3) RTOLI = RTOL(I)
        IF (ITOL .EQ. 2 .OR. ITOL .EQ. 4) ATOLI = ATOL(I)
        IF (RTOLI .LT. 0.0D0) GO TO 619
        IF (ATOLI .LT. 0.0D0) GO TO 620
 65     CONTINUE
C-----------------------------------------------------------------------
C Compute required work array lengths, as far as possible, and test
C these against LRW and LIW.  Then set tentative pointers for work
C arrays.  Pointers to RWORK/IWORK segments are named by prefixing L to
C the name of the segment.  E.g., the segment YH starts at RWORK(LYH).
C Segments of RWORK (in order) are denoted  WM, YH, SAVF, EWT, ACOR.
C If MITER = 1 or 2, the required length of the matrix work space WM
C is not yet known, and so a crude minimum value is used for the
C initial tests of LRW and LIW, and YH is temporarily stored as far
C to the right in RWORK as possible, to leave the maximum amount
C of space for WM for matrix preprocessing.  Thus if MITER = 1 or 2
C and MOSS .ne. 2, some of the segments of RWORK are temporarily
C omitted, as they are not needed in the preprocessing.  These
C omitted segments are: ACOR if ISTATE = 1, EWT and ACOR if ISTATE = 3
C and MOSS = 1, and SAVF, EWT, and ACOR if ISTATE = 3 and MOSS = 0.
C-----------------------------------------------------------------------
      LRAT = LENRAT
      IF (ISTATE .EQ. 1) NYH = N
      LWMIN = 0
      IF (MITER .EQ. 1) LWMIN = 4*N + 10*N/LRAT
      IF (MITER .EQ. 2) LWMIN = 4*N + 11*N/LRAT
      IF (MITER .EQ. 3) LWMIN = N + 2
      LENYH = (MAXORD+1)*NYH
      LREST = LENYH + 3*N
      LENRW = 20 + LWMIN + LREST
      IWORK(17) = LENRW
      LENIW = 30
      IF (MOSS .EQ. 0 .AND. MITER .NE. 0 .AND. MITER .NE. 3)
     1   LENIW = LENIW + N + 1
      IWORK(18) = LENIW
      IF (LENRW .GT. LRW) GO TO 617
      IF (LENIW .GT. LIW) GO TO 618
      LIA = 31
      IF (MOSS .EQ. 0 .AND. MITER .NE. 0 .AND. MITER .NE. 3)
     1   LENIW = LENIW + IWORK(LIA+N) - 1
      IWORK(18) = LENIW
      IF (LENIW .GT. LIW) GO TO 618
      LJA = LIA + N + 1
      LIA = MIN(LIA,LIW)
      LJA = MIN(LJA,LIW)
      LWM = 21
      IF (ISTATE .EQ. 1) NQ = 1
      NCOLM = MIN(NQ+1,MAXORD+2)
      LENYHM = NCOLM*NYH
      LENYHT = LENYH
      IF (MITER .EQ. 1 .OR. MITER .EQ. 2) LENYHT = LENYHM
      IMUL = 2
      IF (ISTATE .EQ. 3) IMUL = MOSS
      IF (MOSS .EQ. 2) IMUL = 3
      LRTEM = LENYHT + IMUL*N
      LWTEM = LWMIN
      IF (MITER .EQ. 1 .OR. MITER .EQ. 2) LWTEM = LRW - 20 - LRTEM
      LENWK = LWTEM
      LYHN = LWM + LWTEM
      LSAVF = LYHN + LENYHT
      LEWT = LSAVF + N
      LACOR = LEWT + N
      ISTATC = ISTATE
      IF (ISTATE .EQ. 1) GO TO 100
C-----------------------------------------------------------------------
C ISTATE = 3.  Move YH to its new location.
C Note that only the part of YH needed for the next step, namely
C MIN(NQ+1,MAXORD+2) columns, is actually moved.
C A temporary error weight array EWT is loaded if MOSS = 2.
C Sparse matrix processing is done in DIPREP/DPREP if MITER = 1 or 2.
C If MAXORD was reduced below NQ, then the pointers are finally set
C so that SAVF is identical to YH(*,MAXORD+2).
C-----------------------------------------------------------------------
      LYHD = LYH - LYHN
      IMAX = LYHN - 1 + LENYHM
C Move YH.  Move right if LYHD < 0; move left if LYHD > 0. -------------
      IF (LYHD .LT. 0) THEN
        DO 72 I = LYHN,IMAX
          J = IMAX + LYHN - I
 72       RWORK(J) = RWORK(J+LYHD)
      ENDIF
      IF (LYHD .GT. 0) THEN
        DO 76 I = LYHN,IMAX
 76       RWORK(I) = RWORK(I+LYHD)
      ENDIF
 80   LYH = LYHN
      IWORK(22) = LYH
      IF (MITER .EQ. 0 .OR. MITER .EQ. 3) GO TO 92
      IF (MOSS .NE. 2) GO TO 85
C Temporarily load EWT if MITER = 1 or 2 and MOSS = 2. -----------------
      CALL DEWSET (N, ITOL, RTOL, ATOL, RWORK(LYH), RWORK(LEWT))
      DO 82 I = 1,N
        IF (RWORK(I+LEWT-1) .LE. 0.0D0) GO TO 621
 82     RWORK(I+LEWT-1) = 1.0D0/RWORK(I+LEWT-1)
 85   CONTINUE
C DIPREP and DPREP do sparse matrix preprocessing if MITER = 1 or 2. ---
      LSAVF = MIN(LSAVF,LRW)
      LEWT = MIN(LEWT,LRW)
      LACOR = MIN(LACOR,LRW)
      CALL DIPREP (NEQ, Y, RWORK, IWORK(LIA),IWORK(LJA), IPFLAG, F, JAC)
      LENRW = LWM - 1 + LENWK + LREST
      IWORK(17) = LENRW
      IF (IPFLAG .NE. -1) IWORK(23) = IPIAN
      IF (IPFLAG .NE. -1) IWORK(24) = IPJAN
      IPGO = -IPFLAG + 1
      GO TO (90, 628, 629, 630, 631, 632, 633), IPGO
 90   IWORK(22) = LYH
      IF (LENRW .GT. LRW) GO TO 617
C Set flag to signal parameter changes to DSTODE. ----------------------
 92   JSTART = -1
      IF (N .EQ. NYH) GO TO 200
C NEQ was reduced.  Zero part of YH to avoid undefined references. -----
      I1 = LYH + L*NYH
      I2 = LYH + (MAXORD + 1)*NYH - 1
      IF (I1 .GT. I2) GO TO 200
      DO 95 I = I1,I2
 95     RWORK(I) = 0.0D0
      GO TO 200
C-----------------------------------------------------------------------
C Block C.
C The next block is for the initial call only (ISTATE = 1).
C It contains all remaining initializations, the initial call to F,
C the sparse matrix preprocessing (MITER = 1 or 2), and the
C calculation of the initial step size.
C The error weights in EWT are inverted after being loaded.
C-----------------------------------------------------------------------
 100  CONTINUE
      LYH = LYHN
      IWORK(22) = LYH
      TN = T
      NST = 0
      H = 1.0D0
      NNZ = 0
      NGP = 0
      NZL = 0
      NZU = 0
C Load the initial value vector in YH. --------------------------------
      DO 105 I = 1,N
 105    RWORK(I+LYH-1) = Y(I)
C Initial call to F.  (LF0 points to YH(*,2).) ------------------------
      LF0 = LYH + NYH
      CALL F (NEQ, T, Y, RWORK(LF0))
      NFE = 1
C Load and invert the EWT array.  (H is temporarily set to 1.0.) -------
      CALL DEWSET (N, ITOL, RTOL, ATOL, RWORK(LYH), RWORK(LEWT))
      DO 110 I = 1,N
        IF (RWORK(I+LEWT-1) .LE. 0.0D0) GO TO 621
 110    RWORK(I+LEWT-1) = 1.0D0/RWORK(I+LEWT-1)
      IF (MITER .EQ. 0 .OR. MITER .EQ. 3) GO TO 120
C DIPREP and DPREP do sparse matrix preprocessing if MITER = 1 or 2. ---
      LACOR = MIN(LACOR,LRW)
      CALL DIPREP (NEQ, Y, RWORK, IWORK(LIA),IWORK(LJA), IPFLAG, F, JAC)
      LENRW = LWM - 1 + LENWK + LREST
      IWORK(17) = LENRW
      IF (IPFLAG .NE. -1) IWORK(23) = IPIAN
      IF (IPFLAG .NE. -1) IWORK(24) = IPJAN
      IPGO = -IPFLAG + 1
      GO TO (115, 628, 629, 630, 631, 632, 633), IPGO
 115  IWORK(22) = LYH
      IF (LENRW .GT. LRW) GO TO 617
C Check TCRIT for legality (ITASK = 4 or 5). --------------------------
 120  CONTINUE
      IF (ITASK .NE. 4 .AND. ITASK .NE. 5) GO TO 125
      TCRIT = RWORK(1)
      IF ((TCRIT - TOUT)*(TOUT - T) .LT. 0.0D0) GO TO 625
      IF (H0 .NE. 0.0D0 .AND. (T + H0 - TCRIT)*H0 .GT. 0.0D0)
     1   H0 = TCRIT - T
C Initialize all remaining parameters. --------------------------------
 125  UROUND = DUMACH()
      JSTART = 0
      IF (MITER .NE. 0) RWORK(LWM) = SQRT(UROUND)
      MSBJ = 50
      NSLJ = 0
      CCMXJ = 0.2D0
      PSMALL = 1000.0D0*UROUND
      RBIG = 0.01D0/PSMALL
      NHNIL = 0
      NJE = 0
      NLU = 0
      NSLAST = 0
      HU = 0.0D0
      NQU = 0
      CCMAX = 0.3D0
      MAXCOR = 3
      MSBP = 20
      MXNCF = 10
C-----------------------------------------------------------------------
C The coding below computes the step size, H0, to be attempted on the
C first step, unless the user has supplied a value for this.
C First check that TOUT - T differs significantly from zero.
C A scalar tolerance quantity TOL is computed, as MAX(RTOL(i))
C if this is positive, or MAX(ATOL(i)/ABS(Y(i))) otherwise, adjusted
C so as to be between 100*UROUND and 1.0E-3.
C Then the computed value H0 is given by..
C                                      NEQ
C   H0**2 = TOL / ( w0**-2 + (1/NEQ) * Sum ( f(i)/ywt(i) )**2  )
C                                       1
C where   w0     = MAX ( ABS(T), ABS(TOUT) ),
C         f(i)   = i-th component of initial value of f,
C         ywt(i) = EWT(i)/TOL  (a weight for y(i)).
C The sign of H0 is inferred from the initial values of TOUT and T.
C ABS(H0) is made .le. ABS(TOUT-T) in any case.
C-----------------------------------------------------------------------
      LF0 = LYH + NYH
      IF (H0 .NE. 0.0D0) GO TO 180
      TDIST = ABS(TOUT - T)
      W0 = MAX(ABS(T),ABS(TOUT))
      IF (TDIST .LT. 2.0D0*UROUND*W0) GO TO 622
      TOL = RTOL(1)
      IF (ITOL .LE. 2) GO TO 140
      DO 130 I = 1,N
 130    TOL = MAX(TOL,RTOL(I))
 140  IF (TOL .GT. 0.0D0) GO TO 160
      ATOLI = ATOL(1)
      DO 150 I = 1,N
        IF (ITOL .EQ. 2 .OR. ITOL .EQ. 4) ATOLI = ATOL(I)
        AYI = ABS(Y(I))
        IF (AYI .NE. 0.0D0) TOL = MAX(TOL,ATOLI/AYI)
 150    CONTINUE
 160  TOL = MAX(TOL,100.0D0*UROUND)
      TOL = MIN(TOL,0.001D0)
      SUM = DVNORM (N, RWORK(LF0), RWORK(LEWT))
      SUM = 1.0D0/(TOL*W0*W0) + TOL*SUM**2
      H0 = 1.0D0/SQRT(SUM)
      H0 = MIN(H0,TDIST)
      H0 = SIGN(H0,TOUT-T)
C Adjust H0 if necessary to meet HMAX bound. --------------------------
 180  RH = ABS(H0)*HMXI
      IF (RH .GT. 1.0D0) H0 = H0/RH
C Load H with H0 and scale YH(*,2) by H0. -----------------------------
      H = H0
      DO 190 I = 1,N
 190    RWORK(I+LF0-1) = H0*RWORK(I+LF0-1)
      GO TO 270
C-----------------------------------------------------------------------
C Block D.
C The next code block is for continuation calls only (ISTATE = 2 or 3)
C and is to check stop conditions before taking a step.
C-----------------------------------------------------------------------
 200  NSLAST = NST
      GO TO (210, 250, 220, 230, 240), ITASK
 210  IF ((TN - TOUT)*H .LT. 0.0D0) GO TO 250
      CALL DINTDY (TOUT, 0, RWORK(LYH), NYH, Y, IFLAG)
      IF (IFLAG .NE. 0) GO TO 627
      T = TOUT
      GO TO 420
 220  TP = TN - HU*(1.0D0 + 100.0D0*UROUND)
      IF ((TP - TOUT)*H .GT. 0.0D0) GO TO 623
      IF ((TN - TOUT)*H .LT. 0.0D0) GO TO 250
      GO TO 400
 230  TCRIT = RWORK(1)
      IF ((TN - TCRIT)*H .GT. 0.0D0) GO TO 624
      IF ((TCRIT - TOUT)*H .LT. 0.0D0) GO TO 625
      IF ((TN - TOUT)*H .LT. 0.0D0) GO TO 245
      CALL DINTDY (TOUT, 0, RWORK(LYH), NYH, Y, IFLAG)
      IF (IFLAG .NE. 0) GO TO 627
      T = TOUT
      GO TO 420
 240  TCRIT = RWORK(1)
      IF ((TN - TCRIT)*H .GT. 0.0D0) GO TO 624
 245  HMX = ABS(TN) + ABS(H)
      IHIT = ABS(TN - TCRIT) .LE. (100.0D0*UROUND*HMX)
      IF (IHIT) GO TO 400
      TNEXT = TN + H*(1.0D0 + 4.0D0*UROUND)
      IF ((TNEXT - TCRIT)*H .LE. 0.0D0) GO TO 250
      H = (TCRIT - TN)*(1.0D0 - 4.0D0*UROUND)
      IF (ISTATE .EQ. 2) JSTART = -2
C-----------------------------------------------------------------------
C Block E.
C The next block is normally executed for all calls and contains
C the call to the one-step core integrator DSTODE.
C
C This is a looping point for the integration steps.
C
C First check for too many steps being taken, update EWT (if not at
C start of problem), check for too much accuracy being requested, and
C check for H below the roundoff level in T.
C-----------------------------------------------------------------------
 250  CONTINUE
      IF ((NST-NSLAST) .GE. MXSTEP) GO TO 500
      CALL DEWSET (N, ITOL, RTOL, ATOL, RWORK(LYH), RWORK(LEWT))
      DO 260 I = 1,N
        IF (RWORK(I+LEWT-1) .LE. 0.0D0) GO TO 510
 260    RWORK(I+LEWT-1) = 1.0D0/RWORK(I+LEWT-1)
 270  TOLSF = UROUND*DVNORM (N, RWORK(LYH), RWORK(LEWT))
      IF (TOLSF .LE. 1.0D0) GO TO 280
      TOLSF = TOLSF*2.0D0
      IF (NST .EQ. 0) GO TO 626
      GO TO 520
 280  IF ((TN + H) .NE. TN) GO TO 290
      NHNIL = NHNIL + 1
      IF (NHNIL .GT. MXHNIL) GO TO 290
      MSG = 'DLSODES- Warning..Internal T (=R1) and H (=R2) are'
      CALL XERRWD (MSG, 50, 101, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG='      such that in the machine, T + H = T on the next step  '
      CALL XERRWD (MSG, 60, 101, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '     (H = step size). Solver will continue anyway.'
      CALL XERRWD (MSG, 50, 101, 0, 0, 0, 0, 2, TN, H)
      IF (NHNIL .LT. MXHNIL) GO TO 290
      MSG = 'DLSODES- Above warning has been issued I1 times.  '
      CALL XERRWD (MSG, 50, 102, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '     It will not be issued again for this problem.'
      CALL XERRWD (MSG, 50, 102, 0, 1, MXHNIL, 0, 0, 0.0D0, 0.0D0)
 290  CONTINUE
C-----------------------------------------------------------------------
C    CALL DSTODE(NEQ,Y,YH,NYH,YH,EWT,SAVF,ACOR,WM,WM,F,JAC,DPRJS,DSOLSS)
C-----------------------------------------------------------------------
      CALL DSTODE (NEQ, Y, RWORK(LYH), NYH, RWORK(LYH), RWORK(LEWT),
     1   RWORK(LSAVF), RWORK(LACOR), RWORK(LWM), RWORK(LWM),
     2   F, JAC, DPRJS, DSOLSS)
      KGO = 1 - KFLAG
      GO TO (300, 530, 540, 550), KGO
C-----------------------------------------------------------------------
C Block F.
C The following block handles the case of a successful return from the
C core integrator (KFLAG = 0).  Test for stop conditions.
C-----------------------------------------------------------------------
 300  INIT = 1
      GO TO (310, 400, 330, 340, 350), ITASK
C ITASK = 1.  if TOUT has been reached, interpolate. -------------------
 310  IF ((TN - TOUT)*H .LT. 0.0D0) GO TO 250
      CALL DINTDY (TOUT, 0, RWORK(LYH), NYH, Y, IFLAG)
      T = TOUT
      GO TO 420
C ITASK = 3.  Jump to exit if TOUT was reached. ------------------------
 330  IF ((TN - TOUT)*H .GE. 0.0D0) GO TO 400
      GO TO 250
C ITASK = 4.  See if TOUT or TCRIT was reached.  Adjust H if necessary.
 340  IF ((TN - TOUT)*H .LT. 0.0D0) GO TO 345
      CALL DINTDY (TOUT, 0, RWORK(LYH), NYH, Y, IFLAG)
      T = TOUT
      GO TO 420
 345  HMX = ABS(TN) + ABS(H)
      IHIT = ABS(TN - TCRIT) .LE. (100.0D0*UROUND*HMX)
      IF (IHIT) GO TO 400
      TNEXT = TN + H*(1.0D0 + 4.0D0*UROUND)
      IF ((TNEXT - TCRIT)*H .LE. 0.0D0) GO TO 250
      H = (TCRIT - TN)*(1.0D0 - 4.0D0*UROUND)
      JSTART = -2
      GO TO 250
C ITASK = 5.  See if TCRIT was reached and jump to exit. ---------------
 350  HMX = ABS(TN) + ABS(H)
      IHIT = ABS(TN - TCRIT) .LE. (100.0D0*UROUND*HMX)
C-----------------------------------------------------------------------
C Block G.
C The following block handles all successful returns from DLSODES.
C If ITASK .ne. 1, Y is loaded from YH and T is set accordingly.
```

```
C ISTATE is set to 2, and the optional outputs are loaded into the
C work arrays before returning.
C-----------------------------------------------------------------------
 400  DO 410 I = 1,N
 410    Y(I) = RWORK(I+LYH-1)
      T = TN
      IF (ITASK .NE. 4 .AND. ITASK .NE. 5) GO TO 420
      IF (IHIT) T = TCRIT
 420  ISTATE = 2
      RWORK(11) = HU
      RWORK(12) = H
      RWORK(13) = TN
      IWORK(11) = NST
      IWORK(12) = NFE
      IWORK(13) = NJE
      IWORK(14) = NQU
      IWORK(15) = NQ
      IWORK(19) = NNZ
      IWORK(20) = NGP
      IWORK(21) = NLU
      IWORK(25) = NZL
      IWORK(26) = NZU
      RETURN
C-----------------------------------------------------------------------
C Block H.
C The following block handles all unsuccessful returns other than
C those for illegal input.  First the error message routine is called.
C If there was an error test or convergence test failure, IMXER is set.
C Then Y is loaded from YH and T is set to TN.
C The optional outputs are loaded into the work arrays before returning.
C-----------------------------------------------------------------------
C The maximum number of steps was taken before reaching TOUT. ----------
 500  MSG = 'DLSODES- At current T (=R1), MXSTEP (=I1) steps   '
      CALL XERRWD (MSG, 50, 201, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '      taken on this call before reaching TOUT     '
      CALL XERRWD (MSG, 50, 201, 0, 1, MXSTEP, 0, 1, TN, 0.0D0)
      ISTATE = -1
      GO TO 580
C EWT(i) .le. 0.0 for some i (not at start of problem). ----------------
 510  EWTI = RWORK(LEWT+I-1)
      MSG = 'DLSODES- At T (=R1), EWT(I1) has become R2 .le. 0.'
      CALL XERRWD (MSG, 50, 202, 0, 1, I, 0, 2, TN, EWTI)
      ISTATE = -6
      GO TO 580
C Too much accuracy requested for machine precision. -------------------
 520  MSG = 'DLSODES- At T (=R1), too much accuracy requested  '
      CALL XERRWD (MSG, 50, 203, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '      for precision of machine..  See TOLSF (=R2) '
      CALL XERRWD (MSG, 50, 203, 0, 0, 0, 0, 2, TN, TOLSF)
      RWORK(14) = TOLSF
      ISTATE = -2
      GO TO 580
C KFLAG = -1.  Error test failed repeatedly or with ABS(H) = HMIN. -----
 530  MSG = 'DLSODES- At T(=R1) and step size H(=R2), the error'
      CALL XERRWD (MSG, 50, 204, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '      test failed repeatedly or with ABS(H) = HMIN'
      CALL XERRWD (MSG, 50, 204, 0, 0, 0, 0, 2, TN, H)
      ISTATE = -4
      GO TO 560
C KFLAG = -2.  Convergence failed repeatedly or with ABS(H) = HMIN. ----
 540  MSG = 'DLSODES- At T (=R1) and step size H (=R2), the    '
      CALL XERRWD (MSG, 50, 205, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '      corrector convergence failed repeatedly     '
      CALL XERRWD (MSG, 50, 205, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '      or with ABS(H) = HMIN   '
      CALL XERRWD (MSG, 30, 205, 0, 0, 0, 0, 2, TN, H)
      ISTATE = -5
      GO TO 560
C KFLAG = -3.  Fatal error flag returned by DPRJS or DSOLSS (CDRV). ----
 550  MSG = 'DLSODES- At T (=R1) and step size H (=R2), a fatal'
      CALL XERRWD (MSG, 50, 207, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '      error flag was returned by CDRV (by way of  '
      CALL XERRWD (MSG, 50, 207, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '      Subroutine DPRJS or DSOLSS)       '
      CALL XERRWD (MSG, 40, 207, 0, 0, 0, 0, 2, TN, H)
      ISTATE = -7
      GO TO 580
C Compute IMXER if relevant. -----------------------------------------
 560  BIG = 0.0D0
      IMXER = 1
      DO 570 I = 1,N
        SIZE = ABS(RWORK(I+LACOR-1)*RWORK(I+LEWT-1))
        IF (BIG .GE. SIZE) GO TO 570
        BIG = SIZE
        IMXER = I
 570    CONTINUE
      IWORK(16) = IMXER
C Set Y vector, T, and optional outputs. -----------------------------
 580  DO 590 I = 1,N
 590    Y(I) = RWORK(I+LYH-1)
      T = TN
      RWORK(11) = HU
      RWORK(12) = H
      RWORK(13) = TN
      IWORK(11) = NST
      IWORK(12) = NFE
      IWORK(13) = NJE
      IWORK(14) = NQU
      IWORK(15) = NQ
      IWORK(19) = NNZ
      IWORK(20) = NGP
      IWORK(21) = NLU
      IWORK(25) = NZL
      IWORK(26) = NZU
      RETURN
C-----------------------------------------------------------------------
C Block I.
C The following block handles all error returns due to illegal input
C (ISTATE = -3), as detected before calling the core integrator.
C First the error message routine is called.  If the illegal input
C is a negative ISTATE, the run is aborted (apparent infinite loop).
C-----------------------------------------------------------------------
 601  MSG = 'DLSODES- ISTATE (=I1) illegal.'
      CALL XERRWD (MSG, 30, 1, 0, 1, ISTATE, 0, 0, 0.0D0, 0.0D0)
      IF (ISTATE .LT. 0) GO TO 800
      GO TO 700
 602  MSG = 'DLSODES- ITASK (=I1) illegal. '
      CALL XERRWD (MSG, 30, 2, 0, 1, ITASK, 0, 0, 0.0D0, 0.0D0)
      GO TO 700
 603  MSG = 'DLSODES- ISTATE.gt.1 but DLSODES not initialized. '
      CALL XERRWD (MSG, 50, 3, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      GO TO 700
 604  MSG = 'DLSODES- NEQ (=I1) .lt. 1     '
      CALL XERRWD (MSG, 30, 4, 0, 1, NEQ(1), 0, 0, 0.0D0, 0.0D0)
      GO TO 700
 605  MSG = 'DLSODES- ISTATE = 3 and NEQ increased (I1 to I2). '
      CALL XERRWD (MSG, 50, 5, 0, 2, N, NEQ(1), 0, 0.0D0, 0.0D0)
      GO TO 700
```

