NAG Library Routine Document

D02NEF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

D02NEF is a routine for integrating stiff systems of implicit ordinary differential equations coupled with algebraic equations.

2 Specification

```
 SUBROUTINE D02NEF(NEQ, T, TOUT, Y, YDOT, RTOL, ATOL, ITASK, RES, JAC,
1                  ICOM, COM, LCOM, IUSER, RUSER, IFAIL)
 INTEGER           NEQ, ITASK, ICOM(50+NEQ), LCOM, IUSER(*), IFAIL
 double precision  T, TOUT, Y(NEQ), YDOT(NEQ), RTOL(*), ATOL(*),
1                  COM(LCOM), RUSER(*)
 EXTERNAL          RES, JAC
```

3 Description

D02NEF is a general purpose routine for integrating the initial value problem for a stiff system of implicit ordinary differential equations with coupled algebraic equations written in the form

$$F\left(t, y, y'\right) = 0.$$

D02NEF uses the DASSL implementation of the Backward Differentiation Formulae (BDF) of orders one to five to solve a system of the above form for y (Y) and y' (YDOT). Values for Y and YDOT at the initial time must be given as input. These values must be consistent, (i.e., if T, Y, YDOT are the given initial values, they must satisfy $F(\mathrm{T}, \mathrm{Y}, \mathrm{YDOT}) = 0$). The routine solves the system from $t = \mathrm{T}$ to $t = \mathrm{TOUT}$.

An outline of a typical calling program for D02NEF is given below. It calls the DASSL implementation of the BDF integrator setup routine D02MWF and the banded matrix setup routine D02NPF (if required), and, if the integration needs to proceed, calls D02MCF before continuing the integration.

```
C     Declarations
C
      EXTERNAL RES, JAC
             .
             .
             .
C     Initialize the integrator
      CALL D02MWF(...)
C     Is the Jacobian matrix banded?
      IF (BANDED) CALL D02NPF(...)

C     Set DT to the required temporal resolution
C     Set TEND to the final time
C     Call the integrator for each temporal value:
1000  CALL D02NEF(...,RES,JAC,...)
C     Continue integration?
      IF (TOUT.LT.TEND .AND. ITASK.GE.0) THEN
        IF (ITASK.NE.1) TOUT = MIN(TOUT+DT,TEND)
C       Print solution
        CALL D02MCF(...)
        GO TO 1000
      ENDIF
             .
             .
             .
```

4 References

None.

5 Parameters

1: NEQ – INTEGER *Input*

On entry: the number of differential-algebraic equations to be solved.

Constraint: NEQ ≥ 1.

2: T – ***double precision*** *Input/Output*

On initial entry: the initial value of the independent variable, t.

On intermediate exit: t, the current value of the independent variable.

On final exit: the value of the independent variable at which the computed solution y is returned (usually at TOUT).

3: TOUT – ***double precision*** *Input*

On entry: the next value of t at which a computed solution is desired.

On initial entry: TOUT is used to determine the direction of integration. Integration is permitted in either direction (see also ITASK).

Constraint: TOUT $\neq$ T.

4: Y(NEQ) – ***double precision*** array *Input/Output*

On initial entry: the vector of initial values of the dependent variables y.

On intermediate exit: the computed solution vector, y, evaluated at $t = T$.

On final exit: the computed solution vector, evaluated at t (usually $t =$ TOUT).

5: YDOT(NEQ) – ***double precision*** array *Input/Output*

On initial entry: YDOT must contain approximations to the time derivatives y' of the vector y evaluated at the initial value of the independent variable.

On exit: the time derivatives y' of the vector y at the last integration point.

6: RTOL($*$) – ***double precision*** array *Input/Output*

Note: the dimension of the array RTOL which depends on the value of ITOL as set in D02MWF; it must be at least NEQ if ITOL = .TRUE. and at least 1 if ITOL = .FALSE. (see D02MWF).

On entry: the relative local error tolerance.

Constraint: RTOL(i) ≥ 0.0, for $i = 1, 2, \ldots, n$

where $n =$ NEQ when ITOL = .TRUE. and $n = 1$ otherwise.

On exit: RTOL remains unchanged unless D02NEF exits with IFAIL = 16 in which case the values may have been increased to values estimated to be appropriate for continuing the integration.

7: ATOL($*$) – ***double precision*** array *Input/Output*

Note: the dimension of the array ATOL which depends on the value of ITOL as set in D02MWF; it must be at least NEQ if ITOL = .TRUE. and at least 1 if ITOL = .FALSE. (see D02MWF).

On entry: the absolute local error tolerance.

Constraint: ATOL(i) ≥ 0.0, for $i = 1, 2, \ldots, n$

where $n =$ NEQ when ITOL = .TRUE. and $n = 1$ otherwise.

On exit: ATOL remains unchanged unless D02NEF exits with IFAIL = 16 in which case the values may have been increased to values estimated to be appropriate for continuing the integration.

8: ITASK – INTEGER *Input/Output*

On initial entry: need not be set.

On exit: the task performed by the integrator on successful completion or an indicator that a problem occurred during integration.

ITASK = 2
: The integration to TOUT was successfully completed (T = TOUT) by stepping exactly to TOUT.

ITASK = 3
: The integration to TOUT was successfully completed (T = TOUT) by stepping past TOUT. Y and YDOT are obtained by interpolation.

ITASK < 0
: Different negative values of ITASK returned correspond to different failure exits. IFAIL should always be checked in such cases and the corrective action taken where appropriate.

ITASK must remain **unchanged** between calls to D02NEF.

9: RES – SUBROUTINE, supplied by the user. *External Procedure*

RES must evaluate the residual

$$R = F(t, y, y').$$

The specification of RES is:

```
SUBROUTINE RES(NEQ, T, Y, YDOT, R, IRES, IUSER, RUSER)
INTEGER          NEQ, IRES, IUSER(*)
double precision T, Y(NEQ), YDOT(NEQ), R(NEQ), RUSER(*)
```

1: NEQ – INTEGER *Input*

On entry: the number of differential-algebraic equations being solved.

2: T – ***double precision*** *Input*

On entry: t, the current value of the independent variable.

3: Y(NEQ) – ***double precision*** array *Input*

On entry: y_i, the current solution component, for $i = 1, 2, \ldots,$ NEQ.

4: YDOT(NEQ) – ***double precision*** array *Input*

On entry: the derivative of the solution at the current point t.

5: R(NEQ) – ***double precision*** array *Output*

On exit: R(i) must contain the ith component of R, for $i = 1, 2, \ldots,$ NEQ where

$$R = F(\mathrm{T}, \mathrm{Y}, \mathrm{YDOT}).$$

6: IRES – INTEGER *Input/Output*

On entry: is always equal to zero.

On exit: IRES should normally be left unchanged. However, if an illegal value of Y is encountered, IRES should be set to -1; D02NEF will then attempt to resolve the problem so that illegal values of Y are not encountered. IRES should be set to -2 if you wish to return control to the calling (sub)routine; this will cause D02NEF to exit with IFAIL = 23.

7: IUSER(∗) – INTEGER array *User Workspace*
8: RUSER(∗) – ***double precision*** array *User Workspace*

You are free to use the arrays IUSER and RUSER to supply information to RES.

RES must be declared as EXTERNAL in the (sub)program from which D02NEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

10: JAC – SUBROUTINE, supplied by the NAG Library or the user. *External Procedure*

Evaluates the matrix of partial derivatives, J, where

$$J_{ij} = \frac{\partial F_i}{\partial y_j} + \text{CJ} \times \frac{\partial F_i}{\partial y'_j}, \qquad i,j = 1,2,\ldots,\text{NEQ}.$$

If this option is not required, the actual argument for JAC must be the dummy routine D02NEZ. D02NEZ is included in the NAG Library. You must indicate to the integrator whether this option is to be used by setting the parameter JCEVAL appropriately in a call to the setup routine D02MWF.