```
606  MSG = 'DLSODES- ITOL (=I1) illegal.  '
     CALL XERRWD (MSG, 30, 6, 0, 1, ITOL, 0, 0, 0.0D0, 0.0D0)
     GO TO 700
607  MSG = 'DLSODES- IOPT (=I1) illegal.  '
     CALL XERRWD (MSG, 30, 7, 0, 1, IOPT, 0, 0, 0.0D0, 0.0D0)
     GO TO 700
608  MSG = 'DLSODES- MF (=I1) illegal.    '
     CALL XERRWD (MSG, 30, 8, 0, 1, MF, 0, 0, 0.0D0, 0.0D0)
     GO TO 700
609  MSG = 'DLSODES- SETH (=R1) .lt. 0.0  '
     CALL XERRWD (MSG, 30, 9, 0, 0, 0, 0, 1, SETH, 0.0D0)
     GO TO 700
611  MSG = 'DLSODES- MAXORD (=I1) .lt. 0  '
     CALL XERRWD (MSG, 30, 11, 0, 1, MAXORD, 0, 0, 0.0D0, 0.0D0)
     GO TO 700
612  MSG = 'DLSODES- MXSTEP (=I1) .lt. 0  '
     CALL XERRWD (MSG, 30, 12, 0, 1, MXSTEP, 0, 0, 0.0D0, 0.0D0)
     GO TO 700
613  MSG = 'DLSODES- MXHNIL (=I1) .lt. 0  '
     CALL XERRWD (MSG, 30, 13, 0, 1, MXHNIL, 0, 0, 0.0D0, 0.0D0)
     GO TO 700
614  MSG = 'DLSODES- TOUT (=R1) behind T (=R2)      '
     CALL XERRWD (MSG, 40, 14, 0, 0, 0, 0, 2, TOUT, T)
     MSG = '      Integration direction is given by H0 (=R1)  '
     CALL XERRWD (MSG, 50, 14, 0, 0, 0, 0, 1, H0, 0.0D0)
     GO TO 700
615  MSG = 'DLSODES- HMAX (=R1) .lt. 0.0  '
     CALL XERRWD (MSG, 30, 15, 0, 0, 0, 0, 1, HMAX, 0.0D0)
     GO TO 700
616  MSG = 'DLSODES- HMIN (=R1) .lt. 0.0  '
     CALL XERRWD (MSG, 30, 16, 0, 0, 0, 0, 1, HMIN, 0.0D0)
     GO TO 700
617  MSG = 'DLSODES- RWORK length is insufficient to proceed. '
     CALL XERRWD (MSG, 50, 17, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
     MSG='        Length needed is .ge. LENRW (=I1), exceeds LRW (=I2)'
     CALL XERRWD (MSG, 60, 17, 0, 2, LENRW, LRW, 0, 0.0D0, 0.0D0)
     GO TO 700
618  MSG = 'DLSODES- IWORK length is insufficient to proceed. '
     CALL XERRWD (MSG, 50, 18, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
     MSG='        Length needed is .ge. LENIW (=I1), exceeds LIW (=I2)'
     CALL XERRWD (MSG, 60, 18, 0, 2, LENIW, LIW, 0, 0.0D0, 0.0D0)
     GO TO 700
619  MSG = 'DLSODES- RTOL(I1) is R1 .lt. 0.0        '
     CALL XERRWD (MSG, 40, 19, 0, 1, I, 0, 1, RTOLI, 0.0D0)
     GO TO 700
620  MSG = 'DLSODES- ATOL(I1) is R1 .lt. 0.0        '
     CALL XERRWD (MSG, 40, 20, 0, 1, I, 0, 1, ATOLI, 0.0D0)
     GO TO 700
621  EWTI = RWORK(LEWT+I-1)
     MSG = 'DLSODES- EWT(I1) is R1 .le. 0.0         '
     CALL XERRWD (MSG, 40, 21, 0, 1, I, 0, 1, EWTI, 0.0D0)
     GO TO 700
622  MSG='DLSODES- TOUT(=R1) too close to T(=R2) to start integration.'
     CALL XERRWD (MSG, 60, 22, 0, 0, 0, 0, 2, TOUT, T)
     GO TO 700
623  MSG='DLSODES- ITASK = I1 and TOUT (=R1) behind TCUR - HU (= R2)  '
     CALL XERRWD (MSG, 60, 23, 0, 1, ITASK, 0, 2, TOUT, TP)
     GO TO 700
624  MSG='DLSODES- ITASK = 4 or 5 and TCRIT (=R1) behind TCUR (=R2)   '
     CALL XERRWD (MSG, 60, 24, 0, 0, 0, 0, 2, TCRIT, TN)
     GO TO 700
625  MSG='DLSODES- ITASK = 4 or 5 and TCRIT (=R1) behind TOUT (=R2)   '
     CALL XERRWD (MSG, 60, 25, 0, 0, 0, 0, 2, TCRIT, TOUT)
     GO TO 700
```

```
 626  MSG = 'DLSODES- At start of problem, too much accuracy   '
      CALL XERRWD (MSG, 50, 26, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG='      requested for precision of machine..  See TOLSF (=R1) '
      CALL XERRWD (MSG, 60, 26, 0, 0, 0, 0, 1, TOLSF, 0.0D0)
      RWORK(14) = TOLSF
      GO TO 700
 627  MSG = 'DLSODES- Trouble in DINTDY.  ITASK = I1, TOUT = R1'
      CALL XERRWD (MSG, 50, 27, 0, 1, ITASK, 0, 1, TOUT, 0.0D0)
      GO TO 700
 628  MSG='DLSODES- RWORK length insufficient (for Subroutine DPREP).  '
      CALL XERRWD (MSG, 60, 28, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG='        Length needed is .ge. LENRW (=I1), exceeds LRW (=I2)'
      CALL XERRWD (MSG, 60, 28, 0, 2, LENRW, LRW, 0, 0.0D0, 0.0D0)
      GO TO 700
 629  MSG='DLSODES- RWORK length insufficient (for Subroutine JGROUP). '
      CALL XERRWD (MSG, 60, 29, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG='        Length needed is .ge. LENRW (=I1), exceeds LRW (=I2)'
      CALL XERRWD (MSG, 60, 29, 0, 2, LENRW, LRW, 0, 0.0D0, 0.0D0)
      GO TO 700
 630  MSG='DLSODES- RWORK length insufficient (for Subroutine ODRV).   '
      CALL XERRWD (MSG, 60, 30, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG='        Length needed is .ge. LENRW (=I1), exceeds LRW (=I2)'
      CALL XERRWD (MSG, 60, 30, 0, 2, LENRW, LRW, 0, 0.0D0, 0.0D0)
      GO TO 700
 631  MSG='DLSODES- Error from ODRV in Yale Sparse Matrix Package.     '
      CALL XERRWD (MSG, 60, 31, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      IMUL = (IYS - 1)/N
      IREM = IYS - IMUL*N
      MSG='      At T (=R1), ODRV returned error flag = I1*NEQ + I2.   '
      CALL XERRWD (MSG, 60, 31, 0, 2, IMUL, IREM, 1, TN, 0.0D0)
      GO TO 700
 632  MSG='DLSODES- RWORK length insufficient (for Subroutine CDRV).   '
      CALL XERRWD (MSG, 60, 32, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG='        Length needed is .ge. LENRW (=I1), exceeds LRW (=I2)'
      CALL XERRWD (MSG, 60, 32, 0, 2, LENRW, LRW, 0, 0.0D0, 0.0D0)
      GO TO 700
 633  MSG='DLSODES- Error from CDRV in Yale Sparse Matrix Package.     '
      CALL XERRWD (MSG, 60, 33, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      IMUL = (IYS - 1)/N
      IREM = IYS - IMUL*N
      MSG='      At T (=R1), CDRV returned error flag = I1*NEQ + I2.   '
      CALL XERRWD (MSG, 60, 33, 0, 2, IMUL, IREM, 1, TN, 0.0D0)
      IF (IMUL .EQ. 2) THEN
      MSG='        Duplicate entry in sparsity structure descriptors.  '
      CALL XERRWD (MSG, 60, 33, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      ENDIF
      IF (IMUL .EQ. 3 .OR. IMUL .EQ. 6) THEN
      MSG='        Insufficient storage for NSFC (called by CDRV).     '
      CALL XERRWD (MSG, 60, 33, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      ENDIF
C
 700  ISTATE = -3
      RETURN
C
 800  MSG = 'DLSODES- Run aborted.. apparent infinite loop.    '
      CALL XERRWD (MSG, 50, 303, 2, 0, 0, 0, 0, 0.0D0, 0.0D0)
      RETURN
C----------------------- End of Subroutine DLSODES ---------------------
      END
*DECK DIPREP
      SUBROUTINE DIPREP (NEQ, Y, RWORK, IA, JA, IPFLAG, F, JAC)
      EXTERNAL F, JAC
      INTEGER NEQ, IA, JA, IPFLAG
      DOUBLE PRECISION Y, RWORK
      DIMENSION NEQ(*), Y(*), RWORK(*), IA(*), JA(*)
      INTEGER IOWND, IOWNS,
     1   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L,
     2   LYH, LEWT, LACOR, LSAVF, LWM, LIWM, METH, MITER,
     3   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      INTEGER IPLOST, IESP, ISTATC, IYS, IBA, IBIAN, IBJAN, IBJGP,
     1   IPIAN, IPJAN, IPJGP, IPIGP, IPR, IPC, IPIC, IPISP, IPRSP, IPA,
     2   LENYH, LENYHM, LENWK, LREQ, LRAT, LREST, LWMIN, MOSS, MSBJ,
     3   NSLJ, NGP, NLU, NNZ, NSP, NZL, NZU
      DOUBLE PRECISION ROWNS,
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND
      DOUBLE PRECISION RLSS
      COMMON /DLS001/ ROWNS(209),
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND,
     2   IOWND(6), IOWNS(6),
     3   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L,
     4   LYH, LEWT, LACOR, LSAVF, LWM, LIWM, METH, MITER,
     5   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      COMMON /DLSS01/ RLSS(6),
     1   IPLOST, IESP, ISTATC, IYS, IBA, IBIAN, IBJAN, IBJGP,
     2   IPIAN, IPJAN, IPJGP, IPIGP, IPR, IPC, IPIC, IPISP, IPRSP, IPA,
     3   LENYH, LENYHM, LENWK, LREQ, LRAT, LREST, LWMIN, MOSS, MSBJ,
     4   NSLJ, NGP, NLU, NNZ, NSP, NZL, NZU
      INTEGER I, IMAX, LEWTN, LYHD, LYHN
C-----------------------------------------------------------------------
C This routine serves as an interface between the driver and
C Subroutine DPREP.  It is called only if MITER is 1 or 2.
C Tasks performed here are:
C  * call DPREP,
C  * reset the required WM segment length LENWK,
C  * move YH back to its final location (following WM in RWORK),
C  * reset pointers for YH, SAVF, EWT, and ACOR, and
C  * move EWT to its new position if ISTATE = 1.
C IPFLAG is an output error indication flag.  IPFLAG = 0 if there was
C no trouble, and IPFLAG is the value of the DPREP error flag IPPER
C if there was trouble in Subroutine DPREP.
C-----------------------------------------------------------------------
      IPFLAG = 0
C Call DPREP to do matrix preprocessing operations. --------------------
      CALL DPREP (NEQ, Y, RWORK(LYH), RWORK(LSAVF), RWORK(LEWT),
     1   RWORK(LACOR), IA, JA, RWORK(LWM), RWORK(LWM), IPFLAG, F, JAC)
      LENWK = MAX(LREQ,LWMIN)
      IF (IPFLAG .LT. 0) RETURN
C If DPREP was successful, move YH to end of required space for WM. ----
      LYHN = LWM + LENWK
      IF (LYHN .GT. LYH) RETURN
      LYHD = LYH - LYHN
      IF (LYHD .EQ. 0) GO TO 20
      IMAX = LYHN - 1 + LENYHM
      DO 10 I = LYHN,IMAX
 10     RWORK(I) = RWORK(I+LYHD)
      LYH = LYHN
C Reset pointers for SAVF, EWT, and ACOR. ------------------------------
 20   LSAVF = LYH + LENYH
      LEWTN = LSAVF + N
      LACOR = LEWTN + N
      IF (ISTATC .EQ. 3) GO TO 40
C If ISTATE = 1, move EWT (left) to its new position. ------------------
      IF (LEWTN .GT. LEWT) RETURN
      DO 30 I = 1,N
 30     RWORK(I+LEWTN-1) = RWORK(I+LEWT-1)
 40   LEWT = LEWTN
      RETURN
C----------------------- End of Subroutine DIPREP ----------------------
```

```
      END
*DECK DPREP
      SUBROUTINE DPREP (NEQ, Y, YH, SAVF, EWT, FTEM, IA, JA,
     1                     WK, IWK, IPPER, F, JAC)
      EXTERNAL F,JAC
      INTEGER NEQ, IA, JA, IWK, IPPER
      DOUBLE PRECISION Y, YH, SAVF, EWT, FTEM, WK
      DIMENSION NEQ(*), Y(*), YH(*), SAVF(*), EWT(*), FTEM(*),
     1   IA(*), JA(*), WK(*), IWK(*)
      INTEGER IOWND, IOWNS,
     1   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L,
     2   LYH, LEWT, LACOR, LSAVF, LWM, LIWM, METH, MITER,
     3   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      INTEGER IPLOST, IESP, ISTATC, IYS, IBA, IBIAN, IBJAN, IBJGP,
     1   IPIAN, IPJAN, IPJGP, IPIGP, IPR, IPC, IPIC, IPISP, IPRSP, IPA,
     2   LENYH, LENYHM, LENWK, LREQ, LRAT, LREST, LWMIN, MOSS, MSBJ,
     3   NSLJ, NGP, NLU, NNZ, NSP, NZL, NZU
      DOUBLE PRECISION ROWNS,
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND
      DOUBLE PRECISION CON0, CONMIN, CCMXJ, PSMALL, RBIG, SETH
      COMMON /DLS001/ ROWNS(209),
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND,
     2   IOWND(6), IOWNS(6),
     3   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L,
     4   LYH, LEWT, LACOR, LSAVF, LWM, LIWM, METH, MITER,
     5   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      COMMON /DLSS01/ CON0, CONMIN, CCMXJ, PSMALL, RBIG, SETH,
     1   IPLOST, IESP, ISTATC, IYS, IBA, IBIAN, IBJAN, IBJGP,
     2   IPIAN, IPJAN, IPJGP, IPIGP, IPR, IPC, IPIC, IPISP, IPRSP, IPA,
     3   LENYH, LENYHM, LENWK, LREQ, LRAT, LREST, LWMIN, MOSS, MSBJ,
     4   NSLJ, NGP, NLU, NNZ, NSP, NZL, NZU
      INTEGER I, IBR, IER, IPIL, IPIU, IPTT1, IPTT2, J, JFOUND, K,
     1   KNEW, KMAX, KMIN, LDIF, LENIGP, LIWK, MAXG, NP1, NZSUT
      DOUBLE PRECISION DQ, DYJ, ERWT, FAC, YJ
C-----------------------------------------------------------------------
C This routine performs preprocessing related to the sparse linear
C systems that must be solved if MITER = 1 or 2.
C The operations that are performed here are:
C  * compute sparseness structure of Jacobian according to MOSS,
C  * compute grouping of column indices (MITER = 2),
C  * compute a new ordering of rows and columns of the matrix,
C  * reorder JA corresponding to the new ordering,
C  * perform a symbolic LU factorization of the matrix, and
C  * set pointers for segments of the IWK/WK array.
C In addition to variables described previously, DPREP uses the
C following for communication:
C YH     = the history array.  Only the first column, containing the
C          current Y vector, is used.  Used only if MOSS .ne. 0.
C SAVF   = a work array of length NEQ, used only if MOSS .ne. 0.
C EWT    = array of length NEQ containing (inverted) error weights.
C          Used only if MOSS = 2 or if ISTATE = MOSS = 1.
C FTEM   = a work array of length NEQ, identical to ACOR in the driver,
C          used only if MOSS = 2.
C WK     = a real work array of length LENWK, identical to WM in
C          the driver.
C IWK    = integer work array, assumed to occupy the same space as WK.
C LENWK  = the length of the work arrays WK and IWK.
C ISTATC = a copy of the driver input argument ISTATE (= 1 on the
C          first call, = 3 on a continuation call).
C IYS    = flag value from ODRV or CDRV.
C IPPER  = output error flag with the following values and meanings:
C          0  no error.
C         -1  insufficient storage for internal structure pointers.
C         -2  insufficient storage for JGROUP.
```

```
C         -3  insufficient storage for ODRV.
C         -4  other error flag from ODRV (should never occur).
C         -5  insufficient storage for CDRV.
C         -6  other error flag from CDRV.
C-----------------------------------------------------------------------
      IBIAN = LRAT*2
      IPIAN = IBIAN + 1
      NP1 = N + 1
      IPJAN = IPIAN + NP1
      IBJAN = IPJAN - 1
      LIWK = LENWK*LRAT
      IF (IPJAN+N-1 .GT. LIWK) GO TO 210
      IF (MOSS .EQ. 0) GO TO 30
C
      IF (ISTATC .EQ. 3) GO TO 20
C ISTATE = 1 and MOSS .ne. 0.  Perturb Y for structure determination. --
      DO 10 I = 1,N
        ERWT = 1.0D0/EWT(I)
        FAC = 1.0D0 + 1.0D0/(I + 1.0D0)
        Y(I) = Y(I) + FAC*SIGN(ERWT,Y(I))
 10     CONTINUE
      GO TO (70, 100), MOSS
C
 20   CONTINUE
C ISTATE = 3 and MOSS .ne. 0.  Load Y from YH(*,1). --------------------
      DO 25 I = 1,N
 25     Y(I) = YH(I)
      GO TO (70, 100), MOSS
C
C MOSS = 0.  Process user's IA,JA.  Add diagonal entries if necessary. -
 30   KNEW = IPJAN
      KMIN = IA(1)
      IWK(IPIAN) = 1
      DO 60 J = 1,N
        JFOUND = 0
        KMAX = IA(J+1) - 1
        IF (KMIN .GT. KMAX) GO TO 45
        DO 40 K = KMIN,KMAX
          I = JA(K)
          IF (I .EQ. J) JFOUND = 1
          IF (KNEW .GT. LIWK) GO TO 210
          IWK(KNEW) = I
          KNEW = KNEW + 1
 40       CONTINUE
        IF (JFOUND .EQ. 1) GO TO 50
 45     IF (KNEW .GT. LIWK) GO TO 210
        IWK(KNEW) = J
        KNEW = KNEW + 1
 50     IWK(IPIAN+J) = KNEW + 1 - IPJAN
        KMIN = KMAX + 1
 60     CONTINUE
      GO TO 140
C
C MOSS = 1.  Compute structure from user-supplied Jacobian routine JAC.
 70   CONTINUE
C A dummy call to F allows user to create temporaries for use in JAC. --
      CALL F (NEQ, TN, Y, SAVF)
      K = IPJAN
      IWK(IPIAN) = 1
      DO 90 J = 1,N
        IF (K .GT. LIWK) GO TO 210
        IWK(K) = J
        K = K + 1
        DO 75 I = 1,N
 75       SAVF(I) = 0.0D0
        CALL JAC (NEQ, TN, Y, J, IWK(IPIAN), IWK(IPJAN), SAVF)
        DO 80 I = 1,N
          IF (ABS(SAVF(I)) .LE. SETH) GO TO 80
          IF (I .EQ. J) GO TO 80
          IF (K .GT. LIWK) GO TO 210
          IWK(K) = I
          K = K + 1
 80       CONTINUE
        IWK(IPIAN+J) = K + 1 - IPJAN
 90     CONTINUE
      GO TO 140
C
C MOSS = 2.  Compute structure from results of N + 1 calls to F. -------
 100  K = IPJAN
      IWK(IPIAN) = 1
      CALL F (NEQ, TN, Y, SAVF)
      DO 120 J = 1,N
        IF (K .GT. LIWK) GO TO 210
        IWK(K) = J
        K = K + 1
        YJ = Y(J)
        ERWT = 1.0D0/EWT(J)
        DYJ = SIGN(ERWT,YJ)
        Y(J) = YJ + DYJ
        CALL F (NEQ, TN, Y, FTEM)
        Y(J) = YJ
        DO 110 I = 1,N
          DQ = (FTEM(I) - SAVF(I))/DYJ
          IF (ABS(DQ) .LE. SETH) GO TO 110
          IF (I .EQ. J) GO TO 110
          IF (K .GT. LIWK) GO TO 210
          IWK(K) = I
          K = K + 1
 110      CONTINUE
        IWK(IPIAN+J) = K + 1 - IPJAN
 120    CONTINUE
C
 140  CONTINUE
      IF (MOSS .EQ. 0 .OR. ISTATC .NE. 1) GO TO 150
C If ISTATE = 1 and MOSS .ne. 0, restore Y from YH. --------------------
      DO 145 I = 1,N
 145    Y(I) = YH(I)
 150  NNZ = IWK(IPIAN+N) - 1
      LENIGP = 0
      IPIGP = IPJAN + NNZ
      IF (MITER .NE. 2) GO TO 160
C
C Compute grouping of column indices (MITER = 2). ----------------------
      MAXG = NP1
      IPJGP = IPJAN + NNZ
      IBJGP = IPJGP - 1
      IPIGP = IPJGP + N
      IPTT1 = IPIGP + NP1
      IPTT2 = IPTT1 + N
      LREQ = IPTT2 + N - 1
      IF (LREQ .GT. LIWK) GO TO 220
      CALL JGROUP (N, IWK(IPIAN), IWK(IPJAN), MAXG, NGP, IWK(IPIGP),
     1   IWK(IPJGP), IWK(IPTT1), IWK(IPTT2), IER)
      IF (IER .NE. 0) GO TO 220
      LENIGP = NGP + 1
C
C Compute new ordering of rows/columns of Jacobian. --------------------
 160  IPR = IPIGP + LENIGP
```