The specification of JAC is:

```
SUBROUTINE JAC(NEQ, T, Y, YDOT, PD, CJ, IUSER, RUSER)
INTEGER          NEQ, IUSER(*)
double precision T, Y(NEQ), YDOT(NEQ), PD(*), CJ, RUSER(*)
```

1: NEQ – INTEGER *Input*

On entry: the number of differential-algebraic equations being solved.

2: T – ***double precision*** *Input*

On entry: t, the current value of the independent variable.

3: Y(NEQ) – ***double precision*** array *Input*

On entry: y_i, the current solution component, for $i = 1,2,\ldots,\text{NEQ}$.

4: YDOT(NEQ) – ***double precision*** array *Input*

On entry: the derivative of the solution at the current point t.

5: PD(∗) – ***double precision*** array *Input/Output*

Note: the dimension of the array PD is NEQ × NEQ when the Jacobian is full and will be $(2 \times \text{ML} + \text{MU} + 1) \times \text{NEQ}$ when the Jacobian is banded (that is, a prior call to D02NPF has been made) (see D02NPF).

On entry: PD is preset to zero before the call to JAC.

On exit: if the Jacobian is full then $\text{PD}((i-j) \times \text{NEQ} + i) = J_{ij}$, for $i,j = 1,2,\ldots,\text{NEQ}$; if the Jacobian is banded then $\text{PD}((j-1) \times \text{NEQ} + \text{ML} + \text{MU} + i - j) = J_{ij}$, for $\max(1, j-\text{MU}) \le i \le \min(n, j+\text{ML})$; (see also in F07BDF (DGBTRF)).

6: CJ – ***double precision*** *Input*

On entry: CJ is a scalar constant which will be defined in D02NEF.

7: IUSER(∗) – INTEGER array *User Workspace*
8: RUSER(∗) – ***double precision*** array *User Workspace*

You are free to use the arrays IUSER and RUSER to supply information to JAC.

JAC must be declared as EXTERNAL in the (sub)program from which D02NEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

11: ICOM(50 + NEQ) – INTEGER array *Communication Array*

ICOM contains information which is usually of no interest, but is necessary for subsequent calls. However you may find the following useful:

ICOM(22)
The order of the method to be attempted on the next step.

ICOM(23)
The order of the method used on the last step.

ICOM(26)
The number of steps taken so far.

ICOM(27)
The number of calls to RES so far.

ICOM(28)
The number of evaluations of the matrix of partial derivatives needed so far.

ICOM(29)
The total number of error test failures so far.

ICOM(30)
The total number of convergence test failures so far.

12: COM(LCOM) – ***double precision*** array *Communication Array*

COM contains information which is usually of no interest, but is necessary for subsequent calls. However you may find the following useful:

COM(3)
The step size to be attempted on the next step.

COM(4)
The current value of the independent variable, i.e., the farthest point integration has reached. This will be different from T only when interpolation has been performed (ITASK = 3).

13: LCOM – INTEGER *Input*

On entry: the dimension of the array COM as declared in the (sub)program from which D02NEF is called.

Constraint: LCOM $\geq 40 + (maxorder + 4) \times$ NEQ + NEQ $\times p + q$ where $maxorder$ is the maximum order that can be used by the integration method (see MAXORD in D02MWF); $p =$ NEQ when the Jacobian is full and $p = (2 \times \text{ML} + \text{MU} + 1)$ when the Jacobian is banded; and, $q = (\text{NEQ}/(\text{ML} + \text{MU} + 1)) + 1$ when the Jacobian is to be evaluated numerically and $q = 0$ otherwise.

14: IUSER(∗) – INTEGER array *User Workspace*
15: RUSER(∗) – ***double precision*** array *User Workspace*

Note: the dimension of the arrays IUSER and RUSER must be at least 1.

You are free to use the arrays IUSER and RUSER to supply information to the calling subroutine.

16: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction for details.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

For environments where it might be inappropriate to halt program execution when an error is detected, the value −1 or 1 is recommended. If the output of error messages is undesirable, then

the value 1 is recommended. Otherwise, because for this routine the values of the output parameters may be useful even if IFAIL $\neq 0$ on exit, the recommended value is -1. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Note: D02NEF may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the routine:

IFAIL $= 1$

On entry, NEQ < 1

IFAIL $= 3$

On entry, TOUT $=$ T,
or TOUT is too close to T to start integration,
or TOUT is behind T in the direction of H0 (see D02MWF),

IFAIL $= 6$

On entry, RTOL$(i) < 0.0$ for $i = 1, 2, \ldots, n$, where $n =$ NEQ when ITOL $= 1$ and $n = 1$ otherwise,
or RTOL$(i) =$ ATOL$(i) = 0.0$ for all relevant i.

IFAIL $= 7$

On entry, ATOL$(i) < 0.0$ for $i = 1, 2, \ldots, n$, where $n =$ NEQ when ITOL $= 1$ and $n = 1$ otherwise.

IFAIL $= 8$

On entry, a previous call to this routine returned ITASK < 0 and IFAIL $\neq 0$, but no appropriate action was taken. For example, if a call returns with IFAIL $= 15$ (ITASK $= -1$) then a call to D02MCF must be made prior to making a continuation call to D02NEF.

IFAIL $= 12$

Either the initialization routine D02MWF has not been called prior to the first call of this routine or one of the communication arrays ICOM, COM has become corrupted.

IFAIL $= 13$

On entry, LCOM $< 40 + (maxorder + 4) \times$ NEQ $+$ NEQ $\times$ NEQ, and the Jacobian is full;
or LCOM $< 40 + (maxorder + 4) \times$ NEQ $+$ NEQ $\times (2 \times$ ML $+$ MU $+ 1) + q$, and the Jacobian is banded,

where $maxorder$ is the maximum order of the integration method to be used, and $q = (\text{NEQ}/(\text{ML} + \text{MU} + 1)) + 1$ when the Jacobian is to be evaluated numerically and $q = 0$ otherwise.

IFAIL $= 15$

The maximum number of steps (500) has been taken on this call and the integration has not reached TOUT. The integration can proceed by calling D02MCF prior to calling D02NEF again; this will reset the step counter to zero.

IFAIL = 16

Too much accuracy was requested for the precision of the machine. On output RTOL and ATOL were increased by an appropriate scale factor to prevent this error exit. Try running the problem again with these scaled tolerances.

IFAIL = 17

A purely relative tolerance was selected for a given solution component, but that solution component has become zero and a pure relative error test is impossible for this component. Perhaps an absolute tolerance requirement is also necessary for this component.

IFAIL = 18

The error test failed repeatedly using the minimum stepsize and so the integration could not proceed. If a (nonzero) minimum stepsize was specified in a call to D02MWF then either a reduction in this minimum stepsize or in the specified tolerances should be considered.

IFAIL = 19

The corrector step to obtain the approximate solution at the next time step repeatedly failed to converge using the minimum stepsize. This may be due to an inconsistency between the system of equations to be solved and initial conditions.

IFAIL = 20

The iteration matrix (Jacobian) has become singular. Please check the Jacobian evaluations when this is performed analytically. Also check for invalid solution values in the call to RES; these should be flagged by returning IRES $= -1$. This is probably due to an error in the analytic evaluation of the Jacobian.

IFAIL = 21

The corrector step could not converge and the error test failed repeatedly.

IFAIL = 22

IRES was set to -1 during a call to RES and the problem could not be resolved.

IFAIL = 23

IRES was set to -2 during a call to RES.

IFAIL = 24

The initial YDOT could not be computed. This could happen because the initial approximation to YDOT was very poor or because no YDOT exists that is consistent with the initial Y.