```
      IPC = IPR
      IPIC = IPC + N
      IPISP = IPIC + N
      IPRSP = (IPISP - 2)/LRAT + 2
      IESP = LENWK + 1 - IPRSP
      IF (IESP .LT. 0) GO TO 230
      IBR = IPR - 1
      DO 170 I = 1,N
 170    IWK(IBR+I) = I
      NSP = LIWK + 1 - IPISP
      CALL ODRV (N, IWK(IPIAN), IWK(IPJAN), WK, IWK(IPR), IWK(IPIC),
     1   NSP, IWK(IPISP), 1, IYS)
      IF (IYS .EQ. 11*N+1) GO TO 240
      IF (IYS .NE. 0) GO TO 230
C
C Reorder JAN and do symbolic LU factorization of matrix. --------------
      IPA = LENWK + 1 - NNZ
      NSP = IPA - IPRSP
      LREQ = MAX(12*N/LRAT, 6*N/LRAT+2*N+NNZ) + 3
      LREQ = LREQ + IPRSP - 1 + NNZ
      IF (LREQ .GT. LENWK) GO TO 250
      IBA = IPA - 1
      DO 180 I = 1,NNZ
 180    WK(IBA+I) = 0.0D0
      IPISP = LRAT*(IPRSP - 1) + 1
      CALL CDRV (N,IWK(IPR),IWK(IPC),IWK(IPIC),IWK(IPIAN),IWK(IPJAN),
     1   WK(IPA),WK(IPA),WK(IPA),NSP,IWK(IPISP),WK(IPRSP),IESP,5,IYS)
      LREQ = LENWK - IESP
      IF (IYS .EQ. 10*N+1) GO TO 250
      IF (IYS .NE. 0) GO TO 260
      IPIL = IPISP
      IPIU = IPIL + 2*N + 1
      NZU = IWK(IPIL+N) - IWK(IPIL)
      NZL = IWK(IPIU+N) - IWK(IPIU)
      IF (LRAT .GT. 1) GO TO 190
      CALL ADJLR (N, IWK(IPISP), LDIF)
      LREQ = LREQ + LDIF
 190  CONTINUE
      IF (LRAT .EQ. 2 .AND. NNZ .EQ. N) LREQ = LREQ + 1
      NSP = NSP + LREQ - LENWK
      IPA = LREQ + 1 - NNZ
      IBA = IPA - 1
      IPPER = 0
      RETURN
C
 210  IPPER = -1
      LREQ = 2 + (2*N + 1)/LRAT
      LREQ = MAX(LENWK+1,LREQ)
      RETURN
C
 220  IPPER = -2
      LREQ = (LREQ - 1)/LRAT + 1
      RETURN
C
 230  IPPER = -3
      CALL CNTNZU (N, IWK(IPIAN), IWK(IPJAN), NZSUT)
      LREQ = LENWK - IESP + (3*N + 4*NZSUT - 1)/LRAT + 1
      RETURN
C
 240  IPPER = -4
      RETURN
C
 250  IPPER = -5
      RETURN
```

```
C
 260  IPPER = -6
      LREQ = LENWK
      RETURN
C----------------------- End of Subroutine DPREP -----------------------
      END
*DECK JGROUP
      SUBROUTINE JGROUP (N,IA,JA,MAXG,NGRP,IGP,JGP,INCL,JDONE,IER)
      INTEGER N, IA, JA, MAXG, NGRP, IGP, JGP, INCL, JDONE, IER
      DIMENSION IA(*), JA(*), IGP(*), JGP(*), INCL(*), JDONE(*)
C-----------------------------------------------------------------------
C This subroutine constructs groupings of the column indices of
C the Jacobian matrix, used in the numerical evaluation of the
C Jacobian by finite differences.
C
C Input:
C N      = the order of the matrix.
C IA,JA  = sparse structure descriptors of the matrix by rows.
C MAXG   = length of available storage in the IGP array.
C
C Output:
C NGRP   = number of groups.
C JGP    = array of length N containing the column indices by groups.
C IGP    = pointer array of length NGRP + 1 to the locations in JGP
C          of the beginning of each group.
C IER    = error indicator.  IER = 0 if no error occurred, or 1 if
C          MAXG was insufficient.
C
C INCL and JDONE are working arrays of length N.
C-----------------------------------------------------------------------
      INTEGER I, J, K, KMIN, KMAX, NCOL, NG
C
      IER = 0
      DO 10 J = 1,N
 10     JDONE(J) = 0
      NCOL = 1
      DO 60 NG = 1,MAXG
        IGP(NG) = NCOL
        DO 20 I = 1,N
 20       INCL(I) = 0
        DO 50 J = 1,N
C Reject column J if it is already in a group.--------------------------
          IF (JDONE(J) .EQ. 1) GO TO 50
          KMIN = IA(J)
          KMAX = IA(J+1) - 1
          DO 30 K = KMIN,KMAX
C Reject column J if it overlaps any column already in this group.------
            I = JA(K)
            IF (INCL(I) .EQ. 1) GO TO 50
 30         CONTINUE
C Accept column J into group NG.----------------------------------------
          JGP(NCOL) = J
          NCOL = NCOL + 1
          JDONE(J) = 1
          DO 40 K = KMIN,KMAX
            I = JA(K)
 40         INCL(I) = 1
 50       CONTINUE
C Stop if this group is empty (grouping is complete).-------------------
        IF (NCOL .EQ. IGP(NG)) GO TO 70
 60     CONTINUE
C Error return if not all columns were chosen (MAXG too small).---------
      IF (NCOL .LE. N) GO TO 80
      NG = MAXG
 70   NGRP = NG - 1
      RETURN
 80   IER = 1
      RETURN
C----------------------- End of Subroutine JGROUP ----------------------
      END
*DECK ADJLR
      SUBROUTINE ADJLR (N, ISP, LDIF)
      INTEGER N, ISP, LDIF
      DIMENSION ISP(*)
C-----------------------------------------------------------------------
C This routine computes an adjustment, LDIF, to the required
C integer storage space in IWK (sparse matrix work space).
C It is called only if the word length ratio is LRAT = 1.
C This is to account for the possibility that the symbolic LU phase
C may require more storage than the numerical LU and solution phases.
C-----------------------------------------------------------------------
      INTEGER IP, JLMAX, JUMAX, LNFC, LSFC, NZLU
C
      IP = 2*N + 1
C Get JLMAX = IJL(N) and JUMAX = IJU(N) (sizes of JL and JU). ----------
      JLMAX = ISP(IP)
      JUMAX = ISP(IP+IP)
C NZLU = (size of L) + (size of U) = (IL(N+1)-IL(1)) + (IU(N+1)-IU(1)).
      NZLU = ISP(N+1) - ISP(1) + ISP(IP+N+1) - ISP(IP+1)
      LSFC = 12*N + 3 + 2*MAX(JLMAX,JUMAX)
      LNFC = 9*N + 2 + JLMAX + JUMAX + NZLU
      LDIF = MAX(0, LSFC - LNFC)
      RETURN
C----------------------- End of Subroutine ADJLR -----------------------
      END
*DECK CNTNZU
      SUBROUTINE CNTNZU (N, IA, JA, NZSUT)
      INTEGER N, IA, JA, NZSUT
      DIMENSION IA(*), JA(*)
C-----------------------------------------------------------------------
C This routine counts the number of nonzero elements in the strict
C upper triangle of the matrix M + M(transpose), where the sparsity
C structure of M is given by pointer arrays IA and JA.
C This is needed to compute the storage requirements for the
C sparse matrix reordering operation in ODRV.
C-----------------------------------------------------------------------
      INTEGER II, JJ, J, JMIN, JMAX, K, KMIN, KMAX, NUM
C
      NUM = 0
      DO 50 II = 1,N
        JMIN = IA(II)
        JMAX = IA(II+1) - 1
        IF (JMIN .GT. JMAX) GO TO 50
        DO 40 J = JMIN,JMAX
          IF (JA(J) - II) 10, 40, 30
 10       JJ =JA(J)
          KMIN = IA(JJ)
          KMAX = IA(JJ+1) - 1
          IF (KMIN .GT. KMAX) GO TO 30
          DO 20 K = KMIN,KMAX
            IF (JA(K) .EQ. II) GO TO 40
 20         CONTINUE
 30       NUM = NUM + 1
 40       CONTINUE
 50     CONTINUE
      NZSUT = NUM
      RETURN
C----------------------- End of Subroutine CNTNZU ----------------------
```

```
      END
*DECK DINTDY
      SUBROUTINE DINTDY (T, K, YH, NYH, DKY, IFLAG)
C***BEGIN PROLOGUE  DINTDY
C***SUBSIDIARY
C***PURPOSE  Interpolate solution derivatives.
C***TYPE      DOUBLE PRECISION (SINTDY-S, DINTDY-D)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C  DINTDY computes interpolated values of the K-th derivative of the
C  dependent variable vector y, and stores it in DKY.  This routine
C  is called within the package with K = 0 and T = TOUT, but may
C  also be called by the user for any K up to the current order.
C  (See detailed instructions in the usage documentation.)
C
C  The computed values in DKY are gotten by interpolation using the
C  Nordsieck history array YH.  This array corresponds uniquely to a
C  vector-valued polynomial of degree NQCUR or less, and DKY is set
C  to the K-th derivative of this polynomial at T.
C  The formula for DKY is:
C               q
C   DKY(i)  =  sum  c(j,K) * (T - tn)**(j-K) * h**(-j) * YH(i,j+1)
C              j=K
C  where  c(j,K) = j*(j-1)*...*(j-K+1), q = NQCUR, tn = TCUR, h = HCUR.
C  The quantities  nq = NQCUR, l = nq+1, N = NEQ, tn, and h are
C  communicated by COMMON.  The above sum is done in reverse order.
C  IFLAG is returned negative if either K or T is out of bounds.
C
C***SEE ALSO  DLSODE
C***ROUTINES CALLED  XERRWD
C***COMMON BLOCKS    DLS001
C***REVISION HISTORY  (YYMMDD)
C   791129  DATE WRITTEN
C   890501  Modified prologue to SLATEC/LDOC format.  (FNF)
C   890503  Minor cosmetic changes.  (FNF)
C   930809  Renamed to allow single/double precision versions. (ACH)
C   010418  Reduced size of Common block /DLS001/. (ACH)
C   031105  Restored 'own' variables to Common block /DLS001/, to
C           enable interrupt/restart feature. (ACH)
C***END PROLOGUE  DINTDY
C**End
      INTEGER K, NYH, IFLAG
      DOUBLE PRECISION T, YH, DKY
      DIMENSION YH(NYH,*), DKY(*)
      INTEGER IOWND, IOWNS,
     1   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L,
     2   LYH, LEWT, LACOR, LSAVF, LWM, LIWM, METH, MITER,
     3   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      DOUBLE PRECISION ROWNS,
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND
      COMMON /DLS001/ ROWNS(209),
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND,
     2   IOWND(6), IOWNS(6),
     3   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L,
     4   LYH, LEWT, LACOR, LSAVF, LWM, LIWM, METH, MITER,
     5   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      INTEGER I, IC, J, JB, JB2, JJ, JJ1, JP1
      DOUBLE PRECISION C, R, S, TP
      CHARACTER*80 MSG
C
C***FIRST EXECUTABLE STATEMENT  DINTDY
      IFLAG = 0
      IF (K .LT. 0 .OR. K .GT. NQ) GO TO 80
      TP = TN - HU -  100.0D0*UROUND*(TN + HU)
      IF ((T-TP)*(T-TN) .GT. 0.0D0) GO TO 90
C
      S = (T - TN)/H
      IC = 1
      IF (K .EQ. 0) GO TO 15
      JJ1 = L - K
      DO 10 JJ = JJ1,NQ
 10     IC = IC*JJ
 15   C = IC
      DO 20 I = 1,N
 20     DKY(I) = C*YH(I,L)
      IF (K .EQ. NQ) GO TO 55
      JB2 = NQ - K
      DO 50 JB = 1,JB2
        J = NQ - JB
        JP1 = J + 1
        IC = 1
        IF (K .EQ. 0) GO TO 35
        JJ1 = JP1 - K
        DO 30 JJ = JJ1,J
 30       IC = IC*JJ
 35     C = IC
        DO 40 I = 1,N
 40       DKY(I) = C*YH(I,JP1) + S*DKY(I)
 50     CONTINUE
      IF (K .EQ. 0) RETURN
 55   R = H**(-K)
      DO 60 I = 1,N
 60     DKY(I) = R*DKY(I)
      RETURN
C
 80   MSG = 'DINTDY-  K (=I1) illegal      '
      CALL XERRWD (MSG, 30, 51, 0, 1, K, 0, 0, 0.0D0, 0.0D0)
      IFLAG = -1
      RETURN
 90   MSG = 'DINTDY-  T (=R1) illegal      '
      CALL XERRWD (MSG, 30, 52, 0, 0, 0, 0, 1, T, 0.0D0)
      MSG='      T not in interval TCUR - HU (= R1) to TCUR (=R2)      '
      CALL XERRWD (MSG, 60, 52, 0, 0, 0, 0, 2, TP, TN)
      IFLAG = -2
      RETURN
C----------------------- END OF SUBROUTINE DINTDY ----------------------
      END
*DECK DSTODE
      SUBROUTINE DSTODE (NEQ, Y, YH, NYH, YH1, EWT, SAVF, ACOR,
     1   WM, IWM, F, JAC, PJAC, SLVS)
C***BEGIN PROLOGUE  DSTODE
C***SUBSIDIARY
C***PURPOSE  Performs one step of an ODEPACK integration.
C***TYPE      DOUBLE PRECISION (SSTODE-S, DSTODE-D)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C  DSTODE performs one step of the integration of an initial value
C  problem for a system of ordinary differential equations.
C  Note:  DSTODE is independent of the value of the iteration method
C  indicator MITER, when this is .ne. 0, and hence is independent
C  of the type of chord method used, or the Jacobian structure.
C  Communication with DSTODE is done with the following variables:
C
C  NEQ    = integer array containing problem size in NEQ(1), and
C           passed as the NEQ argument in all calls to F and JAC.
C  Y      = an array of length .ge. N used as the Y argument in
```

```
C           all calls to F and JAC.
C  YH     = an NYH by LMAX array containing the dependent variables
C           and their approximate scaled derivatives, where
C           LMAX = MAXORD + 1.  YH(i,j+1) contains the approximate
C           j-th derivative of y(i), scaled by h**j/factorial(j)
C           (j = 0,1,...,NQ).  on entry for the first step, the first
C           two columns of YH must be set from the initial values.
C  NYH    = a constant integer .ge. N, the first dimension of YH.
C  YH1    = a one-dimensional array occupying the same space as YH.
C  EWT    = an array of length N containing multiplicative weights
C           for local error measurements.  Local errors in Y(i) are
C           compared to 1.0/EWT(i) in various error tests.
C  SAVF   = an array of working storage, of length N.
C           Also used for input of YH(*,MAXORD+2) when JSTART = -1
C           and MAXORD .lt. the current order NQ.
C  ACOR   = a work array of length N, used for the accumulated
C           corrections.  On a successful return, ACOR(i) contains
C           the estimated one-step local error in Y(i).
C  WM,IWM = real and integer work arrays associated with matrix
C           operations in chord iteration (MITER .ne. 0).
C  PJAC   = name of routine to evaluate and preprocess Jacobian matrix
C           and P = I - h*el0*JAC, if a chord method is being used.
C  SLVS   = name of routine to solve linear system in chord iteration.
C  CCMAX  = maximum relative change in h*el0 before PJAC is called.
C  H      = the step size to be attempted on the next step.
C           H is altered by the error control algorithm during the
C           problem.  H can be either positive or negative, but its
C           sign must remain constant throughout the problem.
C  HMIN   = the minimum absolute value of the step size h to be used.
C  HMXI   = inverse of the maximum absolute value of h to be used.
C           HMXI = 0.0 is allowed and corresponds to an infinite hmax.
C           HMIN and HMXI may be changed at any time, but will not
C           take effect until the next change of h is considered.
C  TN     = the independent variable. TN is updated on each step taken.
C  JSTART = an integer used for input only, with the following
C           values and meanings:
C                0  perform the first step.
C            .gt.0  take a new step continuing from the last.
C               -1  take the next step with a new value of H, MAXORD,
C                     N, METH, MITER, and/or matrix parameters.
C               -2  take the next step with a new value of H,
C                     but with other inputs unchanged.
C           On return, JSTART is set to 1 to facilitate continuation.
C  KFLAG  = a completion code with the following meanings:
C                0  the step was succesful.
C               -1  the requested error could not be achieved.
C               -2  corrector convergence could not be achieved.
C               -3  fatal error in PJAC or SLVS.
C           A return with KFLAG = -1 or -2 means either
C           abs(H) = HMIN or 10 consecutive failures occurred.
C           On a return with KFLAG negative, the values of TN and
C           the YH array are as of the beginning of the last
C           step, and H is the last step size attempted.
C  MAXORD = the maximum order of integration method to be allowed.
C  MAXCOR = the maximum number of corrector iterations allowed.
C  MSBP   = maximum number of steps between PJAC calls (MITER .gt. 0).
C  MXNCF  = maximum number of convergence failures allowed.
C  METH/MITER = the method flags.  See description in driver.
C  N      = the number of first-order differential equations.
C  The values of CCMAX, H, HMIN, HMXI, TN, JSTART, KFLAG, MAXORD,
C  MAXCOR, MSBP, MXNCF, METH, MITER, and N are communicated via COMMON.
C
C***SEE ALSO  DLSODE
C***ROUTINES CALLED  DCFODE, DVNORM
```

```
C***COMMON BLOCKS    DLS001
C***REVISION HISTORY  (YYMMDD)
C   791129  DATE WRITTEN
C   890501  Modified prologue to SLATEC/LDOC format.  (FNF)
C   890503  Minor cosmetic changes.  (FNF)
C   930809  Renamed to allow single/double precision versions. (ACH)
C   010418  Reduced size of Common block /DLS001/. (ACH)
C   031105  Restored 'own' variables to Common block /DLS001/, to
C           enable interrupt/restart feature. (ACH)
C***END PROLOGUE  DSTODE
C**End
      EXTERNAL F, JAC, PJAC, SLVS
      INTEGER NEQ, NYH, IWM
      DOUBLE PRECISION Y, YH, YH1, EWT, SAVF, ACOR, WM
      DIMENSION NEQ(*), Y(*), YH(NYH,*), YH1(*), EWT(*), SAVF(*),
     1   ACOR(*), WM(*), IWM(*)
      INTEGER IOWND, IALTH, IPUP, LMAX, MEO, NQNYH, NSLP,
     1   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L,
     2   LYH, LEWT, LACOR, LSAVF, LWM, LIWM, METH, MITER,
     3   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      INTEGER I, I1, IREDO, IRET, J, JB, M, NCF, NEWQ
      DOUBLE PRECISION CONIT, CRATE, EL, ELCO, HOLD, RMAX, TESCO,
     2   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND
      DOUBLE PRECISION DCON, DDN, DEL, DELP, DSM, DUP, EXDN, EXSM, EXUP,
     1   R, RH, RHDN, RHSM, RHUP, TOLD, DVNORM
      COMMON /DLS001/ CONIT, CRATE, EL(13), ELCO(13,12),
     1   HOLD, RMAX, TESCO(3,12),
     2   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND,
     3   IOWND(6), IALTH, IPUP, LMAX, MEO, NQNYH, NSLP,
     3   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L,
     4   LYH, LEWT, LACOR, LSAVF, LWM, LIWM, METH, MITER,
     5   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
C
C***FIRST EXECUTABLE STATEMENT  DSTODE
      KFLAG = 0
      TOLD = TN
      NCF = 0
      IERPJ = 0
      IERSL = 0
      JCUR = 0
      ICF = 0
      DELP = 0.0D0
      IF (JSTART .GT. 0) GO TO 200
      IF (JSTART .EQ. -1) GO TO 100
      IF (JSTART .EQ. -2) GO TO 160
C-----------------------------------------------------------------------
C On the first call, the order is set to 1, and other variables are
C initialized.  RMAX is the maximum ratio by which H can be increased
C in a single step.  It is initially 1.E4 to compensate for the small
C initial H, but then is normally equal to 10.  If a failure
C occurs (in corrector convergence or error test), RMAX is set to 2
C for the next increase.
C-----------------------------------------------------------------------
      LMAX = MAXORD + 1
      NQ = 1
      L = 2
      IALTH = 2
      RMAX = 10000.0D0
      RC = 0.0D0
      EL0 = 1.0D0
      CRATE = 0.7D0
      HOLD = H
      MEO = METH
      NSLP = 0
      IPUP = MITER
      IRET = 3
      GO TO 140
C-----------------------------------------------------------------------
C The following block handles preliminaries needed when JSTART = -1.
C IPUP is set to MITER to force a matrix update.
C If an order increase is about to be considered (IALTH = 1),
C IALTH is reset to 2 to postpone consideration one more step.
C If the caller has changed METH, DCFODE is called to reset
C the coefficients of the method.
C If the caller has changed MAXORD to a value less than the current
C order NQ, NQ is reduced to MAXORD, and a new H chosen accordingly.
C If H is to be changed, YH must be rescaled.
C If H or METH is being changed, IALTH is reset to L = NQ + 1
C to prevent further changes in H for that many steps.
C-----------------------------------------------------------------------
 100  IPUP = MITER
      LMAX = MAXORD + 1
      IF (IALTH .EQ. 1) IALTH = 2
      IF (METH .EQ. MEO) GO TO 110
      CALL DCFODE (METH, ELCO, TESCO)
      MEO = METH
      IF (NQ .GT. MAXORD) GO TO 120
      IALTH = L
      IRET = 1
      GO TO 150
 110  IF (NQ .LE. MAXORD) GO TO 160
 120  NQ = MAXORD
      L = LMAX
      DO 125 I = 1,L
 125    EL(I) = ELCO(I,NQ)
      NQNYH = NQ*NYH
      RC = RC*EL(1)/EL0
      EL0 = EL(1)
      CONIT = 0.5D0/(NQ+2)
      DDN = DVNORM (N, SAVF, EWT)/TESCO(1,L)
      EXDN = 1.0D0/L
      RHDN = 1.0D0/(1.3D0*DDN**EXDN + 0.0000013D0)
      RH = MIN(RHDN,1.0D0)
      IREDO = 3
      IF (H .EQ. HOLD) GO TO 170
      RH = MIN(RH,ABS(H/HOLD))
      H = HOLD
      GO TO 175
C-----------------------------------------------------------------------
C DCFODE is called to get all the integration coefficients for the
C current METH.  Then the EL vector and related constants are reset
C whenever the order NQ is changed, or at the start of the problem.
C-----------------------------------------------------------------------
 140  CALL DCFODE (METH, ELCO, TESCO)
 150  DO 155 I = 1,L
 155    EL(I) = ELCO(I,NQ)
      NQNYH = NQ*NYH
      RC = RC*EL(1)/EL0
      EL0 = EL(1)
      CONIT = 0.5D0/(NQ+2)
      GO TO (160, 170, 200), IRET
C-----------------------------------------------------------------------
C If H is being changed, the H ratio RH is checked against
C RMAX, HMIN, and HMXI, and the YH array rescaled.  IALTH is set to
C L = NQ + 1 to prevent a change of H for that many steps, unless
C forced by a convergence or error test failure.
C-----------------------------------------------------------------------
 160  IF (H .EQ. HOLD) GO TO 200
```

```
      RH = H/HOLD
      H = HOLD
      IREDO = 3
      GO TO 175
 170  RH = MAX(RH,HMIN/ABS(H))
 175  RH = MIN(RH,RMAX)
      RH = RH/MAX(1.0D0,ABS(H)*HMXI*RH)
      R = 1.0D0
      DO 180 J = 2,L
        R = R*RH
        DO 180 I = 1,N
 180      YH(I,J) = YH(I,J)*R
      H = H*RH
      RC = RC*RH
      IALTH = L
      IF (IREDO .EQ. 0) GO TO 690
C-----------------------------------------------------------------------
C This section computes the predicted values by effectively
C multiplying the YH array by the Pascal Triangle matrix.
C RC is the ratio of new to old values of the coefficient  H*EL(1).
C When RC differs from 1 by more than CCMAX, IPUP is set to MITER
C to force PJAC to be called, if a Jacobian is involved.
C In any case, PJAC is called at least every MSBP steps.
C-----------------------------------------------------------------------
 200  IF (ABS(RC-1.0D0) .GT. CCMAX) IPUP = MITER
      IF (NST .GE. NSLP+MSBP) IPUP = MITER
      TN = TN + H
      I1 = NQNYH + 1
      DO 215 JB = 1,NQ
        I1 = I1 - NYH
Cdir$ ivdep
        DO 210 I = I1,NQNYH
 210      YH1(I) = YH1(I) + YH1(I+NYH)
 215    CONTINUE
C-----------------------------------------------------------------------
C Up to MAXCOR corrector iterations are taken.  A convergence test is
C made on the R.M.S. norm of each correction, weighted by the error
C weight vector EWT.  The sum of the corrections is accumulated in the
C vector ACOR(i).  The YH array is not altered in the corrector loop.
C-----------------------------------------------------------------------
 220  M = 0
      DO 230 I = 1,N
 230    Y(I) = YH(I,1)
      CALL F (NEQ, TN, Y, SAVF)
      NFE = NFE + 1
      IF (IPUP .LE. 0) GO TO 250
C-----------------------------------------------------------------------
C If indicated, the matrix P = I - h*el(1)*J is reevaluated and
C preprocessed before starting the corrector iteration.  IPUP is set
C to 0 as an indicator that this has been done.
C-----------------------------------------------------------------------
      CALL PJAC (NEQ, Y, YH, NYH, EWT, ACOR, SAVF, WM, IWM, F, JAC)
      IPUP = 0
      RC = 1.0D0
      NSLP = NST
      CRATE = 0.7D0
      IF (IERPJ .NE. 0) GO TO 430
 250  DO 260 I = 1,N
 260    ACOR(I) = 0.0D0
 270  IF (MITER .NE. 0) GO TO 350
C-----------------------------------------------------------------------
C In the case of functional iteration, update Y directly from
C the result of the last function evaluation.
C-----------------------------------------------------------------------
```

```
      DO 290 I = 1,N
        SAVF(I) = H*SAVF(I) - YH(I,2)
 290    Y(I) = SAVF(I) - ACOR(I)
      DEL = DVNORM (N, Y, EWT)
      DO 300 I = 1,N
        Y(I) = YH(I,1) + EL(1)*SAVF(I)
 300    ACOR(I) = SAVF(I)
      GO TO 400
C-----------------------------------------------------------------------
C In the case of the chord method, compute the corrector error,
C and solve the linear system with that as right-hand side and
C P as coefficient matrix.
C-----------------------------------------------------------------------
 350  DO 360 I = 1,N
 360    Y(I) = H*SAVF(I) - (YH(I,2) + ACOR(I))
      CALL SLVS (WM, IWM, Y, SAVF)
      IF (IERSL .LT. 0) GO TO 430
      IF (IERSL .GT. 0) GO TO 410
      DEL = DVNORM (N, Y, EWT)
      DO 380 I = 1,N
        ACOR(I) = ACOR(I) + Y(I)
 380    Y(I) = YH(I,1) + EL(1)*ACOR(I)
C-----------------------------------------------------------------------
C Test for convergence.  If M.gt.0, an estimate of the convergence
C rate constant is stored in CRATE, and this is used in the test.
C-----------------------------------------------------------------------
 400  IF (M .NE. 0) CRATE = MAX(0.2D0*CRATE,DEL/DELP)
      DCON = DEL*MIN(1.0D0,1.5D0*CRATE)/(TESCO(2,NQ)*CONIT)
      IF (DCON .LE. 1.0D0) GO TO 450
      M = M + 1
      IF (M .EQ. MAXCOR) GO TO 410
      IF (M .GE. 2 .AND. DEL .GT. 2.0D0*DELP) GO TO 410
      DELP = DEL
      CALL F (NEQ, TN, Y, SAVF)
      NFE = NFE + 1
      GO TO 270
C-----------------------------------------------------------------------
C The corrector iteration failed to converge.
C If MITER .ne. 0 and the Jacobian is out of date, PJAC is called for
C the next try.  Otherwise the YH array is retracted to its values
C before prediction, and H is reduced, if possible.  If H cannot be
C reduced or MXNCF failures have occurred, exit with KFLAG = -2.
C-----------------------------------------------------------------------
 410  IF (MITER .EQ. 0 .OR. JCUR .EQ. 1) GO TO 430
      ICF = 1
      IPUP = MITER
      GO TO 220
 430  ICF = 2
      NCF = NCF + 1
      RMAX = 2.0D0
      TN = TOLD
      I1 = NQNYH + 1
      DO 445 JB = 1,NQ
        I1 = I1 - NYH
Cdir$ ivdep
        DO 440 I = I1,NQNYH
 440      YH1(I) = YH1(I) - YH1(I+NYH)
 445    CONTINUE
      IF (IERPJ .LT. 0 .OR. IERSL .LT. 0) GO TO 680
      IF (ABS(H) .LE. HMIN*1.00001D0) GO TO 670
      IF (NCF .EQ. MXNCF) GO TO 670
      RH = 0.25D0
      IPUP = MITER
      IREDO = 1
      GO TO 170
C-----------------------------------------------------------------------
C The corrector has converged.  JCUR is set to 0
C to signal that the Jacobian involved may need updating later.
C The local error test is made and control passes to statement 500
C if it fails.
C-----------------------------------------------------------------------
 450  JCUR = 0
      IF (M .EQ. 0) DSM = DEL/TESCO(2,NQ)
      IF (M .GT. 0) DSM = DVNORM (N, ACOR, EWT)/TESCO(2,NQ)
      IF (DSM .GT. 1.0D0) GO TO 500
C-----------------------------------------------------------------------
C After a successful step, update the YH array.
C Consider changing H if IALTH = 1.  Otherwise decrease IALTH by 1.
C If IALTH is then 1 and NQ .lt. MAXORD, then ACOR is saved for
C use in a possible order increase on the next step.
C If a change in H is considered, an increase or decrease in order
C by one is considered also.  A change in H is made only if it is by a
C factor of at least 1.1.  If not, IALTH is set to 3 to prevent
C testing for that many steps.
C-----------------------------------------------------------------------
      KFLAG = 0
      IREDO = 0
      NST = NST + 1
      HU = H
      NQU = NQ
      DO 470 J = 1,L
        DO 470 I = 1,N
 470      YH(I,J) = YH(I,J) + EL(J)*ACOR(I)
      IALTH = IALTH - 1
      IF (IALTH .EQ. 0) GO TO 520
      IF (IALTH .GT. 1) GO TO 700
      IF (L .EQ. LMAX) GO TO 700
      DO 490 I = 1,N
 490    YH(I,LMAX) = ACOR(I)
      GO TO 700
C-----------------------------------------------------------------------
C The error test failed.  KFLAG keeps track of multiple failures.
C Restore TN and the YH array to their previous values, and prepare
C to try the step again.  Compute the optimum step size for this or
C one lower order.  After 2 or more failures, H is forced to decrease
C by a factor of 0.2 or less.
C-----------------------------------------------------------------------
 500  KFLAG = KFLAG - 1
      TN = TOLD
      I1 = NQNYH + 1
      DO 515 JB = 1,NQ
        I1 = I1 - NYH
Cdir$ ivdep
        DO 510 I = I1,NQNYH
 510      YH1(I) = YH1(I) - YH1(I+NYH)
 515    CONTINUE
      RMAX = 2.0D0
      IF (ABS(H) .LE. HMIN*1.00001D0) GO TO 660
      IF (KFLAG .LE. -3) GO TO 640
      IREDO = 2
      RHUP = 0.0D0
      GO TO 540
C-----------------------------------------------------------------------
C Regardless of the success or failure of the step, factors
C RHDN, RHSM, and RHUP are computed, by which H could be multiplied
C at order NQ - 1, order NQ, or order NQ + 1, respectively.
C In the case of failure, RHUP = 0.0 to avoid an order increase.
C The largest of these is determined and the new order chosen
```

```
C accordingly.  If the order is to be increased, we compute one
C additional scaled derivative.
C-----------------------------------------------------------------------
 520  RHUP = 0.0D0
      IF (L .EQ. LMAX) GO TO 540
      DO 530 I = 1,N
 530    SAVF(I) = ACOR(I) - YH(I,LMAX)
      DUP = DVNORM (N, SAVF, EWT)/TESCO(3,NQ)
      EXUP = 1.0D0/(L+1)
      RHUP = 1.0D0/(1.4D0*DUP**EXUP + 0.0000014D0)
 540  EXSM = 1.0D0/L
      RHSM = 1.0D0/(1.2D0*DSM**EXSM + 0.0000012D0)
      RHDN = 0.0D0
      IF (NQ .EQ. 1) GO TO 560
      DDN = DVNORM (N, YH(1,L), EWT)/TESCO(1,NQ)
      EXDN = 1.0D0/NQ
      RHDN = 1.0D0/(1.3D0*DDN**EXDN + 0.0000013D0)
 560  IF (RHSM .GE. RHUP) GO TO 570
      IF (RHUP .GT. RHDN) GO TO 590
      GO TO 580
 570  IF (RHSM .LT. RHDN) GO TO 580
      NEWQ = NQ
      RH = RHSM
      GO TO 620
 580  NEWQ = NQ - 1
      RH = RHDN
      IF (KFLAG .LT. 0 .AND. RH .GT. 1.0D0) RH = 1.0D0
      GO TO 620
 590  NEWQ = L
      RH = RHUP
      IF (RH .LT. 1.1D0) GO TO 610
      R = EL(L)/L
      DO 600 I = 1,N
 600    YH(I,NEWQ+1) = ACOR(I)*R
      GO TO 630
 610  IALTH = 3
      GO TO 700
 620  IF ((KFLAG .EQ. 0) .AND. (RH .LT. 1.1D0)) GO TO 610
      IF (KFLAG .LE. -2) RH = MIN(RH,0.2D0)
C-----------------------------------------------------------------------
C If there is a change of order, reset NQ, l, and the coefficients.
C In any case H is reset according to RH and the YH array is rescaled.
C Then exit from 690 if the step was OK, or redo the step otherwise.
C-----------------------------------------------------------------------
      IF (NEWQ .EQ. NQ) GO TO 170
 630  NQ = NEWQ
      L = NQ + 1
      IRET = 2
      GO TO 150
C-----------------------------------------------------------------------
C Control reaches this section if 3 or more failures have occured.
C If 10 failures have occurred, exit with KFLAG = -1.
C It is assumed that the derivatives that have accumulated in the
C YH array have errors of the wrong order.  Hence the first
C derivative is recomputed, and the order is set to 1.  Then
C H is reduced by a factor of 10, and the step is retried,
C until it succeeds or H reaches HMIN.
C-----------------------------------------------------------------------
 640  IF (KFLAG .EQ. -10) GO TO 660
      RH = 0.1D0
      RH = MAX(HMIN/ABS(H),RH)
      H = H*RH
      DO 645 I = 1,N
 645    Y(I) = YH(I,1)
```

```
      CALL F (NEQ, TN, Y, SAVF)
      NFE = NFE + 1
      DO 650 I = 1,N
 650    YH(I,2) = H*SAVF(I)
      IPUP = MITER
      IALTH = 5
      IF (NQ .EQ. 1) GO TO 200
      NQ = 1
      L = 2
      IRET = 3
      GO TO 150
C-----------------------------------------------------------------------
C All returns are made through this section.  H is saved in HOLD
C to allow the caller to change H on the next step.
C-----------------------------------------------------------------------
 660  KFLAG = -1
      GO TO 720
 670  KFLAG = -2
      GO TO 720
 680  KFLAG = -3
      GO TO 720
 690  RMAX = 10.0D0
 700  R = 1.0D0/TESCO(2,NQU)
      DO 710 I = 1,N
 710    ACOR(I) = ACOR(I)*R
 720  HOLD = H
      JSTART = 1
      RETURN
C----------------------- END OF SUBROUTINE DSTODE ----------------------
      END
*DECK DCFODE
      SUBROUTINE DCFODE (METH, ELCO, TESCO)
C***BEGIN PROLOGUE  DCFODE
C***SUBSIDIARY
C***PURPOSE  Set ODE integrator coefficients.
C***TYPE      DOUBLE PRECISION (SCFODE-S, DCFODE-D)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C  DCFODE is called by the integrator routine to set coefficients
C  needed there.  The coefficients for the current method, as
C  given by the value of METH, are set for all orders and saved.
C  The maximum order assumed here is 12 if METH = 1 and 5 if METH = 2.
C  (A smaller value of the maximum order is also allowed.)
C  DCFODE is called once at the beginning of the problem,
C  and is not called again unless and until METH is changed.
C
C  The ELCO array contains the basic method coefficients.
C  The coefficients el(i), 1 .le. i .le. nq+1, for the method of
C  order nq are stored in ELCO(i,nq).  They are given by a genetrating
C  polynomial, i.e.,
C      l(x) = el(1) + el(2)*x + ... + el(nq+1)*x**nq.
C  For the implicit Adams methods, l(x) is given by
C      dl/dx = (x+1)*(x+2)*...*(x+nq-1)/factorial(nq-1),    l(-1) = 0.
C  For the BDF methods, l(x) is given by
C      l(x) = (x+1)*(x+2)* ... *(x+nq)/K,
C  where         K = factorial(nq)*(1 + 1/2 + ... + 1/nq).
C
C  The TESCO array contains test constants used for the
C  local error test and the selection of step size and/or order.
C  At order nq, TESCO(k,nq) is used for the selection of step
C  size at order nq - 1 if k = 1, at order nq if k = 2, and at order
C  nq + 1 if k = 3.
C
```

```
C***SEE ALSO  DLSODE
C***ROUTINES CALLED  (NONE)
C***REVISION HISTORY  (YYMMDD)
C   791129  DATE WRITTEN
C   890501  Modified prologue to SLATEC/LDOC format.  (FNF)
C   890503  Minor cosmetic changes.  (FNF)
C   930809  Renamed to allow single/double precision versions. (ACH)
C***END PROLOGUE  DCFODE
C**End
      INTEGER METH
      INTEGER I, IB, NQ, NQM1, NQP1
      DOUBLE PRECISION ELCO, TESCO
      DOUBLE PRECISION AGAMQ, FNQ, FNQM1, PC, PINT, RAGQ,
     1   RQFAC, RQ1FAC, TSIGN, XPIN
      DIMENSION ELCO(13,12), TESCO(3,12)
      DIMENSION PC(12)
C
C***FIRST EXECUTABLE STATEMENT  DCFODE
      GO TO (100, 200), METH
C
 100  ELCO(1,1) = 1.0D0
      ELCO(2,1) = 1.0D0
      TESCO(1,1) = 0.0D0
      TESCO(2,1) = 2.0D0
      TESCO(1,2) = 1.0D0
      TESCO(3,12) = 0.0D0
      PC(1) = 1.0D0
      RQFAC = 1.0D0
      DO 140 NQ = 2,12
C-----------------------------------------------------------------------
C The PC array will contain the coefficients of the polynomial
C     p(x) = (x+1)*(x+2)*...*(x+nq-1).
C Initially, p(x) = 1.
C-----------------------------------------------------------------------
        RQ1FAC = RQFAC
        RQFAC = RQFAC/NQ
        NQM1 = NQ - 1
        FNQM1 = NQM1
        NQP1 = NQ + 1
C Form coefficients of p(x)*(x+nq-1). ----------------------------------
        PC(NQ) = 0.0D0
        DO 110 IB = 1,NQM1
          I = NQP1 - IB
 110      PC(I) = PC(I-1) + FNQM1*PC(I)
        PC(1) = FNQM1*PC(1)
C Compute integral, -1 to 0, of p(x) and x*p(x). -----------------------
        PINT = PC(1)
        XPIN = PC(1)/2.0D0
        TSIGN = 1.0D0
        DO 120 I = 2,NQ
          TSIGN = -TSIGN
          PINT = PINT + TSIGN*PC(I)/I
 120      XPIN = XPIN + TSIGN*PC(I)/(I+1)
C Store coefficients in ELCO and TESCO. --------------------------------
        ELCO(1,NQ) = PINT*RQ1FAC
        ELCO(2,NQ) = 1.0D0
        DO 130 I = 2,NQ
 130      ELCO(I+1,NQ) = RQ1FAC*PC(I)/I
        AGAMQ = RQFAC*XPIN
        RAGQ = 1.0D0/AGAMQ
        TESCO(2,NQ) = RAGQ
        IF (NQ .LT. 12) TESCO(1,NQP1) = RAGQ*RQFAC/NQP1
        TESCO(3,NQM1) = RAGQ
 140    CONTINUE
```

```
      RETURN
C
 200  PC(1) = 1.0D0
      RQ1FAC = 1.0D0
      DO 230 NQ = 1,5
C-----------------------------------------------------------------------
C The PC array will contain the coefficients of the polynomial
C     p(x) = (x+1)*(x+2)*...*(x+nq).
C Initially, p(x) = 1.
C-----------------------------------------------------------------------
        FNQ = NQ
        NQP1 = NQ + 1
C Form coefficients of p(x)*(x+nq). ------------------------------------
        PC(NQP1) = 0.0D0
        DO 210 IB = 1,NQ
          I = NQ + 2 - IB
 210      PC(I) = PC(I-1) + FNQ*PC(I)
        PC(1) = FNQ*PC(1)
C Store coefficients in ELCO and TESCO. --------------------------------
        DO 220 I = 1,NQP1
 220      ELCO(I,NQ) = PC(I)/PC(2)
        ELCO(2,NQ) = 1.0D0
        TESCO(1,NQ) = RQ1FAC
        TESCO(2,NQ) = NQP1/ELCO(1,NQ)
        TESCO(3,NQ) = (NQ+2)/ELCO(1,NQ)
        RQ1FAC = RQ1FAC/FNQ
 230    CONTINUE
      RETURN
C----------------------- END OF SUBROUTINE DCFODE ----------------------
      END
*DECK DPRJS
      SUBROUTINE DPRJS (NEQ,Y,YH,NYH,EWT,FTEM,SAVF,WK,IWK,F,JAC)
      EXTERNAL F,JAC
      INTEGER NEQ, NYH, IWK
      DOUBLE PRECISION Y, YH, EWT, FTEM, SAVF, WK
      DIMENSION NEQ(*), Y(*), YH(NYH,*), EWT(*), FTEM(*), SAVF(*),
     1   WK(*), IWK(*)
      INTEGER IOWND, IOWNS,
     1   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L,
     2   LYH, LEWT, LACOR, LSAVF, LWM, LIWM, METH, MITER,
     3   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      INTEGER IPLOST, IESP, ISTATC, IYS, IBA, IBIAN, IBJAN, IBJGP,
     1   IPIAN, IPJAN, IPJGP, IPIGP, IPR, IPC, IPIC, IPISP, IPRSP, IPA,
     2   LENYH, LENYHM, LENWK, LREQ, LRAT, LREST, LWMIN, MOSS, MSBJ,
     3   NSLJ, NGP, NLU, NNZ, NSP, NZL, NZU
      DOUBLE PRECISION ROWNS,
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND
      DOUBLE PRECISION CON0, CONMIN, CCMXJ, PSMALL, RBIG, SETH
      COMMON /DLS001/ ROWNS(209),
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND,
     2   IOWND(6), IOWNS(6),
     3   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L,
     4   LYH, LEWT, LACOR, LSAVF, LWM, LIWM, METH, MITER,
     5   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      COMMON /DLSS01/ CON0, CONMIN, CCMXJ, PSMALL, RBIG, SETH,
     1   IPLOST, IESP, ISTATC, IYS, IBA, IBIAN, IBJAN, IBJGP,
     2   IPIAN, IPJAN, IPJGP, IPIGP, IPR, IPC, IPIC, IPISP, IPRSP, IPA,
     3   LENYH, LENYHM, LENWK, LREQ, LRAT, LREST, LWMIN, MOSS, MSBJ,
     4   NSLJ, NGP, NLU, NNZ, NSP, NZL, NZU
      INTEGER I, IMUL, J, JJ, JOK, JMAX, JMIN, K, KMAX, KMIN, NG
      DOUBLE PRECISION CON, DI, FAC, HL0, PIJ, R, R0, RCON, RCONT,
     1   SRUR, DVNORM
C-----------------------------------------------------------------------
C DPRJS is called to compute and process the matrix
```

```
C P = I - H*EL(1)*J , where J is an approximation to the Jacobian.
C J is computed by columns, either by the user-supplied routine JAC
C if MITER = 1, or by finite differencing if MITER = 2.
C if MITER = 3, a diagonal approximation to J is used.
C if MITER = 1 or 2, and if the existing value of the Jacobian
C (as contained in P) is considered acceptable, then a new value of
C P is reconstructed from the old value.  In any case, when MITER
C is 1 or 2, the P matrix is subjected to LU decomposition in CDRV.
C P and its LU decomposition are stored (separately) in WK.
C
C In addition to variables described previously, communication
C with DPRJS uses the following:
C Y     = array containing predicted values on entry.
C FTEM  = work array of length N (ACOR in DSTODE).
C SAVF  = array containing f evaluated at predicted y.
C WK    = real work space for matrices.  On output it contains the
C         inverse diagonal matrix if MITER = 3, and P and its sparse
C         LU decomposition if MITER is 1 or 2.
C         Storage of matrix elements starts at WK(3).
C         WK also contains the following matrix-related data:
C         WK(1) = SQRT(UROUND), used in numerical Jacobian increments.
C         WK(2) = H*EL0, saved for later use if MITER = 3.
C IWK   = integer work space for matrix-related data, assumed to
C         be equivalenced to WK.  In addition, WK(IPRSP) and IWK(IPISP)
C         are assumed to have identical locations.
C EL0   = EL(1) (input).
C IERPJ = output error flag (in Common).
C       = 0 if no error.
C       = 1  if zero pivot found in CDRV.
C       = 2  if a singular matrix arose with MITER = 3.
C       = -1 if insufficient storage for CDRV (should not occur here).
C       = -2 if other error found in CDRV (should not occur here).
C JCUR  = output flag showing status of (approximate) Jacobian matrix:
C          = 1 to indicate that the Jacobian is now current, or
C          = 0 to indicate that a saved value was used.
C This routine also uses other variables in Common.
C-----------------------------------------------------------------------
      HL0 = H*EL0
      CON = -HL0
      IF (MITER .EQ. 3) GO TO 300
C See whether J should be reevaluated (JOK = 0) or not (JOK = 1). ------
      JOK = 1
      IF (NST .EQ. 0 .OR. NST .GE. NSLJ+MSBJ) JOK = 0
      IF (ICF .EQ. 1 .AND. ABS(RC - 1.0D0) .LT. CCMXJ) JOK = 0
      IF (ICF .EQ. 2) JOK = 0
      IF (JOK .EQ. 1) GO TO 250
C
C MITER = 1 or 2, and the Jacobian is to be reevaluated. ---------------
 20   JCUR = 1
      NJE = NJE + 1
      NSLJ = NST
      IPLOST = 0
      CONMIN = ABS(CON)
      GO TO (100, 200), MITER
C
C If MITER = 1, call JAC, multiply by scalar, and add identity. --------
 100  CONTINUE
      KMIN = IWK(IPIAN)
      DO 130 J = 1, N
        KMAX = IWK(IPIAN+J) - 1
        DO 110 I = 1,N
 110      FTEM(I) = 0.0D0
        CALL JAC (NEQ, TN, Y, J, IWK(IPIAN), IWK(IPJAN), FTEM)
        DO 120 K = KMIN, KMAX
```