IFAIL = 25

Repeated occurrences of input constraint violations have been detected. This could result in a potential infinite loop.

7 Accuracy

The accuracy of the numerical solution may be controlled by a careful choice of the parameters RTOL and ATOL. You are advised to use scalar error control unless the components of the solution are expected to be poorly scaled. For the type of decaying solution typical of many stiff problems, relative error control with a small absolute error threshold will be most appropriate (that is, you are advised to choose ITOL = 0 with ATOL(1) small but positive).

8 Further Comments

The cost of computing a solution depends critically on the size of the differential system and to a lesser extent on the degree of stiffness of the problem. For banded systems the cost is proportional to NEQ $\times$ (ML + MU + 1)^2, while for full systems the cost is proportional to NEQ^3. Note however that for moderately sized problems which are only mildly nonlinear the cost may be dominated by factors proportional to NEQ $\times$ (ML + MU + 1) and NEQ^2 respectively.

9 Example

For this routine two examples are presented. There is a single example program for D02NEF, with a main program and the code to solve the two example problems given in Example 1 (EX1) and Example 2 (EX2).

Example 1 (EX1)

This example solves the well-known stiff Robertson problem written in implicit form

$$\begin{aligned} r_1 &= -0.04a + 1.0\text{E}4bc - a' \\ r_2 &= 0.04a - 1.0\text{E}4bc - 3.0\text{E}7b^2 - b' \\ r_3 &= 3.0\text{E}7b^2 - c' \end{aligned}$$

with initial conditions $a = 1.0$ and $b = c = 0.0$ over the range $[0, 0.1]$ the BDF method (setup routine D02MWF and D02NPF).

Example 2 (EX2)

This example illustrates the use of D02NEF to solve a simple algebraic problem by continuation. The equation $4 - 2y + 0.1e^y t = 0$ from $t = 0$ (where $y = 2$) to $t = 1$.

9.1 Program Text

```
*       D02NEF Example Program Text
*       Mark 22 Release. NAG Copyright 2008
*
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. External Subroutines ..
      EXTERNAL         EX1, EX2
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02NEF Example Program Results'
      CALL EX1
      CALL EX2
      END
*
      SUBROUTINE EX1
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, MAXORD, LICOM, MU, ML, LCOM
      PARAMETER        (NEQ=3,MAXORD=5,LICOM=50+NEQ,MU=2,ML=1,
     +                 LCOM=40+(MAXORD+4)*NEQ+(2*ML+MU+1)
     +                 *NEQ+2*(NEQ/(ML+MU+1)+1))
      INTEGER          IJAC
      PARAMETER        (IJAC=1)
*     .. Local Scalars ..
      DOUBLE PRECISION H0, HMAX, T, TOUT
      INTEGER          I, IFAIL, ITASK, ITOL, J
      CHARACTER*8      JCEVAL
*     .. Local Arrays ..
      DOUBLE PRECISION ATOL(NEQ), COM(LCOM), RTOL(NEQ), RUSER(1),
     +                 Y(NEQ), YDOT(NEQ)
      INTEGER          ICOM(LICOM), IUSER(3)
*     .. External Subroutines ..
      EXTERNAL         D02MCF, D02MWF, D02NEF, D02NPF, JAC1, RES1
*     .. Executable Statements ..
```

```
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'D02NEF Example 1'
      WRITE (NOUT,*)
      ITOL = 1
      DO I = 1, NEQ
         RTOL(I) = 1.0D-3
         ATOL(I) = 1.0D-6
         YDOT(I) = 0.0D0
      END DO
      IF (IJAC.EQ.1) THEN
         JCEVAL = 'Analytic'
      ELSE
         JCEVAL = 'Numeric'
      END IF
*     Set initial values
      Y(1) = 1.0D0
      Y(2) = 0.0D0
      Y(3) = 0.0D0
*
*     Initialize the problem, specifying that the Jacobian is to be
*     evaluated analytically using the provided routine JAC.
*
      HMAX = 0.0D0
      H0 = 0.0D0
      T = 0.0D0
      TOUT = 0.02D0
*
      IFAIL = 1
      CALL D02MWF(NEQ,MAXORD,JCEVAL,HMAX,H0,ITOL,ICOM,LICOM,COM,LCOM,
     +            IFAIL)
      IF (IFAIL.NE.0) THEN
         WRITE (NOUT,99996) IFAIL
         GO TO 40
      END IF
*
*     Specify that the Jacobian is banded.
*
      IFAIL = 0
      CALL D02NPF(NEQ,ML,MU,ICOM,LICOM,IFAIL)
*
*     Use the IUSER array to pass the band dimensions through to JAC.
*     An alternative would be to hard code values for ML and MU in JAC.
*
      IUSER(1) = ML
      IUSER(2) = MU
      IUSER(3) = IJAC
*
      WRITE (NOUT,*) '    t            y(1)        y(2)        y(3)   '
      WRITE (NOUT,99998) T, (Y(I),I=1,NEQ)
      ITASK = 0
*
*     Obtain the solution at 5 equally spaced values of T.
*
      DO 20 J = 1, 5
         IFAIL = 1
         CALL D02NEF(NEQ,T,TOUT,Y,YDOT,RTOL,ATOL,ITASK,RES1,JAC1,ICOM,
     +               COM,LCOM,IUSER,RUSER,IFAIL)
         WRITE (NOUT,99998) T, (Y(I),I=1,NEQ)
         IF (IFAIL.NE.0) THEN
            WRITE (NOUT,99997) IFAIL
            GO TO 40
         END IF
         TOUT = TOUT + 0.02D0
         CALL D02MCF(ICOM)
   20 CONTINUE
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) ITASK
*
   40 CONTINUE
*
```

```
99999 FORMAT (1X,'The integrator completed task, ITASK = ',I4)
99998 FORMAT (1X,F8.4,3X,3(F12.6))
99997 FORMAT (1X,' ** D02NEF returned with IFAIL = ',I5)
99996 FORMAT (1X,' ** D02MWF returned with IFAIL = ',I5)
      END
*
      SUBROUTINE RES1(NEQ,T,Y,YDOT,R,IRES,IUSER,RUSER)
*     .. Scalar Arguments ..