```
          I = IWK(IBJAN+K)
          WK(IBA+K) = FTEM(I)*CON
          IF (I .EQ. J) WK(IBA+K) = WK(IBA+K) + 1.0D0
 120      CONTINUE
        KMIN = KMAX + 1
 130    CONTINUE
      GO TO 290
C
C If MITER = 2, make NGP calls to F to approximate J and P. ------------
 200  CONTINUE
      FAC = DVNORM(N, SAVF, EWT)
      R0 = 1000.0D0 * ABS(H) * UROUND * N * FAC
      IF (R0 .EQ. 0.0D0) R0 = 1.0D0
      SRUR = WK(1)
      JMIN = IWK(IPIGP)
      DO 240 NG = 1,NGP
        JMAX = IWK(IPIGP+NG) - 1
        DO 210 J = JMIN,JMAX
          JJ = IWK(IBJGP+J)
          R = MAX(SRUR*ABS(Y(JJ)),R0/EWT(JJ))
 210      Y(JJ) = Y(JJ) + R
        CALL F (NEQ, TN, Y, FTEM)
        DO 230 J = JMIN,JMAX
          JJ = IWK(IBJGP+J)
          Y(JJ) = YH(JJ,1)
          R = MAX(SRUR*ABS(Y(JJ)),R0/EWT(JJ))
          FAC = -HL0/R
          KMIN =IWK(IBIAN+JJ)
          KMAX =IWK(IBIAN+JJ+1) - 1
          DO 220 K = KMIN,KMAX
            I = IWK(IBJAN+K)
            WK(IBA+K) = (FTEM(I) - SAVF(I))*FAC
            IF (I .EQ. JJ) WK(IBA+K) = WK(IBA+K) + 1.0D0
 220        CONTINUE
 230      CONTINUE
        JMIN = JMAX + 1
 240    CONTINUE
      NFE = NFE + NGP
      GO TO 290
C
C If JOK = 1, reconstruct new P from old P. ----------------------------
 250  JCUR = 0
      RCON = CON/CON0
      RCONT = ABS(CON)/CONMIN
      IF (RCONT .GT. RBIG .AND. IPLOST .EQ. 1) GO TO 20
      KMIN = IWK(IPIAN)
      DO 275 J = 1,N
        KMAX = IWK(IPIAN+J) - 1
        DO 270 K = KMIN,KMAX
          I = IWK(IBJAN+K)
          PIJ = WK(IBA+K)
          IF (I .NE. J) GO TO 260
          PIJ = PIJ - 1.0D0
          IF (ABS(PIJ) .GE. PSMALL) GO TO 260
            IPLOST = 1
            CONMIN = MIN(ABS(CON0),CONMIN)
 260      PIJ = PIJ*RCON
          IF (I .EQ. J) PIJ = PIJ + 1.0D0
          WK(IBA+K) = PIJ
 270      CONTINUE
        KMIN = KMAX + 1
 275    CONTINUE
C
C Do numerical factorization of P matrix. ------------------------------
```

```
 290  NLU = NLU + 1
      CON0 = CON
      IERPJ = 0
      DO 295 I = 1,N
 295    FTEM(I) = 0.0D0
      CALL CDRV (N,IWK(IPR),IWK(IPC),IWK(IPIC),IWK(IPIAN),IWK(IPJAN),
     1   WK(IPA),FTEM,FTEM,NSP,IWK(IPISP),WK(IPRSP),IESP,2,IYS)
      IF (IYS .EQ. 0) RETURN
      IMUL = (IYS - 1)/N
      IERPJ = -2
      IF (IMUL .EQ. 8) IERPJ = 1
      IF (IMUL .EQ. 10) IERPJ = -1
      RETURN
C
C If MITER = 3, construct a diagonal approximation to J and P. ---------
 300  CONTINUE
      JCUR = 1
      NJE = NJE + 1
      WK(2) = HL0
      IERPJ = 0
      R = EL0*0.1D0
      DO 310 I = 1,N
 310    Y(I) = Y(I) + R*(H*SAVF(I) - YH(I,2))
      CALL F (NEQ, TN, Y, WK(3))
      NFE = NFE + 1
      DO 320 I = 1,N
        R0 = H*SAVF(I) - YH(I,2)
        DI = 0.1D0*R0 - H*(WK(I+2) - SAVF(I))
        WK(I+2) = 1.0D0
        IF (ABS(R0) .LT. UROUND/EWT(I)) GO TO 320
        IF (ABS(DI) .EQ. 0.0D0) GO TO 330
        WK(I+2) = 0.1D0*R0/DI
 320    CONTINUE
      RETURN
 330  IERPJ = 2
      RETURN
C----------------------- End of Subroutine DPRJS -----------------------
      END
*DECK DSOLSS
      SUBROUTINE DSOLSS (WK, IWK, X, TEM)
      INTEGER IWK
      DOUBLE PRECISION WK, X, TEM
      DIMENSION WK(*), IWK(*), X(*), TEM(*)
      INTEGER IOWND, IOWNS,
     1   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L,
     2   LYH, LEWT, LACOR, LSAVF, LWM, LIWM, METH, MITER,
     3   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      INTEGER IPLOST, IESP, ISTATC, IYS, IBA, IBIAN, IBJAN, IBJGP,
     1   IPIAN, IPJAN, IPJGP, IPIGP, IPR, IPC, IPIC, IPISP, IPRSP, IPA,
     2   LENYH, LENYHM, LENWK, LREQ, LRAT, LREST, LWMIN, MOSS, MSBJ,
     3   NSLJ, NGP, NLU, NNZ, NSP, NZL, NZU
      DOUBLE PRECISION ROWNS,
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND
      DOUBLE PRECISION RLSS
      COMMON /DLS001/ ROWNS(209),
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND,
     2   IOWND(6), IOWNS(6),
     3   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L,
     4   LYH, LEWT, LACOR, LSAVF, LWM, LIWM, METH, MITER,
     5   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      COMMON /DLSS01/ RLSS(6),
     1   IPLOST, IESP, ISTATC, IYS, IBA, IBIAN, IBJAN, IBJGP,
     2   IPIAN, IPJAN, IPJGP, IPIGP, IPR, IPC, IPIC, IPISP, IPRSP, IPA,
     3   LENYH, LENYHM, LENWK, LREQ, LRAT, LREST, LWMIN, MOSS, MSBJ,
     4   NSLJ, NGP, NLU, NNZ, NSP, NZL, NZU
      INTEGER I
      DOUBLE PRECISION DI, HL0, PHL0, R
C-----------------------------------------------------------------------
C This routine manages the solution of the linear system arising from
C a chord iteration.  It is called if MITER .ne. 0.
C If MITER is 1 or 2, it calls CDRV to accomplish this.
C If MITER = 3 it updates the coefficient H*EL0 in the diagonal
C matrix, and then computes the solution.
C communication with DSOLSS uses the following variables:
C WK    = real work space containing the inverse diagonal matrix if
C         MITER = 3 and the LU decomposition of the matrix otherwise.
C         Storage of matrix elements starts at WK(3).
C         WK also contains the following matrix-related data:
C         WK(1) = SQRT(UROUND) (not used here),
C         WK(2) = HL0, the previous value of H*EL0, used if MITER = 3.
C IWK   = integer work space for matrix-related data, assumed to
C         be equivalenced to WK.  In addition, WK(IPRSP) and IWK(IPISP)
C         are assumed to have identical locations.
C X     = the right-hand side vector on input, and the solution vector
C         on output, of length N.
C TEM   = vector of work space of length N, not used in this version.
C IERSL = output flag (in Common).
C         IERSL = 0  if no trouble occurred.
C         IERSL = -1 if CDRV returned an error flag (MITER = 1 or 2).
C                    This should never occur and is considered fatal.
C         IERSL = 1  if a singular matrix arose with MITER = 3.
C This routine also uses other variables in Common.
C-----------------------------------------------------------------------
      IERSL = 0
      GO TO (100, 100, 300), MITER
 100  CALL CDRV (N,IWK(IPR),IWK(IPC),IWK(IPIC),IWK(IPIAN),IWK(IPJAN),
     1   WK(IPA),X,X,NSP,IWK(IPISP),WK(IPRSP),IESP,4,IERSL)
      IF (IERSL .NE. 0) IERSL = -1
      RETURN
C
 300  PHL0 = WK(2)
      HL0 = H*EL0
      WK(2) = HL0
      IF (HL0 .EQ. PHL0) GO TO 330
      R = HL0/PHL0
      DO 320 I = 1,N
        DI = 1.0D0 - R*(1.0D0 - 1.0D0/WK(I+2))
        IF (ABS(DI) .EQ. 0.0D0) GO TO 390
 320    WK(I+2) = 1.0D0/DI
 330  DO 340 I = 1,N
 340    X(I) = WK(I+2)*X(I)
      RETURN
 390  IERSL = 1
      RETURN
C
C----------------------- End of Subroutine DSOLSS ----------------------
      END
*DECK DEWSET
      SUBROUTINE DEWSET (N, ITOL, RTOL, ATOL, YCUR, EWT)
C***BEGIN PROLOGUE  DEWSET
C***SUBSIDIARY
C***PURPOSE  Set error weight vector.
C***TYPE      DOUBLE PRECISION (SEWSET-S, DEWSET-D)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C  This subroutine sets the error weight vector EWT according to
C      EWT(i) = RTOL(i)*ABS(YCUR(i)) + ATOL(i),  i = 1,...,N,
C  with the subscript on RTOL and/or ATOL possibly replaced by 1 above,
C  depending on the value of ITOL.
C
C***SEE ALSO  DLSODE
C***ROUTINES CALLED  (NONE)
C***REVISION HISTORY  (YYMMDD)
C   791129  DATE WRITTEN
C   890501  Modified prologue to SLATEC/LDOC format.  (FNF)
C   890503  Minor cosmetic changes.  (FNF)
C   930809  Renamed to allow single/double precision versions. (ACH)
C***END PROLOGUE  DEWSET
C**End
      INTEGER N, ITOL
      INTEGER I
      DOUBLE PRECISION RTOL, ATOL, YCUR, EWT
      DIMENSION RTOL(*), ATOL(*), YCUR(N), EWT(N)
C
C***FIRST EXECUTABLE STATEMENT  DEWSET
      GO TO (10, 20, 30, 40), ITOL
 10   CONTINUE
      DO 15 I = 1,N
 15     EWT(I) = RTOL(1)*ABS(YCUR(I)) + ATOL(1)
      RETURN
 20   CONTINUE
      DO 25 I = 1,N
 25     EWT(I) = RTOL(1)*ABS(YCUR(I)) + ATOL(I)
      RETURN
 30   CONTINUE
      DO 35 I = 1,N
 35     EWT(I) = RTOL(I)*ABS(YCUR(I)) + ATOL(1)
      RETURN
 40   CONTINUE
      DO 45 I = 1,N
 45     EWT(I) = RTOL(I)*ABS(YCUR(I)) + ATOL(I)
      RETURN
C----------------------- END OF SUBROUTINE DEWSET ----------------------
      END
*DECK DVNORM
      DOUBLE PRECISION FUNCTION DVNORM (N, V, W)
C***BEGIN PROLOGUE  DVNORM
C***SUBSIDIARY
C***PURPOSE  Weighted root-mean-square vector norm.
C***TYPE      DOUBLE PRECISION (SVNORM-S, DVNORM-D)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C  This function routine computes the weighted root-mean-square norm
C  of the vector of length N contained in the array V, with weights
C  contained in the array W of length N:
C    DVNORM = SQRT( (1/N) * SUM( V(i)*W(i) )**2 )
C
C***SEE ALSO  DLSODE
C***ROUTINES CALLED  (NONE)
C***REVISION HISTORY  (YYMMDD)
C   791129  DATE WRITTEN
C   890501  Modified prologue to SLATEC/LDOC format.  (FNF)
C   890503  Minor cosmetic changes.  (FNF)
C   930809  Renamed to allow single/double precision versions. (ACH)
C***END PROLOGUE  DVNORM
C**End
      INTEGER N,   I
      DOUBLE PRECISION V, W,   SUM
      DIMENSION V(N), W(N)
C
```

```
C***FIRST EXECUTABLE STATEMENT  DVNORM
      SUM = 0.0D0
      DO 10 I = 1,N
 10     SUM = SUM + (V(I)*W(I))**2
      DVNORM = SQRT(SUM/N)
      RETURN
C----------------------- END OF FUNCTION DVNORM ------------------------
      END
*DECK DSRCMS
      SUBROUTINE DSRCMS (RSAV, ISAV, JOB)
C-----------------------------------------------------------------------
C This routine saves or restores (depending on JOB) the contents of
C the Common blocks DLS001, DLSS01, which are used
C internally by one or more ODEPACK solvers.
C
C RSAV = real array of length 224 or more.
C ISAV = integer array of length 71 or more.
C JOB  = flag indicating to save or restore the Common blocks:
C        JOB  = 1 if Common is to be saved (written to RSAV/ISAV)
C        JOB  = 2 if Common is to be restored (read from RSAV/ISAV)
C        A call with JOB = 2 presumes a prior call with JOB = 1.
C-----------------------------------------------------------------------
      INTEGER ISAV, JOB
      INTEGER ILS, ILSS
      INTEGER I, LENILS, LENISS, LENRLS, LENRSS
      DOUBLE PRECISION RSAV,   RLS, RLSS
      DIMENSION RSAV(*), ISAV(*)
      SAVE LENRLS, LENILS, LENRSS, LENISS
      COMMON /DLS001/ RLS(218), ILS(37)
      COMMON /DLSS01/ RLSS(6), ILSS(34)
      DATA LENRLS/218/, LENILS/37/, LENRSS/6/, LENISS/34/
C
      IF (JOB .EQ. 2) GO TO 100
      DO 10 I = 1,LENRLS
 10     RSAV(I) = RLS(I)
      DO 15 I = 1,LENRSS
 15     RSAV(LENRLS+I) = RLSS(I)
C
      DO 20 I = 1,LENILS
 20     ISAV(I) = ILS(I)
      DO 25 I = 1,LENISS
 25     ISAV(LENILS+I) = ILSS(I)
C
      RETURN
C
 100  CONTINUE
      DO 110 I = 1,LENRLS
 110     RLS(I) = RSAV(I)
      DO 115 I = 1,LENRSS
 115     RLSS(I) = RSAV(LENRLS+I)
C
      DO 120 I = 1,LENILS
 120     ILS(I) = ISAV(I)
      DO 125 I = 1,LENISS
 125     ILSS(I) = ISAV(LENILS+I)
C
      RETURN
C----------------------- End of Subroutine DSRCMS ----------------------
      END
*DECK ODRV
      subroutine odrv
     *     (n, ia,ja,a, p,ip, nsp,isp, path, flag)
c                                                               5/2/83
c***********************************************************************
```

```
c  odrv -- driver for sparse matrix reordering routines
c***********************************************************************
c
c  description
c
c    odrv finds a minimum degree ordering of the rows and columns
c    of a matrix m stored in (ia,ja,a) format (see below).  for the
c    reordered matrix, the work and storage required to perform
c    gaussian elimination is (usually) significantly less.
c
c    note.. odrv and its subordinate routines have been modified to
c    compute orderings for general matrices, not necessarily having any
c    symmetry.  the miminum degree ordering is computed for the
c    structure of the symmetric matrix  m + m-transpose.
c    modifications to the original odrv module have been made in
c    the coding in subroutine mdi, and in the initial comments in
c    subroutines odrv and md.
c
c    if only the nonzero entries in the upper triangle of m are being
c    stored, then odrv symmetrically reorders (ia,ja,a), (optionally)
c    with the diagonal entries placed first in each row.  this is to
c    ensure that if m(i,j) will be in the upper triangle of m with
c    respect to the new ordering, then m(i,j) is stored in row i (and
c    thus m(j,i) is not stored),  whereas if m(i,j) will be in the
c    strict lower triangle of m, then m(j,i) is stored in row j (and
c    thus m(i,j) is not stored).
c
c
c  storage of sparse matrices
c
c    the nonzero entries of the matrix m are stored row-by-row in the
c    array a.  to identify the individual nonzero entries in each row,
c    we need to know in which column each entry lies.  these column
c    indices are stored in the array ja.  i.e., if  a(k) = m(i,j),  then
c    ja(k) = j.  to identify the individual rows, we need to know where
c    each row starts.  these row pointers are stored in the array ia.
c    i.e., if m(i,j) is the first nonzero entry (stored) in the i-th row
c    and  a(k) = m(i,j),  then  ia(i) = k.  moreover, ia(n+1) points to
c    the first location following the last element in the last row.
c    thus, the number of entries in the i-th row is  ia(i+1) - ia(i),
c    the nonzero entries in the i-th row are stored consecutively in
c
c            a(ia(i)),  a(ia(i)+1),  ..., a(ia(i+1)-1),
c
c    and the corresponding column indices are stored consecutively in
c
c            ja(ia(i)), ja(ia(i)+1), ..., ja(ia(i+1)-1).
c
c    when the coefficient matrix is symmetric, only the nonzero entries
c    in the upper triangle need be stored.  for example, the matrix
c
c             ( 1  0  2  3  0 )
c             ( 0  4  0  0  0 )
c         m = ( 2  0  5  6  0 )
c             ( 3  0  6  7  8 )
c             ( 0  0  0  8  9 )
c
c    could be stored as
c
c            - 1  2  3  4  5  6  7  8  9 10 11 12 13
c         ---+--------------------------------------
c         ia - 1  4  5  8 12 14
c         ja - 1  3  4  2  1  3  4  1  3  4  5  4  5
c          a - 1  2  3  4  2  5  6  3  6  7  8  8  9
```

```
c
c    or (symmetrically) as
c
c           - 1  2  3  4  5  6  7  8  9
c         ---+--------------------------
c         ia - 1  4  5  7  9 10
c         ja - 1  3  4  2  3  4  4  5  5
c          a - 1  2  3  4  5  6  7  8  9          .
c
c
c  parameters
c
c    n    - order of the matrix
c
c    ia   - integer one-dimensional array containing pointers to delimit
c           rows in ja and a.  dimension = n+1
c
c    ja   - integer one-dimensional array containing the column indices
c           corresponding to the elements of a.  dimension = number of
c           nonzero entries in (the upper triangle of) m
c
c    a    - real one-dimensional array containing the nonzero entries in
c           (the upper triangle of) m, stored by rows.  dimension =
c           number of nonzero entries in (the upper triangle of) m
c
c    p    - integer one-dimensional array used to return the permutation
c           of the rows and columns of m corresponding to the minimum
c           degree ordering.  dimension = n
c
c    ip   - integer one-dimensional array used to return the inverse of
c           the permutation returned in p.  dimension = n
c
c    nsp  - declared dimension of the one-dimensional array isp.  nsp
c           must be at least  3n+4k,  where k is the number of nonzeroes
c           in the strict upper triangle of m
c
c    isp  - integer one-dimensional array used for working storage.
c           dimension = nsp
c
c    path - integer path specification.  values and their meanings are -
c             1  find minimum degree ordering only
c             2  find minimum degree ordering and reorder symmetrically
c                  stored matrix (used when only the nonzero entries in
c                  the upper triangle of m are being stored)
c             3  reorder symmetrically stored matrix as specified by
c                  input permutation (used when an ordering has already
c                  been determined and only the nonzero entries in the
c                  upper triangle of m are being stored)
c             4  same as 2 but put diagonal entries at start of each row
c             5  same as 3 but put diagonal entries at start of each row
c
c    flag - integer error flag.  values and their meanings are -
c              0    no errors detected
c              9n+k  insufficient storage in md
c             10n+1  insufficient storage in odrv
c             11n+1  illegal path specification
c
c
c  conversion from real to double precision
c
c    change the real declarations in odrv and sro to double precision
c    declarations.
c
c-----------------------------------------------------------------------
```

```
c
      integer  ia(*), ja(*),  p(*), ip(*),  isp(*),  path,  flag,
     *   v, l, head,  tmp, q
c...  real  a(*)
      double precision  a(*)
      logical  dflag
c
c----initialize error flag and validate path specification
      flag = 0
      if (path.lt.1 .or. 5.lt.path)  go to 111
c
c----allocate storage and find minimum degree ordering
      if ((path-1) * (path-2) * (path-4) .ne. 0)  go to 1
        max = (nsp-n)/2
        v    = 1
        l    = v     +  max
        head = l     +  max
        next = head  +  n
        if (max.lt.n)  go to 110
c
        call  md
     *     (n, ia,ja, max,isp(v),isp(l), isp(head),p,ip, isp(v), flag)
        if (flag.ne.0)  go to 100
c
c----allocate storage and symmetrically reorder matrix
   1  if ((path-2) * (path-3) * (path-4) * (path-5) .ne. 0)  go to 2
        tmp = (nsp+1) -      n
        q   = tmp     - (ia(n+1)-1)
        if (q.lt.1)  go to 110
c
        dflag = path.eq.4 .or. path.eq.5
        call sro
     *     (n,  ip,  ia, ja, a,  isp(tmp),  isp(q),  dflag)
c
   2  return
c
c ** error -- error detected in md
 100  return
c ** error -- insufficient storage
 110  flag = 10*n + 1
      return
c ** error -- illegal path specified
 111  flag = 11*n + 1
      return
      end
      subroutine nnfc
     *     (n, r,c,ic, ia,ja,a, z, b,
     *      lmax,il,jl,ijl,l, d, umax,iu,ju,iju,u,
     *      row, tmp, irl,jrl, flag)
c*** subroutine nnfc
c*** numerical ldu-factorization of sparse nonsymmetric matrix and
c      solution of system of linear equations (compressed pointer
c      storage)
c
c
c       input variables..  n, r, c, ic, ia, ja, a, b,
c                          il, jl, ijl, lmax, iu, ju, iju, umax
c       output variables.. z, l, d, u, flag
c
c       parameters used internally..
c nia   - irl,  - vectors used to find the rows of  l.  at the kth step
c nia   - jrl       of the factorization,  jrl(k)  points to the head
c       -           of a linked list in  jrl  of column indices j
c       -           such j .lt. k and  l(k,j)  is nonzero.  zero
```