      DOUBLE PRECISION T
      INTEGER          IRES, NEQ
*     .. Array Arguments ..
      DOUBLE PRECISION R(NEQ), RUSER(1), Y(NEQ), YDOT(NEQ)
      INTEGER          IUSER(1)
*     .. Executable Statements ..
      R(1) = -0.04D0*Y(1) + 1.0D4*Y(2)*Y(3) - YDOT(1)
      R(2) = 0.04D0*Y(1) - 1.0D4*Y(2)*Y(3) - 3.0D7*Y(2)*Y(2) - YDOT(2)
      R(3) = 3.0D7*Y(2)*Y(2) - YDOT(3)
      RETURN
      END
*
      SUBROUTINE JAC1(NEQ,T,Y,YDOT,PD,CJ,IUSER,RUSER)
*     .. Scalar Arguments ..
      DOUBLE PRECISION CJ, T
      INTEGER          NEQ
*     .. Array Arguments ..
      DOUBLE PRECISION PD(*), RUSER(1), Y(NEQ), YDOT(NEQ)
      INTEGER          IUSER(3)
*     .. Local Scalars ..
      INTEGER          IJAC, ML, MU
*     .. External Subroutines ..
      EXTERNAL         D02NEZ, MYJAC1
*     .. Executable Statements ..
      ML = IUSER(1)
      MU = IUSER(2)
      IJAC = IUSER(3)
*
      IF (IJAC.EQ.1) THEN
         CALL MYJAC1(NEQ,ML,MU,T,Y,YDOT,PD,CJ)
      ELSE
         CALL D02NEZ(NEQ,T,Y,YDOT,PD,CJ,IUSER,RUSER)
      END IF
*
      RETURN
      END
*
      SUBROUTINE MYJAC1(NEQ,ML,MU,T,Y,YDOT,PD,CJ)
*     .. Scalar Arguments ..
      DOUBLE PRECISION CJ, T
      INTEGER          ML, MU, NEQ
*     .. Array Arguments ..
      DOUBLE PRECISION PD(2*ML+MU+1,NEQ), Y(NEQ), YDOT(NEQ)
*     .. Local Scalars ..
      INTEGER          MD, MS
*     .. Executable Statements ..
*     Main diagonal PDFULL(i,i), i=1,NEQ
      MD = MU + ML + 1
      PD(MD,1) = -0.04D0 - CJ
      PD(MD,2) = -1.0D4*Y(3) - 6.0D7*Y(2) - CJ
      PD(MD,3) = -CJ
*     1 Sub-diagonal PDFULL(i+1:i), i=1,NEQ-1
      MS = MD + 1
      PD(MS,1) = 0.04D0
      PD(MS,2) = 6.0D7*Y(2)
*     First super-diagonal PDFULL(i-1,i), i=2, NEQ
      MS = MD - 1
      PD(MS,2) = 1.0D4*Y(3)
      PD(MS,3) = -1.0D4*Y(2)
*     Second super-diagonal PDFULL(i-2,i), i=3, NEQ
      MS = MD - 2
      PD(MS,3) = 1.0D4*Y(2)
*
```

```
      RETURN
      END
*
      SUBROUTINE EX2
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, MAXORD, LICOM, LCOM
      PARAMETER        (NEQ=1,MAXORD=5,LICOM=50+NEQ,LCOM=40+(MAXORD+4)
     +                 *NEQ+NEQ*NEQ)
      INTEGER          IJAC
      PARAMETER        (IJAC=1)
*     .. Local Scalars ..
      DOUBLE PRECISION H0, HMAX, T, TOUT
      INTEGER          I, IFAIL, ITASK, ITOL, J
      CHARACTER*8      JCEVAL
*     .. Local Arrays ..
      DOUBLE PRECISION ATOL(NEQ), COM(LCOM), RTOL(NEQ), RUSER(1),
     +                 Y(NEQ), YDOT(NEQ)
      INTEGER          ICOM(LICOM), IUSER(1)
*     .. External Subroutines ..
      EXTERNAL         D02MCF, D02MWF, D02NEF, JAC2, RES2
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'D02NEF Example 2'
      WRITE (NOUT,*)
      ITOL = 1
      DO I = 1, NEQ
         RTOL(I) = 0.0D0
         ATOL(I) = 1.0D-8
         YDOT(I) = 0.0D0
      END DO
      IF (IJAC.EQ.1) THEN
         JCEVAL = 'Analytic'
      ELSE
         JCEVAL = 'Numeric'
      END IF
*
*     Initialize the problem, specifying that the Jacobian is to be
*     evaluated analytically using the provided routine JAC.
*
      HMAX = 0.0D0
      H0 = 0.0D0
      T = 0.0D0
      TOUT = 0.2D0
      Y(1) = 2.0D0
*
      IFAIL = 1
      CALL D02MWF(NEQ,MAXORD,JCEVAL,HMAX,H0,ITOL,ICOM,LICOM,COM,LCOM,
     +            IFAIL)
      IF (IFAIL.NE.0) THEN
         WRITE (NOUT,99996) IFAIL
         GO TO 40
      END IF
*
*     Use the IUSER array to pass whether numerical or analytic Jacobian
*     is to be used.
*
      IUSER(1) = IJAC
*
      WRITE (NOUT,*) '    t            y(1)'
      WRITE (NOUT,99998) T, (Y(I),I=1,NEQ)
      ITASK = 0
*
*     Obtain the solution at 5 equally spaced values of T.
*
      DO 20 J = 1, 5
         IFAIL = 1
         CALL D02NEF(NEQ,T,TOUT,Y,YDOT,RTOL,ATOL,ITASK,RES2,JAC2,ICOM,
     +               COM,LCOM,IUSER,RUSER,IFAIL)
         WRITE (NOUT,99998) T, (Y(I),I=1,NEQ)
```

```
         IF (IFAIL.NE.0) THEN
            WRITE (NOUT,99997) IFAIL
            GO TO 40
         END IF
         TOUT = TOUT + 0.2D0
         CALL D02MCF(ICOM)
   20 CONTINUE
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) ITASK
*
   40 CONTINUE
*
99999 FORMAT (1X,'The integrator completed task, ITASK = ',I4)
99998 FORMAT (1X,F8.4,3X,3(F12.6))
99997 FORMAT (1X,' ** D02NEF returned with IFAIL = ',I5)
99996 FORMAT (1X,' ** D02MWF returned with IFAIL = ',I5)
      END
*
      SUBROUTINE RES2(NEQ,T,Y,YDOT,R,IRES,IUSER,RUSER)
*     .. Scalar Arguments ..
      DOUBLE PRECISION T
      INTEGER          IRES, NEQ
*     .. Array Arguments ..
      DOUBLE PRECISION R(NEQ), RUSER(1), Y(NEQ), YDOT(NEQ)
      INTEGER          IUSER(1)
*     .. Intrinsic Functions ..
      INTRINSIC        EXP
*     .. Executable Statements ..
      R(1) = 4.0D0 - Y(1)**2 + T*0.1D0*EXP(Y(1))
      RETURN
      END
*
      SUBROUTINE JAC2(NEQ,T,Y,YDOT,PD,CJ,IUSER,RUSER)
*     .. Scalar Arguments ..
      DOUBLE PRECISION CJ, T
      INTEGER          NEQ
*     .. Array Arguments ..
      DOUBLE PRECISION PD(NEQ*NEQ), RUSER(1), Y(NEQ), YDOT(NEQ)
      INTEGER          IUSER(1)
*     .. Local Scalars ..
      INTEGER          IJAC
*     .. External Subroutines ..
      EXTERNAL         D02NEZ, MYJAC2
*     .. Executable Statements ..
      IJAC = IUSER(1)
*
      IF (IJAC.EQ.1) THEN
         CALL MYJAC2(NEQ,T,Y,YDOT,PD,CJ)
      ELSE
         CALL D02NEZ(NEQ,T,Y,YDOT,PD,CJ,IUSER,RUSER)
      END IF
*
      RETURN
      END
*
      SUBROUTINE MYJAC2(NEQ,T,Y,YDOT,PD,CJ)
*     .. Scalar Arguments ..
      DOUBLE PRECISION CJ, T
      INTEGER          NEQ
*     .. Array Arguments ..
      DOUBLE PRECISION PD(NEQ*NEQ), Y(NEQ), YDOT(NEQ)
*     .. Intrinsic Functions ..
      INTRINSIC        EXP
*     .. Executable Statements ..
      PD(1) = -2.0D0*Y(1) + 0.1D0*T*Y(1)*EXP(Y(1))
*
      RETURN
      END
```

9.2 Program Data

None.

9.3 Program Results

```
 D02NEF Example Program Results

 D02NEF Example 1

     t            y(1)        y(2)        y(3)
   0.0000       1.000000    0.000000    0.000000
   0.0200       0.999204    0.000036    0.000760
   0.0400       0.998415    0.000036    0.001549
   0.0600       0.997631    0.000036    0.002333
   0.0800       0.996852    0.000036    0.003112
   0.1000       0.996080    0.000036    0.003884

 The integrator completed task, ITASK =    3

 D02NEF Example 2

     t            y(1)
   0.0000       2.000000
   0.2000       2.038016
   0.4000       2.078379
   0.6000       2.121462
   0.8000       2.167736
   1.0000       2.217821

 The integrator completed task, ITASK =    3
```