```
c       -           indicates the end of the list.  irl(j)  (j.lt.k)
c       -           points to the smallest i such that i .ge. k and
c       -           l(i,j)  is nonzero.
c       -           size of each = n.
c fia    - row   - holds intermediate values in calculation of  u and l.
c       -           size = n.
c fia    - tmp   - holds new right-hand side  b*  for solution of the
c       -           equation ux = b*.
c       -           size = n.
c
c  internal variables..
c    jmin, jmax - indices of the first and last positions in a row to
c      be examined.
c    sum - used in calculating  tmp.
c
      integer rk,umax
      integer  r(*), c(*), ic(*), ia(*), ja(*), il(*), jl(*), ijl(*)
      integer  iu(*), ju(*), iju(*), irl(*), jrl(*), flag
c     real  a(*), l(*), d(*), u(*), z(*), b(*), row(*)
c     real tmp(*), lki, sum, dk
      double precision  a(*), l(*), d(*), u(*), z(*), b(*), row(*)
      double precision  tmp(*), lki, sum, dk
c
c  ******  initialize pointers and test storage  ***********************
      if(il(n+1)-1 .gt. lmax) go to 104
      if(iu(n+1)-1 .gt. umax) go to 107
      do 1 k=1,n
        irl(k) = il(k)
        jrl(k) = 0
   1    continue
c
c  ******  for each row  ***********************************************
      do 19 k=1,n
c  ******  reverse jrl and zero row where kth row of l will fill in  ***
        row(k) = 0
        i1 = 0
        if (jrl(k) .eq. 0) go to 3
        i = jrl(k)
   2    i2 = jrl(i)
        jrl(i) = i1
        i1 = i
        row(i) = 0
        i = i2
        if (i .ne. 0) go to 2
c  ******  set row to zero where u will fill in  ***********************
   3    jmin = iju(k)
        jmax = jmin + iu(k+1) - iu(k) - 1
        if (jmin .gt. jmax) go to 5
        do 4 j=jmin,jmax
   4      row(ju(j)) = 0
c  ******  place kth row of a in row  **********************************
   5    rk = r(k)
        jmin = ia(rk)
        jmax = ia(rk+1) - 1
        do 6 j=jmin,jmax
          row(ic(ja(j))) = a(j)
   6      continue
c  ******  initialize sum, and link through jrl  ***********************
        sum = b(rk)
        i = i1
        if (i .eq. 0) go to 10
c  ******  assign the kth row of l and adjust row, sum  ****************
   7      lki = -row(i)
c  ******  if l is not required, then comment out the following line  **
```

```
          l(irl(i)) = -lki
          sum = sum + lki * tmp(i)
          jmin = iu(i)
          jmax = iu(i+1) - 1
          if (jmin .gt. jmax) go to 9
          mu = iju(i) - jmin
          do 8 j=jmin,jmax
   8        row(ju(mu+j)) = row(ju(mu+j)) + lki * u(j)
   9      i = jrl(i)
          if (i .ne. 0) go to 7
c
c  ******  assign kth row of u and diagonal d, set tmp(k)  *************
  10    if (row(k) .eq. 0.0d0) go to 108
        dk = 1.0d0 / row(k)
        d(k) = dk
        tmp(k) = sum * dk
        if (k .eq. n) go to 19
        jmin = iu(k)
        jmax = iu(k+1) - 1
        if (jmin .gt. jmax)  go to 12
        mu = iju(k) - jmin
        do 11 j=jmin,jmax
  11      u(j) = row(ju(mu+j)) * dk
  12    continue
c
c  ******  update irl and jrl, keeping jrl in decreasing order  ********
        i = i1
        if (i .eq. 0) go to 18
  14    irl(i) = irl(i) + 1
        i1 = jrl(i)
        if (irl(i) .ge. il(i+1)) go to 17
        ijlb = irl(i) - il(i) + ijl(i)
        j = jl(ijlb)
  15    if (i .gt. jrl(j)) go to 16
          j = jrl(j)
          go to 15
  16    jrl(i) = jrl(j)
        jrl(j) = i
  17    i = i1
        if (i .ne. 0) go to 14
  18    if (irl(k) .ge. il(k+1)) go to 19
        j = jl(ijl(k))
        jrl(k) = jrl(j)
        jrl(j) = k
  19    continue
c
c  ******  solve  ux = tmp  by back substitution  **********************
      k = n
      do 22 i=1,n
        sum =  tmp(k)
        jmin = iu(k)
        jmax = iu(k+1) - 1
        if (jmin .gt. jmax)  go to 21
        mu = iju(k) - jmin
        do 20 j=jmin,jmax
  20      sum = sum - u(j) * tmp(ju(mu+j))
  21    tmp(k) =  sum
        z(c(k)) =  sum
  22    k = k-1
      flag = 0
      return
c
c ** error.. insufficient storage for l
 104  flag = 4*n + 1
```

```
      return
c ** error.. insufficient storage for u
 107  flag = 7*n + 1
      return
c ** error.. zero pivot
 108  flag = 8*n + k
      return
      end
      subroutine nntc
     *     (n, r, c, il, jl, ijl, l, d, iu, ju, iju, u, z, b, tmp)
c*** subroutine nntc
c*** numeric solution of the transpose of a sparse nonsymmetric system
c      of linear equations given lu-factorization (compressed pointer
c      storage)
c
c
c       input variables..  n, r, c, il, jl, ijl, l, d, iu, ju, iju, u, b
c       output variables.. z
c
c       parameters used internally..
c fia   - tmp   - temporary vector which gets result of solving ut y = b
c       -           size = n.
c
c  internal variables..
c    jmin, jmax - indices of the first and last positions in a row of
c      u or l  to be used.
c
      integer r(*), c(*), il(*), jl(*), ijl(*), iu(*), ju(*), iju(*)
c     real l(*), d(*), u(*), b(*), z(*), tmp(*), tmpk,sum
      double precision l(*), d(*), u(*), b(*), z(*), tmp(*), tmpk,sum
c
c  ******  set tmp to reordered b  ************************************
      do 1 k=1,n
   1    tmp(k) = b(c(k))
c  ******  solve  ut y = b  by forward substitution  ******************
      do 3 k=1,n
        jmin = iu(k)
        jmax = iu(k+1) - 1
        tmpk = -tmp(k)
        if (jmin .gt. jmax) go to 3
        mu = iju(k) - jmin
        do 2 j=jmin,jmax
   2      tmp(ju(mu+j)) = tmp(ju(mu+j)) + tmpk * u(j)
   3    continue
c  ******  solve  lt x = y  by back substitution  *********************
      k = n
      do 6 i=1,n
        sum = -tmp(k)
        jmin = il(k)
        jmax = il(k+1) - 1
        if (jmin .gt. jmax) go to 5
        ml = ijl(k) - jmin
        do 4 j=jmin,jmax
   4      sum = sum + l(j) * tmp(jl(ml+j))
   5    tmp(k) = -sum * d(k)
        z(r(k)) = tmp(k)
        k = k - 1
   6    continue
      return
      end
      subroutine nsfc
     *      (n, r, ic, ia,ja, jlmax,il,jl,ijl, jumax,iu,ju,iju,
     *       q, ira,jra, irac, irl,jrl, iru,jru, flag)
c*** subroutine nsfc
```

```
c*** symbolic ldu-factorization of nonsymmetric sparse matrix
c      (compressed pointer storage)
c
c
c       input variables.. n, r, ic, ia, ja, jlmax, jumax.
c       output variables.. il, jl, ijl, iu, ju, iju, flag.
c
c       parameters used internally..
c nia   - q     - suppose  m*  is the result of reordering  m.  if
c       -           processing of the ith row of  m*  (hence the ith
c       -           row of  u) is being done,  q(j)  is initially
c       -           nonzero if  m*(i,j) is nonzero (j.ge.i).  since
c       -           values need not be stored, each entry points to the
c       -           next nonzero and  q(n+1)  points to the first.  n+1
c       -           indicates the end of the list.  for example, if n=9
c       -           and the 5th row of  m*  is
c       -              0 x x 0 x 0 0 x 0
c       -           then  q  will initially be
c       -              a a a a 8 a a 10 5           (a - arbitrary).
c       -           as the algorithm proceeds, other elements of  q
c       -           are inserted in the list because of fillin.
c       -           q  is used in an analogous manner to compute the
c       -           ith column of  l.
c       -           size = n+1.
c nia   - ira,  - vectors used to find the columns of  m.  at the kth
c nia   - jra,      step of the factorization,  irac(k)  points to the
c nia   - irac      head of a linked list in  jra  of row indices i
c       -           such that i .ge. k and  m(i,k)  is nonzero.  zero
c       -           indicates the end of the list.  ira(i)  (i.ge.k)
c       -           points to the smallest j such that j .ge. k and
c       -           m(i,j)  is nonzero.
c       -           size of each = n.
c nia   - irl,  - vectors used to find the rows of  l.  at the kth step
c nia   - jrl       of the factorization,  jrl(k)  points to the head
c       -           of a linked list in  jrl  of column indices j
c       -           such j .lt. k and  l(k,j)  is nonzero.  zero
c       -           indicates the end of the list.  irl(j)  (j.lt.k)
c       -           points to the smallest i such that i .ge. k and
c       -           l(i,j)  is nonzero.
c       -           size of each = n.
c nia   - iru,  - vectors used in a manner analogous to  irl and jrl
c nia   - jru       to find the columns of  u.
c       -           size of each = n.
c
c  internal variables..
c    jlptr - points to the last position used in  jl.
c    juptr - points to the last position used in  ju.
c    jmin,jmax - are the indices in  a or u  of the first and last
c                elements to be examined in a given row.
c                for example,  jmin=ia(k), jmax=ia(k+1)-1.
c
      integer cend, qm, rend, rk, vj
      integer ia(*), ja(*), ira(*), jra(*), il(*), jl(*), ijl(*)
      integer iu(*), ju(*), iju(*), irl(*), jrl(*), iru(*), jru(*)
      integer r(*), ic(*), q(*), irac(*), flag
c
c  ******  initialize pointers  ****************************************
      np1 = n + 1
      jlmin = 1
      jlptr = 0
      il(1) = 1
      jumin = 1
      juptr = 0
      iu(1) = 1
      do 1 k=1,n
        irac(k) = 0
        jra(k) = 0
        jrl(k) = 0
   1    jru(k) = 0
c  ******  initialize column pointers for a  ***************************
      do 2 k=1,n
        rk = r(k)
        iak = ia(rk)
        if (iak .ge. ia(rk+1))  go to 101
        jaiak = ic(ja(iak))
        if (jaiak .gt. k)  go to 105
        jra(k) = irac(jaiak)
        irac(jaiak) = k
   2    ira(k) = iak
c
c  ******  for each column of l and row of u  **************************
      do 41 k=1,n
c
c  ******  initialize q for computing kth column of l  *****************
        q(np1) = np1
        luk = -1
c  ******  by filling in kth column of a  ******************************
        vj = irac(k)
        if (vj .eq. 0)  go to 5
   3      qm = np1
   4      m = qm
          qm =  q(m)
          if (qm .lt. vj)  go to 4
          if (qm .eq. vj)  go to 102
            luk = luk + 1
            q(m) = vj
            q(vj) = qm
            vj = jra(vj)
            if (vj .ne. 0)  go to 3
c  ******  link through jru  *******************************************
   5    lastid = 0
        lasti = 0
        ijl(k) = jlptr
        i = k
   6      i = jru(i)
          if (i .eq. 0)  go to 10
          qm = np1
          jmin = irl(i)
          jmax = ijl(i) + il(i+1) - il(i) - 1
          long = jmax - jmin
          if (long .lt. 0)  go to 6
          jtmp = jl(jmin)
          if (jtmp .ne. k)  long = long + 1
          if (jtmp .eq. k)  r(i) = -r(i)
          if (lastid .ge. long)  go to 7
            lasti = i
            lastid = long
c  ******  and merge the corresponding columns into the kth column  ****
   7      do 9 j=jmin,jmax
            vj = jl(j)
   8        m = qm
            qm = q(m)
            if (qm .lt. vj)  go to 8
            if (qm .eq. vj)  go to 9
              luk = luk + 1
              q(m) = vj
              q(vj) = qm
              qm = vj
   9        continue
            go to 6
c  ******  lasti is the longest column merged into the kth  ************
c  ******  see if it equals the entire kth column  ********************
  10    qm = q(np1)
        if (qm .ne. k)  go to 105
        if (luk .eq. 0)  go to 17
        if (lastid .ne. luk)  go to 11
c  ******  if so, jl can be compressed  *******************************
        irll = irl(lasti)
        ijl(k) = irll + 1
        if (jl(irll) .ne. k)  ijl(k) = ijl(k) - 1
        go to 17
c  ******  if not, see if kth column can overlap the previous one  *****
  11    if (jlmin .gt. jlptr)  go to 15
        qm = q(qm)
        do 12 j=jlmin,jlptr
          if (jl(j) - qm)  12, 13, 15
  12      continue
        go to 15
  13    ijl(k) = j
        do 14 i=j,jlptr
          if (jl(i) .ne. qm)  go to 15
          qm = q(qm)
          if (qm .gt. n)  go to 17
  14      continue
        jlptr = j - 1
c  ******  move column indices from q to jl, update vectors  ***********
  15    jlmin = jlptr + 1
        ijl(k) = jlmin
        if (luk .eq. 0)  go to 17
        jlptr = jlptr + luk
        if (jlptr .gt. jlmax)  go to 103
          qm = q(np1)
          do 16 j=jlmin,jlptr
            qm = q(qm)
  16        jl(j) = qm
  17    irl(k) = ijl(k)
        il(k+1) = il(k) + luk
c
c  ******  initialize q for computing kth row of u  *******************
        q(np1) = np1
        luk = -1
c  ******  by filling in kth row of reordered a  **********************
        rk = r(k)
        jmin = ira(k)
        jmax = ia(rk+1) - 1
        if (jmin .gt. jmax)  go to 20
        do 19 j=jmin,jmax
          vj = ic(ja(j))
          qm = np1
  18      m = qm
          qm = q(m)
          if (qm .lt. vj)  go to 18
          if (qm .eq. vj)  go to 102
            luk = luk + 1
            q(m) = vj
            q(vj) = qm
  19      continue
c  ******  link through jrl,  ******************************************
  20    lastid = 0
        lasti = 0
        iju(k) = juptr
        i = k
        i1 = jrl(k)
  21      i = i1
          if (i .eq. 0)  go to 26
          i1 = jrl(i)
          qm = np1
          jmin = iru(i)
          jmax = iju(i) + iu(i+1) - iu(i) - 1
          long = jmax - jmin
          if (long .lt. 0)  go to 21
          jtmp = ju(jmin)
          if (jtmp .eq. k)  go to 22
c  ******  update irl and jrl, *****************************************
            long = long + 1
            cend = ijl(i) + il(i+1) - il(i)
            irl(i) = irl(i) + 1
            if (irl(i) .ge. cend)  go to 22
              j = jl(irl(i))
              jrl(i) = jrl(j)
              jrl(j) = i
  22      if (lastid .ge. long)  go to 23
            lasti = i
            lastid = long
c  ******  and merge the corresponding rows into the kth row  **********
  23      do 25 j=jmin,jmax
            vj = ju(j)
  24        m = qm
            qm = q(m)
            if (qm .lt. vj)  go to 24
            if (qm .eq. vj)  go to 25
              luk = luk + 1
              q(m) = vj
              q(vj) = qm
              qm = vj
  25        continue
          go to 21
c  ******  update jrl(k) and irl(k)  ***********************************
  26    if (il(k+1) .le. il(k))  go to 27
          j = jl(irl(k))
          jrl(k) = jrl(j)
          jrl(j) = k
c  ******  lasti is the longest row merged into the kth  ***************
c  ******  see if it equals the entire kth row  ***********************
  27    qm = q(np1)
        if (qm .ne. k)  go to 105
        if (luk .eq. 0)  go to 34
        if (lastid .ne. luk)  go to 28
c  ******  if so, ju can be compressed  *******************************
        irul = iru(lasti)
        iju(k) = irul + 1
        if (ju(irul) .ne. k)  iju(k) = iju(k) - 1
        go to 34
c  ******  if not, see if kth row can overlap the previous one  ********
  28    if (jumin .gt. juptr)  go to 32
        qm = q(qm)
        do 29 j=jumin,juptr
          if (ju(j) - qm)  29, 30, 32
  29      continue
        go to 32
  30    iju(k) = j
        do 31 i=j,juptr
          if (ju(i) .ne. qm)  go to 32
          qm = q(qm)
          if (qm .gt. n)  go to 34
  31      continue
```

```
        juptr = j - 1
c  ******  move row indices from q to ju, update vectors  **************
  32    jumin = juptr + 1
        iju(k) = jumin
        if (luk .eq. 0)  go to 34
        juptr = juptr + luk
        if (juptr .gt. jumax)  go to 106
          qm = q(np1)
          do 33 j=jumin,juptr
            qm = q(qm)
  33        ju(j) = qm
  34    iru(k) = iju(k)
        iu(k+1) = iu(k) + luk
c
c  ******  update iru, jru  ********************************************
        i = k
  35      i1 = jru(i)
          if (r(i) .lt. 0)  go to 36
          rend = iju(i) + iu(i+1) - iu(i)
          if (iru(i) .ge. rend)  go to 37
            j = ju(iru(i))
            jru(i) = jru(j)
            jru(j) = i
            go to 37
  36      r(i) = -r(i)
  37      i = i1
          if (i .eq. 0)  go to 38
          iru(i) = iru(i) + 1
          go to 35
c
c  ******  update ira, jra, irac  **************************************
  38    i = irac(k)
        if (i .eq. 0)  go to 41
  39      i1 = jra(i)
          ira(i) = ira(i) + 1
          if (ira(i) .ge. ia(r(i)+1))  go to 40
          irai = ira(i)
          jairai = ic(ja(irai))
          if (jairai .gt. i)  go to 40
          jra(i) = irac(jairai)
          irac(jairai) = i
  40      i = i1
          if (i .ne. 0)  go to 39
  41    continue
c
      ijl(n) = jlptr
      iju(n) = juptr
      flag = 0
      return
c
c ** error.. null row in a
 101  flag = n + rk
      return
c ** error.. duplicate entry in a
 102  flag = 2*n + rk
      return
c ** error.. insufficient storage for jl
 103  flag = 3*n + k
      return
c ** error.. null pivot
 105  flag = 5*n + k
      return
c ** error.. insufficient storage for ju
 106  flag = 6*n + k
```

```
      return
      end
      subroutine nnsc
     *     (n, r, c, il, jl, ijl, l, d, iu, ju, iju, u, z, b, tmp)
c*** subroutine nnsc
c*** numerical solution of sparse nonsymmetric system of linear
c      equations given ldu-factorization (compressed pointer storage)
c
c
c       input variables..  n, r, c, il, jl, ijl, l, d, iu, ju, iju, u, b
c       output variables.. z
c
c       parameters used internally..
c fia   - tmp   - temporary vector which gets result of solving  ly = b.
c       -           size = n.
c
c  internal variables..
c    jmin, jmax - indices of the first and last positions in a row of
c      u or l  to be used.
c
      integer r(*), c(*), il(*), jl(*), ijl(*), iu(*), ju(*), iju(*)
c     real l(*), d(*), u(*), b(*), z(*), tmp(*), tmpk, sum
      double precision  l(*), d(*), u(*), b(*), z(*), tmp(*), tmpk,sum
c
c  ******  set tmp to reordered b  *************************************
      do 1 k=1,n
   1    tmp(k) = b(r(k))
c  ******  solve  ly = b  by forward substitution  ********************
      do 3 k=1,n
        jmin = il(k)
        jmax = il(k+1) - 1
        tmpk = -d(k) * tmp(k)
        tmp(k) = -tmpk
        if (jmin .gt. jmax) go to 3
        ml = ijl(k) - jmin
        do 2 j=jmin,jmax
   2      tmp(jl(ml+j)) = tmp(jl(ml+j)) + tmpk * l(j)
   3    continue
c  ******  solve  ux = y  by back substitution  ***********************
      k = n
      do 6 i=1,n
        sum = -tmp(k)
        jmin = iu(k)
        jmax = iu(k+1) - 1
        if (jmin .gt. jmax) go to 5
        mu = iju(k) - jmin
        do 4 j=jmin,jmax
   4      sum = sum + u(j) * tmp(ju(mu+j))
   5    tmp(k) = -sum
        z(c(k)) = -sum
        k = k - 1
   6    continue
      return
      end
      subroutine nroc (n, ic, ia, ja, a, jar, ar, p, flag)
c
c       ---------------------------------------------------------------
c
c               yale sparse matrix package - nonsymmetric codes
c                    solving the system of equations mx = b
c
c    i.   calling sequences
c         the coefficient matrix can be processed by an ordering routine
c    (e.g., to reduce fillin or ensure numerical stability) before using
```

```
c     the remaining subroutines.  if no reordering is done, then set
c     r(i) = c(i) = ic(i) = i  for i=1,...,n.  if an ordering subroutine
c     is used, then nroc should be used to reorder the coefficient matrix
c     the calling sequence is --
c         (       (matrix ordering))
c         (nroc   (matrix reordering))
c          nsfc   (symbolic factorization to determine where fillin will
c                   occur during numeric factorization)
c          nnfc   (numeric factorization into product ldu of unit lower
c                   triangular matrix l, diagonal matrix d, and unit
c                   upper triangular matrix u, and solution of linear
c                   system)
c          nnsc   (solution of linear system for additional right-hand
c                   side using ldu factorization from nnfc)
c     (if only one system of equations is to be solved, then the
c     subroutine trk should be used.)
c
c     ii.  storage of sparse matrices
c          the nonzero entries of the coefficient matrix m are stored
c     row-by-row in the array a.  to identify the individual nonzero
c     entries in each row, we need to know in which column each entry
c     lies.  the column indices which correspond to the nonzero entries
c     of m are stored in the array ja.  i.e., if  a(k) = m(i,j),  then
c     ja(k) = j.  in addition, we need to know where each row starts and
c     how long it is.  the index positions in ja and a where the rows of
c     m begin are stored in the array ia.  i.e., if m(i,j) is the first
c     (leftmost) entry in the i-th row and  a(k) = m(i,j),  then
c     ia(i) = k.  moreover, the index in ja and a of the first location
c     following the last element in the last row is stored in ia(n+1).
c     thus, the number of entries in the i-th row is given by
c     ia(i+1) - ia(i),  the nonzero entries of the i-th row are stored
c     consecutively in
c             a(ia(i)),  a(ia(i)+1),  ..., a(ia(i+1)-1),
c     and the corresponding column indices are stored consecutively in
c             ja(ia(i)), ja(ia(i)+1), ..., ja(ia(i+1)-1).
c     for example, the 5 by 5 matrix
c                 ( 1. 0. 2. 0. 0.)
c                 ( 0. 3. 0. 0. 0.)
c             m = ( 0. 4. 5. 6. 0.)
c                 ( 0. 0. 0. 7. 0.)
c                 ( 0. 0. 0. 8. 9.)
c     would be stored as
c                - 1  2  3  4  5  6  7  8  9
c             ---+--------------------------
c             ia - 1  3  4  7  8 10
c             ja - 1  3  2  2  3  4  4  4  5
c              a - 1. 2. 3. 4. 5. 6. 7. 8. 9.         .
c
c          the strict upper (lower) triangular portion of the matrix
c     u (l) is stored in a similar fashion using the arrays  iu, ju, u
c     (il, jl, l)  except that an additional array iju (ijl) is used to
c     compress storage of ju (jl) by allowing some sequences of column
c     (row) indices to used for more than one row (column)  (n.b., l is
c     stored by columns).  iju(k) (ijl(k)) points to the starting
c     location in ju (jl) of entries for the kth row (column).
c     compression in ju (jl) occurs in two ways.  first, if a row
c     (column) i was merged into the current row (column) k, and the
c     number of elements merged in from (the tail portion of) row
c     (column) i is the same as the final length of row (column) k, then
c     the kth row (column) and the tail of row (column) i are identical
c     and iju(k) (ijl(k)) points to the start of the tail.  second, if
c     some tail portion of the (k-1)st row (column) is identical to the
c     head of the kth row (column), then iju(k) (ijl(k)) points to the
c     start of that tail portion.  for example, the nonzero structure of
```

```
c    the strict upper triangular part of the matrix
c            d 0 x x x
c            0 d 0 x x
c            0 0 d x 0
c            0 0 0 d x
c            0 0 0 0 d
c    would be represented as
c                - 1 2 3 4 5 6
c            ----+-----------
c             iu - 1 4 6 7 8 8
c             ju - 3 4 5 4
c            iju - 1 2 4 3           .
c    the diagonal entries of l and u are assumed to be equal to one and
c    are not stored.  the array d contains the reciprocals of the
c    diagonal entries of the matrix d.
c
c    iii. additional storage savings
c         in nsfc, r and ic can be the same array in the calling
c    sequence if no reordering of the coefficient matrix has been done.
c         in nnfc, r, c, and ic can all be the same array if no
c    reordering has been done.  if only the rows have been reordered,
c    then c and ic can be the same array.  if the row and column
c    orderings are the same, then r and c can be the same array.  z and
c    row can be the same array.
c         in nnsc or nntc, r and c can be the same array if no
c    reordering has been done or if the row and column orderings are the
c    same.  z and b can be the same array.  however, then b will be
c    destroyed.
c
c    iv.  parameters
c         following is a list of parameters to the programs.  names are
c    uniform among the various subroutines.  class abbreviations are --
c       n - integer variable
c       f - real variable
c       v - supplies a value to a subroutine
c       r - returns a result from a subroutine
c       i - used internally by a subroutine
c       a - array
c
c class - parameter
c ------+----------
c fva   - a     - nonzero entries of the coefficient matrix m, stored
c       -           by rows.
c       -           size = number of nonzero entries in m.
c fva   - b     - right-hand side b.
c       -           size = n.
c nva   - c     - ordering of the columns of m.
c       -           size = n.
c fvra  - d     - reciprocals of the diagonal entries of the matrix d.
c       -           size = n.
c nr    - flag  - error flag.  values and their meanings are --
c       -            0     no errors detected
c       -            n+k   null row in a  --  row = k
c       -           2n+k   duplicate entry in a  --  row = k
c       -           3n+k   insufficient storage for jl  --  row = k
c       -           4n+1   insufficient storage for l
c       -           5n+k   null pivot  --  row = k
c       -           6n+k   insufficient storage for ju  --  row = k
c       -           7n+1   insufficient storage for u
c       -           8n+k   zero pivot  --  row = k
c nva   - ia    - pointers to delimit the rows of a.
c       -           size = n+1.
c nvra  - ijl   - pointers to the first element in each column in jl,
c       -           used to compress storage in jl.
```

```
c       -           size = n.
c nvra  - iju   - pointers to the first element in each row in ju, used
c       -           to compress storage in ju.
c       -           size = n.
c nvra  - il    - pointers to delimit the columns of l.
c       -           size = n+1.
c nvra  - iu    - pointers to delimit the rows of u.
c       -           size = n+1.
c nva   - ja    - column numbers corresponding to the elements of a.
c       -           size = size of a.
c nvra  - jl    - row numbers corresponding to the elements of l.
c       -           size = jlmax.
c nv    - jlmax - declared dimension of jl.  jlmax must be larger than
c       -           the number of nonzeros in the strict lower triangle
c       -           of m plus fillin minus compression.
c nvra  - ju    - column numbers corresponding to the elements of u.
c       -           size = jumax.
c nv    - jumax - declared dimension of ju.  jumax must be larger than
c       -           the number of nonzeros in the strict upper triangle
c       -           of m plus fillin minus compression.
c fvra  - l     - nonzero entries in the strict lower triangular portion
c       -           of the matrix l, stored by columns.
c       -           size = lmax.
c nv    - lmax  - declared dimension of l.  lmax must be larger than
c       -           the number of nonzeros in the strict lower triangle
c       -           of m plus fillin  (il(n+1)-1 after nsfc).
c nv    - n     - number of variables/equations.
c nva   - r     - ordering of the rows of m.
c       -           size = n.
c fvra  - u     - nonzero entries in the strict upper triangular portion
c       -           of the matrix u, stored by rows.
c       -           size = umax.
c nv    - umax  - declared dimension of u.  umax must be larger than
c       -           the number of nonzeros in the strict upper triangle
c       -           of m plus fillin  (iu(n+1)-1 after nsfc).
c fra   - z     - solution x.
c       -           size = n.
c
c       ----------------------------------------------------------------
c
c*** subroutine nroc
c*** reorders rows of a, leaving row order unchanged
c
c
c       input parameters.. n, ic, ia, ja, a
c       output parameters.. ja, a, flag
c
c       parameters used internally..
c nia   - p     - at the kth step, p is a linked list of the reordered
c       -           column indices of the kth row of a.  p(n+1) points
c       -           to the first entry in the list.
c       -           size = n+1.
c nia   - jar   - at the kth step,jar contains the elements of the
c       -           reordered column indices of a.
c       -           size = n.
c fia   - ar    - at the kth step, ar contains the elements of the
c       -           reordered row of a.
c       -           size = n.
c
      integer  ic(*), ia(*), ja(*), jar(*), p(*), flag
c     real  a(*), ar(*)
      double precision  a(*), ar(*)
c
c  ******  for each nonempty row  *******************************
```

```
      do 5 k=1,n
        jmin = ia(k)
        jmax = ia(k+1) - 1
        if(jmin .gt. jmax) go to 5
        p(n+1) = n + 1
c  ******  insert each element in the list  *********************
        do 3 j=jmin,jmax
          newj = ic(ja(j))
          i = n + 1
   1      if(p(i) .ge. newj) go to 2
            i = p(i)
            go to 1
   2      if(p(i) .eq. newj) go to 102
          p(newj) = p(i)
          p(i) = newj
          jar(newj) = ja(j)
          ar(newj) = a(j)
   3      continue
c  ******  replace old row in ja and a  *************************
        i = n + 1
        do 4 j=jmin,jmax
          i = p(i)
          ja(j) = jar(i)
   4      a(j) = ar(i)
   5    continue
      flag = 0
      return
c
c ** error.. duplicate entry in a
 102  flag = n + k
      return
      end
*DECK CDRV
      subroutine cdrv
     *     (n, r,c,ic, ia,ja,a, b, z, nsp,isp,rsp,esp, path, flag)
c*** subroutine cdrv
c*** driver for subroutines for solving sparse nonsymmetric systems of
c       linear equations (compressed pointer storage)
c
c
c    parameters
c    class abbreviations are--
c       n - integer variable
c       f - real variable
c       v - supplies a value to the driver
c       r - returns a result from the driver
c       i - used internally by the driver
c       a - array
c
c class - parameter
c ------+----------
c       -
c         the nonzero entries of the coefficient matrix m are stored
c    row-by-row in the array a.  to identify the individual nonzero
c    entries in each row, we need to know in which column each entry
c    lies.  the column indices which correspond to the nonzero entries
c    of m are stored in the array ja.  i.e., if  a(k) = m(i,j),  then
c    ja(k) = j.  in addition, we need to know where each row starts and
c    how long it is.  the index positions in ja and a where the rows of
c    m begin are stored in the array ia.  i.e., if m(i,j) is the first
c    nonzero entry (stored) in the i-th row and a(k) = m(i,j),  then
c    ia(i) = k.  moreover, the index in ja and a of the first location
c    following the last element in the last row is stored in ia(n+1).
c    thus, the number of entries in the i-th row is given by
```

```
c    ia(i+1) - ia(i),  the nonzero entries of the i-th row are stored
c    consecutively in
c            a(ia(i)),  a(ia(i)+1),  ..., a(ia(i+1)-1),
c    and the corresponding column indices are stored consecutively in
c            ja(ia(i)), ja(ia(i)+1), ..., ja(ia(i+1)-1).
c    for example, the 5 by 5 matrix
c                ( 1. 0. 2. 0. 0.)
c                ( 0. 3. 0. 0. 0.)
c            m = ( 0. 4. 5. 6. 0.)
c                ( 0. 0. 0. 7. 0.)
c                ( 0. 0. 0. 8. 9.)
c    would be stored as
c               - 1  2  3  4  5  6  7  8  9
c            ---+--------------------------
c            ia - 1  3  4  7  8 10
c            ja - 1  3  2  2  3  4  4  4  5
c             a - 1. 2. 3. 4. 5. 6. 7. 8. 9.         .
c
c nv    - n     - number of variables/equations.
c fva   - a     - nonzero entries of the coefficient matrix m, stored
c       -           by rows.
c       -           size = number of nonzero entries in m.
c nva   - ia    - pointers to delimit the rows in a.
c       -           size = n+1.
c nva   - ja    - column numbers corresponding to the elements of a.
c       -           size = size of a.
c fva   - b     - right-hand side b.  b and z can the same array.
c       -           size = n.
c fra   - z     - solution x.  b and z can be the same array.
c       -           size = n.
c
c         the rows and columns of the original matrix m can be
c    reordered (e.g., to reduce fillin or ensure numerical stability)
c    before calling the driver.  if no reordering is done, then set
c    r(i) = c(i) = ic(i) = i  for i=1,...,n.  the solution z is returned
c    in the original order.
c         if the columns have been reordered (i.e.,  c(i).ne.i  for some
c    i), then the driver will call a subroutine (nroc) which rearranges
c    each row of ja and a, leaving the rows in the original order, but
c    placing the elements of each row in increasing order with respect
c    to the new ordering.  if  path.ne.1,  then nroc is assumed to have
c    been called already.
c
c nva   - r     - ordering of the rows of m.
c       -           size = n.
c nva   - c     - ordering of the columns of m.
c       -           size = n.
c nva   - ic    - inverse of the ordering of the columns of m.  i.e.,
c       -           ic(c(i)) = i  for i=1,...,n.
c       -           size = n.
c
c         the solution of the system of linear equations is divided into
c    three stages --
c      nsfc -- the matrix m is processed symbolically to determine where
c               fillin will occur during the numeric factorization.
c      nnfc -- the matrix m is factored numerically into the product ldu
c               of a unit lower triangular matrix l, a diagonal matrix
c               d, and a unit upper triangular matrix u, and the system
c               mx = b  is solved.
c      nnsc -- the linear system  mx = b  is solved using the ldu
c  or           factorization from nnfc.
c      nntc -- the transposed linear system  mt x = b  is solved using
c               the ldu factorization from nnf.
c    for several systems whose coefficient matrices have the same
```

```
c    nonzero structure, nsfc need be done only once (for the first
c    system).  then nnfc is done once for each additional system.  for
c    several systems with the same coefficient matrix, nsfc and nnfc
c    need be done only once (for the first system).  then nnsc or nntc
c    is done once for each additional right-hand side.
c
c nv    - path  - path specification.  values and their meanings are --
c       -           1  perform nroc, nsfc, and nnfc.
c       -           2  perform nnfc only  (nsfc is assumed to have been
c       -               done in a manner compatible with the storage
c       -               allocation used in the driver).
c       -           3  perform nnsc only  (nsfc and nnfc are assumed to
c       -               have been done in a manner compatible with the
c       -               storage allocation used in the driver).
c       -           4  perform nntc only  (nsfc and nnfc are assumed to
c       -               have been done in a manner compatible with the
c       -               storage allocation used in the driver).
c       -           5  perform nroc and nsfc.
c
c         various errors are detected by the driver and the individual
c    subroutines.
c
c nr    - flag  - error flag.  values and their meanings are --
c       -             0     no errors detected
c       -             n+k   null row in a  --  row = k
c       -            2n+k   duplicate entry in a  --  row = k
c       -            3n+k   insufficient storage in nsfc  --  row = k
c       -            4n+1   insufficient storage in nnfc
c       -            5n+k   null pivot  --  row = k
c       -            6n+k   insufficient storage in nsfc  --  row = k
c       -            7n+1   insufficient storage in nnfc
c       -            8n+k   zero pivot  --  row = k
c       -           10n+1   insufficient storage in cdrv
c       -           11n+1   illegal path specification
c
c         working storage is needed for the factored form of the matrix
c    m plus various temporary vectors.  the arrays isp and rsp should be
c    equivalenced.  integer storage is allocated from the beginning of
c    isp and real storage from the end of rsp.
c
c nv    - nsp   - declared dimension of rsp.  nsp generally must
c       -           be larger than  8n+2 + 2k  (where  k = (number of
c       -           nonzero entries in m)).
c nvira - isp   - integer working storage divided up into various arrays
c       -           needed by the subroutines.  isp and rsp should be
c       -           equivalenced.
c       -           size = lratio*nsp.
c fvira - rsp   - real working storage divided up into various arrays
c       -           needed by the subroutines.  isp and rsp should be
c       -           equivalenced.
c       -           size = nsp.
c nr    - esp   - if sufficient storage was available to perform the
c       -           symbolic factorization (nsfc), then esp is set to
c       -           the amount of excess storage provided (negative if
c       -           insufficient storage was available to perform the
c       -           numeric factorization (nnfc)).
c
c
c  conversion to double precision
c
c    to convert these routines for double precision arrays..
c    (1) use the double precision declarations in place of the real
c    declarations in each subprogram, as given in comment cards.
c    (2) change the data-loaded value of the integer  lratio
```

```
c    in subroutine cdrv, as indicated below.
c    (3) change e0 to d0 in the constants in statement number 10
c    in subroutine nnfc and the line following that.
c
      integer  r(*), c(*), ic(*),  ia(*), ja(*),  isp(*), esp,  path,
     *   flag,  d, u, q, row, tmp, ar,  umax
c     real  a(*), b(*), z(*), rsp(*)
      double precision  a(*), b(*), z(*), rsp(*)
c
c  set lratio equal to the ratio between the length of floating point
c  and integer array data.  e. g., lratio = 1 for (real, integer),
c  lratio = 2 for (double precision, integer)
c
      data lratio/2/
c
      if (path.lt.1 .or. 5.lt.path)  go to 111
c******initialize and divide up temporary storage  *******************
      il   = 1
      ijl  = il  + (n+1)
      iu   = ijl +   n
      iju  = iu  + (n+1)
      irl  = iju +   n
      jrl  = irl +   n
      jl   = jrl +   n
c
c  ******  reorder a if necessary, call nsfc if flag is set  ***********
      if ((path-1) * (path-5) .ne. 0)  go to 5
        max = (lratio*nsp + 1 - jl) - (n+1) - 5*n
        jlmax = max/2
        q     = jl   + jlmax
        ira   = q    + (n+1)
        jra   = ira  +   n
        irac  = jra  +   n
        iru   = irac +   n
        jru   = iru  +   n
        jutmp = jru  +   n
        jumax = lratio*nsp  + 1 - jutmp
        esp = max/lratio
        if (jlmax.le.0 .or. jumax.le.0)  go to 110
c
        do 1 i=1,n
          if (c(i).ne.i)  go to 2
   1      continue
        go to 3
   2    ar = nsp + 1 - n
        call  nroc
     *     (n, ic, ia,ja,a, isp(il), rsp(ar), isp(iu), flag)
        if (flag.ne.0)  go to 100
c
   3    call  nsfc
     *     (n, r, ic, ia,ja,
     *      jlmax, isp(il), isp(jl), isp(ijl),
     *      jumax, isp(iu), isp(jutmp), isp(iju),
     *      isp(q), isp(ira), isp(jra), isp(irac),
     *      isp(irl), isp(jrl), isp(iru), isp(jru),  flag)
        if(flag .ne. 0)  go to 100
c  ******  move ju next to jl  *****************************************
        jlmax = isp(ijl+n-1)
        ju    = jl + jlmax
        jumax = isp(iju+n-1)
        if (jumax.le.0)  go to 5
        do 4 j=1,jumax
   4      isp(ju+j-1) = isp(jutmp+j-1)
c
```

```
c  ******  call remaining subroutines  *********************************
   5  jlmax = isp(ijl+n-1)
      ju    = jl  + jlmax
      jumax = isp(iju+n-1)
      l     = (ju + jumax - 2 + lratio)  /  lratio    +    1
      lmax  = isp(il+n) - 1
      d     = l   + lmax
      u     = d   + n
      row   = nsp + 1 - n
      tmp   = row - n
      umax  = tmp - u
      esp   = umax - (isp(iu+n) - 1)
c
      if ((path-1) * (path-2) .ne. 0)  go to 6
        if (umax.lt.0)  go to 110
        call nnfc
     *     (n,  r, c, ic,  ia, ja, a, z, b,
     *      lmax, isp(il), isp(jl), isp(ijl), rsp(l),  rsp(d),
     *      umax, isp(iu), isp(ju), isp(iju), rsp(u),
     *      rsp(row), rsp(tmp),  isp(irl), isp(jrl),  flag)
        if(flag .ne. 0)  go to 100
c
   6  if ((path-3) .ne. 0)  go to 7
        call nnsc
     *     (n,  r, c,  isp(il), isp(jl), isp(ijl), rsp(l),
     *      rsp(d),    isp(iu), isp(ju), isp(iju), rsp(u),
     *      z, b,  rsp(tmp))
c
   7  if ((path-4) .ne. 0)  go to 8
        call nntc
     *     (n,  r, c,  isp(il), isp(jl), isp(ijl), rsp(l),
     *      rsp(d),    isp(iu), isp(ju), isp(iju), rsp(u),
     *      z, b,  rsp(tmp))
   8  return
c
c ** error.. error detected in nroc, nsfc, nnfc, or nnsc
 100  return
c ** error.. insufficient storage
 110  flag = 10*n + 1
      return
c ** error.. illegal path specification
 111  flag = 11*n + 1
      return
      end
      subroutine sro
     *     (n, ip, ia,ja,a, q, r, dflag)
c***********************************************************************
c  sro -- symmetric reordering of sparse symmetric matrix
c***********************************************************************
c
c  description
c
c    the nonzero entries of the matrix m are assumed to be stored
c    symmetrically in (ia,ja,a) format (i.e., not both m(i,j) and m(j,i)
c    are stored if i ne j).
c
c    sro does not rearrange the order of the rows, but does move
c    nonzeroes from one row to another to ensure that if m(i,j) will be
c    in the upper triangle of m with respect to the new ordering, then
c    m(i,j) is stored in row i (and thus m(j,i) is not stored),  whereas
c    if m(i,j) will be in the strict lower triangle of m, then m(j,i) is
c    stored in row j (and thus m(i,j) is not stored).
c
c
```

```
c  additional parameters
c
c    q     - integer one-dimensional work array.  dimension = n
c
c    r     - integer one-dimensional work array.  dimension = number of
c            nonzero entries in the upper triangle of m
c
c    dflag - logical variable.  if dflag = .true., then store nonzero
c            diagonal elements at the beginning of the row
c
c-----------------------------------------------------------------------
c
      integer  ip(*),  ia(*), ja(*),  q(*), r(*)
c...  real  a(*),  ak
      double precision  a(*),  ak
      logical  dflag
c
c
c--phase 1 -- find row in which to store each nonzero
c----initialize count of nonzeroes to be stored in each row
      do 1 i=1,n
  1     q(i) = 0
c
c----for each nonzero element a(j)
      do 3 i=1,n
        jmin = ia(i)
        jmax = ia(i+1) - 1
        if (jmin.gt.jmax)  go to 3
        do 2 j=jmin,jmax
c
c--------find row (=r(j)) and column (=ja(j)) in which to store a(j) ...
          k = ja(j)
          if (ip(k).lt.ip(i))  ja(j) = i
          if (ip(k).ge.ip(i))  k = i
          r(j) = k
c
c--------... and increment count of nonzeroes (=q(r(j)) in that row
  2       q(k) = q(k) + 1
  3     continue
c
c
c--phase 2 -- find new ia and permutation to apply to (ja,a)
c----determine pointers to delimit rows in permuted (ja,a)
      do 4 i=1,n
        ia(i+1) = ia(i) + q(i)
  4     q(i) = ia(i+1)
c
c----determine where each (ja(j),a(j)) is stored in permuted (ja,a)
c----for each nonzero element (in reverse order)
      ilast = 0
      jmin = ia(1)
      jmax = ia(n+1) - 1
      j = jmax
      do 6 jdummy=jmin,jmax
        i = r(j)
        if (.not.dflag .or. ja(j).ne.i .or. i.eq.ilast)  go to 5
c
c------if dflag, then put diagonal nonzero at beginning of row
          r(j) = ia(i)
          ilast = i
          go to 6
c
c------put (off-diagonal) nonzero in last unused location in row
  5       q(i) = q(i) - 1
```

```
          r(j) = q(i)
c
  6     j = j-1
c
c
c--phase 3 -- permute (ja,a) to upper triangular form (wrt new ordering)
      do 8 j=jmin,jmax
  7     if (r(j).eq.j)  go to 8
          k = r(j)
          r(j) = r(k)
          r(k) = k
          jak = ja(k)
          ja(k) = ja(j)
          ja(j) = jak
          ak = a(k)
          a(k) = a(j)
          a(j) = ak
          go to 7
  8     continue
c
      return
      end
      subroutine md
     *     (n, ia,ja, max, v,l, head,last,next, mark, flag)
c***********************************************************************
c  md -- minimum degree algorithm (based on element model)
c***********************************************************************
c
c  description
c
c    md finds a minimum degree ordering of the rows and columns of a
c    general sparse matrix m stored in (ia,ja,a) format.
c    when the structure of m is nonsymmetric, the ordering is that
c    obtained for the symmetric matrix  m + m-transpose.
c
c
c  additional parameters
c
c    max  - declared dimension of the one-dimensional arrays v and l.
c           max must be at least  n+2k,  where k is the number of
c           nonzeroes in the strict upper triangle of m + m-transpose
c
c    v    - integer one-dimensional work array.  dimension = max
c
c    l    - integer one-dimensional work array.  dimension = max
c
c    head - integer one-dimensional work array.  dimension = n
c
c    last - integer one-dimensional array used to return the permutation
c           of the rows and columns of m corresponding to the minimum
c           degree ordering.  dimension = n
c
c    next - integer one-dimensional array used to return the inverse of
c           the permutation returned in last.  dimension = n
c
c    mark - integer one-dimensional work array (may be the same as v).
c           dimension = n
c
c    flag - integer error flag.  values and their meanings are -
c             0     no errors detected
c             9n+k  insufficient storage in md
c
c
c  definitions of internal parameters
```

```
c
c    ---------+-----------------------------------------------------
c    v(s)     - value field of list entry
c    ---------+-----------------------------------------------------
c    l(s)     - link field of list entry  (0 =) end of list)
c    ---------+-----------------------------------------------------
c    l(vi)    - pointer to element list of uneliminated vertex vi
c    ---------+-----------------------------------------------------
c    l(ej)    - pointer to boundary list of active element ej
c    ---------+-----------------------------------------------------
c    head(d)  - vj =) vj head of d-list d
c             -  0 =) no vertex in d-list d
c
c
c             -                 vi uneliminated vertex
c             -          vi in ek           -       vi not in ek
c    ---------+-----------------------------+-------------------------
c    next(vi) - undefined but nonnegative   - vj =) vj next in d-list
c             -                             -  0 =) vi tail of d-list
c    ---------+-----------------------------+-------------------------
c    last(vi) - (not set until mdp)         - -d =) vi head of d-list d
c             --vk =) compute degree        - vj =) vj last in d-list
c             - ej =) vi prototype of ej    -  0 =) vi not in any d-list
c             -  0 =) do not compute degree -
c    ---------+-----------------------------+-------------------------
c    mark(vi) - mark(vk)                    - nonneg. tag .lt. mark(vk)
c
c
c             -                   vi eliminated vertex
c             -      ei active element      -           otherwise
c    ---------+-----------------------------+-------------------------
c    next(vi) - -j =) vi was j-th vertex    - -j =) vi was j-th vertex
c             -       to be eliminated      -       to be eliminated
c    ---------+-----------------------------+-------------------------
c    last(vi) -  m =) size of ei = m        - undefined
c    ---------+-----------------------------+-------------------------
c    mark(vi) - -m =) overlap count of ei   - undefined
c             -       with ek = m           -
c             - otherwise nonnegative tag   -
c             -       .lt. mark(vk)         -
c
c-----------------------------------------------------------------------
c
       integer  ia(*), ja(*),  v(*), l(*),  head(*), last(*), next(*),
      *   mark(*),  flag,  tag, dmin, vk,ek, tail
       equivalence  (vk,ek)
c
c----initialization
       tag = 0
       call  mdi
      *   (n, ia,ja, max,v,l, head,last,next, mark,tag, flag)
       if (flag.ne.0)  return
c
       k = 0
       dmin = 1
c
c----while  k .lt. n  do
   1  if (k.ge.n)  go to 4
c
c------search for vertex of minimum degree
   2    if (head(dmin).gt.0)  go to 3
          dmin = dmin + 1
          go to 2
c
```

```
c------remove vertex vk of minimum degree from degree list
   3    vk = head(dmin)
        head(dmin) = next(vk)
        if (head(dmin).gt.0)  last(head(dmin)) = -dmin
c
c------number vertex vk, adjust tag, and tag vk
        k = k+1
        next(vk) = -k
        last(ek) = dmin - 1
        tag = tag + last(ek)
        mark(vk) = tag
c
c------form element ek from uneliminated neighbors of vk
        call  mdm
     *     (vk,tail, v,l, last,next, mark)
c
c------purge inactive elements and do mass elimination
        call  mdp
     *     (k,ek,tail, v,l, head,last,next, mark)
c
c------update degrees of uneliminated vertices in ek
        call  mdu
     *     (ek,dmin, v,l, head,last,next, mark)
c
        go to 1
c
c----generate inverse permutation from permutation
   4  do 5 k=1,n
        next(k) = -next(k)
   5    last(next(k)) = k
c
      return
      end
      subroutine mdi
     *     (n, ia,ja, max,v,l, head,last,next, mark,tag, flag)
c***********************************************************************
c  mdi -- initialization
c***********************************************************************
      integer  ia(*), ja(*),  v(*), l(*),  head(*), last(*), next(*),
     *   mark(*), tag,  flag,  sfs, vi,dvi, vj
c
c----initialize degrees, element lists, and degree lists
      do 1 vi=1,n
        mark(vi) = 1
        l(vi) = 0
   1    head(vi) = 0
      sfs = n+1
c
c----create nonzero structure
c----for each nonzero entry a(vi,vj)
      do 6 vi=1,n
        jmin = ia(vi)
        jmax = ia(vi+1) - 1
        if (jmin.gt.jmax)  go to 6
        do 5 j=jmin,jmax
          vj = ja(j)
          if (vj-vi) 2, 5, 4
c
c------if a(vi,vj) is in strict lower triangle
c------check for previous occurrence of a(vj,vi)
   2      lvk = vi
          kmax = mark(vi) - 1
          if (kmax .eq. 0) go to 4
          do 3 k=1,kmax
```

```
            lvk = l(lvk)
            if (v(lvk).eq.vj) go to 5
   3        continue
c----for unentered entries a(vi,vj)
   4        if (sfs.ge.max)  go to 101
c
c------enter vj in element list for vi
            mark(vi) = mark(vi) + 1
            v(sfs) = vj
            l(sfs) = l(vi)
            l(vi) = sfs
            sfs = sfs+1
c
c------enter vi in element list for vj
            mark(vj) = mark(vj) + 1
            v(sfs) = vi
            l(sfs) = l(vj)
            l(vj) = sfs
            sfs = sfs+1
   5      continue
   6    continue
c
c----create degree lists and initialize mark vector
      do 7 vi=1,n
        dvi = mark(vi)
        next(vi) = head(dvi)
        head(dvi) = vi
        last(vi) = -dvi
        nextvi = next(vi)
        if (nextvi.gt.0)  last(nextvi) = vi
   7    mark(vi) = tag
c
      return
c
c ** error-  insufficient storage
 101  flag = 9*n + vi
      return
      end
      subroutine mdm
     *     (vk,tail, v,l, last,next, mark)
c***********************************************************************
c  mdm -- form element from uneliminated neighbors of vk
c***********************************************************************
      integer  vk, tail,  v(*), l(*),   last(*), next(*),   mark(*),
     *   tag, s,ls,vs,es, b,lb,vb, blp,blpmax
      equivalence  (vs, es)
c
c----initialize tag and list of uneliminated neighbors
      tag = mark(vk)
      tail = vk
c
c----for each vertex/element vs/es in element list of vk
      ls = l(vk)
   1  s = ls
      if (s.eq.0)  go to 5
        ls = l(s)
        vs = v(s)
        if (next(vs).lt.0)  go to 2
c
c------if vs is uneliminated vertex, then tag and append to list of
c------uneliminated neighbors
          mark(vs) = tag
          l(tail) = s
          tail = s
          go to 4
c
c------if es is active element, then ...
c--------for each vertex vb in boundary list of element es
   2      lb = l(es)
          blpmax = last(es)
          do 3 blp=1,blpmax
            b = lb
            lb = l(b)
            vb = v(b)
c
c----------if vb is untagged vertex, then tag and append to list of
c----------uneliminated neighbors
            if (mark(vb).ge.tag)  go to 3
              mark(vb) = tag
              l(tail) = b
              tail = b
   3        continue
c
c--------mark es inactive
          mark(es) = tag
c
   4    go to 1
c
c----terminate list of uneliminated neighbors
   5  l(tail) = 0
c
      return
      end
      subroutine mdp
     *     (k,ek,tail, v,l, head,last,next, mark)
c***********************************************************************
c  mdp -- purge inactive elements and do mass elimination
c***********************************************************************
      integer  ek, tail,  v(*), l(*),  head(*), last(*), next(*),
     *   mark(*),  tag, free, li,vi,lvi,evi, s,ls,es, ilp,ilpmax
c
c----initialize tag
      tag = mark(ek)
c
c----for each vertex vi in ek
      li = ek
      ilpmax = last(ek)
      if (ilpmax.le.0)  go to 12
      do 11 ilp=1,ilpmax
        i = li
        li = l(i)
        vi = v(li)
c
c------remove vi from degree list
        if (last(vi).eq.0)  go to 3
          if (last(vi).gt.0)  go to 1
            head(-last(vi)) = next(vi)
            go to 2
   1        next(last(vi)) = next(vi)
   2      if (next(vi).gt.0)  last(next(vi)) = last(vi)
c
c------remove inactive items from element list of vi
   3    ls = vi
   4    s = ls
        ls = l(s)
        if (ls.eq.0)  go to 6
          es = v(ls)
          if (mark(es).lt.tag)  go to 5
```

```
            free = ls
            l(s) = l(ls)
            ls = s
   5      go to 4
c
c------if vi is interior vertex, then remove from list and eliminate
   6    lvi = l(vi)
        if (lvi.ne.0)  go to 7
          l(i) = l(li)
          li = i
c
          k = k+1
          next(vi) = -k
          last(ek) = last(ek) - 1
          go to 11
c
c------else ...
c--------classify vertex vi
   7      if (l(lvi).ne.0)  go to 9
            evi = v(lvi)
            if (next(evi).ge.0)  go to 9
              if (mark(evi).lt.0)  go to 8
c
c----------if vi is prototype vertex, then mark as such, initialize
c----------overlap count for corresponding element, and move vi to end
c----------of boundary list
                last(vi) = evi
                mark(evi) = -1
                l(tail) = li
                tail = li
                l(i) = l(li)
                li = i
                go to 10
c
c----------else if vi is duplicate vertex, then mark as such and adjust
c----------overlap count for corresponding element
   8            last(vi) = 0
                mark(evi) = mark(evi) - 1
                go to 10
c
c----------else mark vi to compute degree
   9            last(vi) = -ek
c
c--------insert ek in element list of vi
  10      v(free) = ek
          l(free) = l(vi)
          l(vi) = free
  11    continue
c
c----terminate boundary list
  12  l(tail) = 0
c
      return
      end
      subroutine mdu
     *     (ek,dmin, v,l, head,last,next, mark)
c***********************************************************************
c  mdu -- update degrees of uneliminated vertices in ek
c***********************************************************************
      integer  ek, dmin,  v(*), l(*),  head(*), last(*), next(*),
     *   mark(*),  tag, vi,evi,dvi, s,vs,es, b,vb, ilp,ilpmax,
     *   blp,blpmax
      equivalence  (vs, es)
c
```

```
c----initialize tag
      tag = mark(ek) - last(ek)
c
c----for each vertex vi in ek
      i = ek
      ilpmax = last(ek)
      if (ilpmax.le.0)  go to 11
      do 10 ilp=1,ilpmax
        i = l(i)
        vi = v(i)
        if (last(vi))  1, 10, 8
c
c------if vi neither prototype nor duplicate vertex, then merge elements
c------to compute degree
   1      tag = tag + 1
          dvi = last(ek)
c
c--------for each vertex/element vs/es in element list of vi
          s = l(vi)
   2      s = l(s)
          if (s.eq.0)  go to 9
            vs = v(s)
            if (next(vs).lt.0)  go to 3
c
c----------if vs is uneliminated vertex, then tag and adjust degree
              mark(vs) = tag
              dvi = dvi + 1
              go to 5
c
c----------if es is active element, then expand
c------------check for outmatched vertex
   3          if (mark(es).lt.0)  go to 6
c
c------------for each vertex vb in es
              b = es
              blpmax = last(es)
              do 4 blp=1,blpmax
                b = l(b)
                vb = v(b)
c
c--------------if vb is untagged, then tag and adjust degree
                if (mark(vb).ge.tag)  go to 4
                  mark(vb) = tag
                  dvi = dvi + 1
   4            continue
c
   5        go to 2
c
c------else if vi is outmatched vertex, then adjust overlaps but do not
c------compute degree
   6      last(vi) = 0
          mark(es) = mark(es) - 1
   7      s = l(s)
          if (s.eq.0)  go to 10
            es = v(s)
            if (mark(es).lt.0)  mark(es) = mark(es) - 1
            go to 7
c
c------else if vi is prototype vertex, then calculate degree by
c------inclusion/exclusion and reset overlap count
   8      evi = last(vi)
          dvi = last(ek) + last(evi) + mark(evi)
          mark(evi) = 0
c
```

```
c------insert vi in appropriate degree list
   9    next(vi) = head(dvi)
        head(dvi) = vi
        last(vi) = -dvi
        if (next(vi).gt.0)  last(next(vi)) = vi
        if (dvi.lt.dmin)  dmin = dvi
c
  10    continue
c
  11  return
      end
*DECK DUMACH
      DOUBLE PRECISION FUNCTION DUMACH ()
C***BEGIN PROLOGUE  DUMACH
C***PURPOSE  Compute the unit roundoff of the machine.
C***CATEGORY  R1
C***TYPE      DOUBLE PRECISION (RUMACH-S, DUMACH-D)
C***KEYWORDS  MACHINE CONSTANTS
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C *Usage:
C        DOUBLE PRECISION  A, DUMACH
C        A = DUMACH()
C
C *Function Return Values:
C     A : the unit roundoff of the machine.
C
C *Description:
C     The unit roundoff is defined as the smallest positive machine
C     number u such that  1.0 + u .ne. 1.0.  This is computed by DUMACH
C     in a machine-independent manner.
C
C***REFERENCES  (NONE)
C***ROUTINES CALLED  DUMSUM
C***REVISION HISTORY  (YYYYMMDD)
C   19930216  DATE WRITTEN
C   19930818  Added SLATEC-format prologue.  (FNF)
C   20030707  Added DUMSUM to force normal storage of COMP.  (ACH)
C***END PROLOGUE  DUMACH
C
      DOUBLE PRECISION U, COMP
C***FIRST EXECUTABLE STATEMENT  DUMACH
      U = 1.0D0
 10   U = U*0.5D0
      CALL DUMSUM(1.0D0, U, COMP)
      IF (COMP .NE. 1.0D0) GO TO 10
      DUMACH = U*2.0D0
      RETURN
C----------------------- End of Function DUMACH ------------------------
      END
      SUBROUTINE DUMSUM(A,B,C)
C     Routine to force normal storing of A + B, for DUMACH.
      DOUBLE PRECISION A, B, C
      C = A + B
      RETURN
      END
*DECK XERRWD
      SUBROUTINE XERRWD (MSG, NMES, NERR, LEVEL, NI, I1, I2, NR, R1, R2)
C***BEGIN PROLOGUE  XERRWD
C***SUBSIDIARY
C***PURPOSE  Write error message with values.
C***CATEGORY  R3C
C***TYPE      DOUBLE PRECISION (XERRWV-S, XERRWD-D)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
```

```
C***DESCRIPTION
C
C  Subroutines XERRWD, XSETF, XSETUN, and the function routine IXSAV,
C  as given here, constitute a simplified version of the SLATEC error
C  handling package.
C
C  All arguments are input arguments.
C
C  MSG    = The message (character array).
C  NMES   = The length of MSG (number of characters).
C  NERR   = The error number (not used).
C  LEVEL  = The error level..
C           0 or 1 means recoverable (control returns to caller).
C           2 means fatal (run is aborted--see note below).
C  NI     = Number of integers (0, 1, or 2) to be printed with message.
C  I1,I2  = Integers to be printed, depending on NI.
C  NR     = Number of reals (0, 1, or 2) to be printed with message.
C  R1,R2  = Reals to be printed, depending on NR.
C
C  Note..  this routine is machine-dependent and specialized for use
C  in limited context, in the following ways..
C  1. The argument MSG is assumed to be of type CHARACTER, and
C     the message is printed with a format of (1X,A).
C  2. The message is assumed to take only one line.
C     Multi-line messages are generated by repeated calls.
C  3. If LEVEL = 2, control passes to the statement   STOP
C     to abort the run.  This statement may be machine-dependent.
C  4. R1 and R2 are assumed to be in double precision and are printed
C     in D21.13 format.
C
C***ROUTINES CALLED  IXSAV
C***REVISION HISTORY  (YYMMDD)
C   920831  DATE WRITTEN
C   921118  Replaced MFLGSV/LUNSAV by IXSAV. (ACH)
C   930329  Modified prologue to SLATEC format. (FNF)
C   930407  Changed MSG from CHARACTER*1 array to variable. (FNF)
C   930922  Minor cosmetic change. (FNF)
C***END PROLOGUE  XERRWD
C
C*Internal Notes:
C
C For a different default logical unit number, IXSAV (or a subsidiary
C routine that it calls) will need to be modified.
C For a different run-abort command, change the statement following
C statement 100 at the end.
C-----------------------------------------------------------------------
C Subroutines called by XERRWD.. None
C Function routine called by XERRWD.. IXSAV
C-----------------------------------------------------------------------
C**End
C
C  Declare arguments.
C
      DOUBLE PRECISION R1, R2
      INTEGER NMES, NERR, LEVEL, NI, I1, I2, NR
      CHARACTER*(*) MSG
C
C  Declare local variables.
C
      INTEGER LUNIT, IXSAV, MESFLG
C
C  Get logical unit number and message print flag.
C
C***FIRST EXECUTABLE STATEMENT  XERRWD
```

```
      LUNIT = IXSAV (1, 0, .FALSE.)
      MESFLG = IXSAV (2, 0, .FALSE.)
      IF (MESFLG .EQ. 0) GO TO 100
C
C  Write the message.
C
      WRITE (LUNIT,10)  MSG
 10   FORMAT(1X,A)
      IF (NI .EQ. 1) WRITE (LUNIT, 20) I1
 20   FORMAT(6X,'In above message,  I1 =',I10)
      IF (NI .EQ. 2) WRITE (LUNIT, 30) I1,I2
 30   FORMAT(6X,'In above message,  I1 =',I10,3X,'I2 =',I10)
      IF (NR .EQ. 1) WRITE (LUNIT, 40) R1
 40   FORMAT(6X,'In above message,  R1 =',D21.13)
      IF (NR .EQ. 2) WRITE (LUNIT, 50) R1,R2
 50   FORMAT(6X,'In above,  R1 =',D21.13,3X,'R2 =',D21.13)
C
C  Abort the run if LEVEL = 2.
C
 100  IF (LEVEL .NE. 2) RETURN
      STOP
C----------------------- End of Subroutine XERRWD ----------------------
      END
*DECK XSETUN
      SUBROUTINE XSETUN (LUN)
C***BEGIN PROLOGUE  XSETUN
C***PURPOSE  Reset the logical unit number for error messages.
C***CATEGORY  R3B
C***TYPE      ALL (XSETUN-A)
C***KEYWORDS  ERROR CONTROL
C***DESCRIPTION
C
C   XSETUN sets the logical unit number for error messages to LUN.
C
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***SEE ALSO  XERRWD, XERRWV
C***REFERENCES  (NONE)
C***ROUTINES CALLED  IXSAV
C***REVISION HISTORY  (YYMMDD)
C   921118  DATE WRITTEN
C   930329  Added SLATEC format prologue. (FNF)
C   930407  Corrected SEE ALSO section. (FNF)
C   930922  Made user-callable, and other cosmetic changes. (FNF)
C***END PROLOGUE  XSETUN
C
C Subroutines called by XSETUN.. None
C Function routine called by XSETUN.. IXSAV
C-----------------------------------------------------------------------
C**End
      INTEGER LUN, JUNK, IXSAV
C
C***FIRST EXECUTABLE STATEMENT  XSETUN
      IF (LUN .GT. 0) JUNK = IXSAV (1,LUN,.TRUE.)
      RETURN
C----------------------- End of Subroutine XSETUN ----------------------
      END
*DECK XSETF
      SUBROUTINE XSETF (MFLAG)
C***BEGIN PROLOGUE  XSETF
C***PURPOSE  Reset the error print control flag.
C***CATEGORY  R3A
C***TYPE      ALL (XSETF-A)
C***KEYWORDS  ERROR CONTROL
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
```

```
C***DESCRIPTION
C
C   XSETF sets the error print control flag to MFLAG:
C      MFLAG=1 means print all messages (the default).
C      MFLAG=0 means no printing.
C
C***SEE ALSO  XERRWD, XERRWV
C***REFERENCES  (NONE)
C***ROUTINES CALLED  IXSAV
C***REVISION HISTORY  (YYMMDD)
C   921118  DATE WRITTEN
C   930329  Added SLATEC format prologue. (FNF)
C   930407  Corrected SEE ALSO section. (FNF)
C   930922  Made user-callable, and other cosmetic changes. (FNF)
C***END PROLOGUE  XSETF
C
C Subroutines called by XSETF.. None
C Function routine called by XSETF.. IXSAV
C-----------------------------------------------------------------------
C**End
      INTEGER MFLAG, JUNK, IXSAV
C
C***FIRST EXECUTABLE STATEMENT  XSETF
      IF (MFLAG .EQ. 0 .OR. MFLAG .EQ. 1) JUNK = IXSAV (2,MFLAG,.TRUE.)
      RETURN
C----------------------- End of Subroutine XSETF -----------------------
      END
*DECK IXSAV
      INTEGER FUNCTION IXSAV (IPAR, IVALUE, ISET)
C***BEGIN PROLOGUE  IXSAV
C***SUBSIDIARY
C***PURPOSE  Save and recall error message control parameters.
C***CATEGORY  R3C
C***TYPE      ALL (IXSAV-A)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C  IXSAV saves and recalls one of two error message parameters:
C    LUNIT, the logical unit number to which messages are printed, and
C    MESFLG, the message print flag.
C  This is a modification of the SLATEC library routine J4SAVE.
C
C  Saved local variables..
C   LUNIT  = Logical unit number for messages.  The default is obtained
C            by a call to IUMACH (may be machine-dependent).
C   MESFLG = Print control flag..
C            1 means print all messages (the default).
C            0 means no printing.
C
C  On input..
C    IPAR   = Parameter indicator (1 for LUNIT, 2 for MESFLG).
C    IVALUE = The value to be set for the parameter, if ISET = .TRUE.
C    ISET   = Logical flag to indicate whether to read or write.
C             If ISET = .TRUE., the parameter will be given
C             the value IVALUE.  If ISET = .FALSE., the parameter
C             will be unchanged, and IVALUE is a dummy argument.
C
C  On return..
C    IXSAV = The (old) value of the parameter.
C
C***SEE ALSO  XERRWD, XERRWV
C***ROUTINES CALLED  IUMACH
C***REVISION HISTORY  (YYMMDD)
C   921118  DATE WRITTEN
```

```
C   930329  Modified prologue to SLATEC format. (FNF)
C   930915  Added IUMACH call to get default output unit.  (ACH)
C   930922  Minor cosmetic changes. (FNF)
C   010425  Type declaration for IUMACH added. (ACH)
C***END PROLOGUE  IXSAV
C
C Subroutines called by IXSAV.. None
C Function routine called by IXSAV.. IUMACH
C-----------------------------------------------------------------------
C**End
      LOGICAL ISET
      INTEGER IPAR, IVALUE
C-----------------------------------------------------------------------
      INTEGER IUMACH, LUNIT, MESFLG
C-----------------------------------------------------------------------
C The following Fortran-77 declaration is to cause the values of the
C listed (local) variables to be saved between calls to this routine.
C-----------------------------------------------------------------------
      SAVE LUNIT, MESFLG
      DATA LUNIT/-1/, MESFLG/1/
C
C***FIRST EXECUTABLE STATEMENT  IXSAV
      IF (IPAR .EQ. 1) THEN
        IF (LUNIT .EQ. -1) LUNIT = IUMACH()
        IXSAV = LUNIT
        IF (ISET) LUNIT = IVALUE
        ENDIF
C
      IF (IPAR .EQ. 2) THEN
        IXSAV = MESFLG
        IF (ISET) MESFLG = IVALUE
        ENDIF
C
      RETURN
C----------------------- End of Function IXSAV ------------------------
      END
*DECK IUMACH
      INTEGER FUNCTION IUMACH()
C***BEGIN PROLOGUE  IUMACH
C***PURPOSE  Provide standard output unit number.
C***CATEGORY  R1
C***TYPE      INTEGER (IUMACH-I)
C***KEYWORDS  MACHINE CONSTANTS
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C *Usage:
C        INTEGER  LOUT, IUMACH
C        LOUT = IUMACH()
C
C *Function Return Values:
C     LOUT : the standard logical unit for Fortran output.
C
C***REFERENCES  (NONE)
C***ROUTINES CALLED  (NONE)
C***REVISION HISTORY  (YYMMDD)
C   930915  DATE WRITTEN
C   930922  Made user-callable, and other cosmetic changes. (FNF)
C***END PROLOGUE  IUMACH
C
C*Internal Notes:
C  The built-in value of 6 is standard on a wide range of Fortran
C  systems.  This may be machine-dependent.
C**End
C***FIRST EXECUTABLE STATEMENT  IUMACH
```

```
      IUMACH = 6
C
      RETURN
C----------------------- End of Function IUMACH ------------------------
      END
```