# 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,     &
                   ICOM, COM, LCOM, IUSER, RUSER, IFAIL)

INTEGER            NEQ, ITASK, ICOM(50+NEQ), LCOM, IUSER(*), IFAIL
REAL (KIND=nag_wp) T, TOUT, Y(NEQ), YDOT(NEQ), RTOL(*), ATOL(*),          &
                   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

$$F(t, y, y') = 0.$$

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.

```
!     Declarations

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

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

## 4 References

None.

## 5 Arguments

1: NEQ – INTEGER *Input*

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

*Constraint*: NEQ $\geq 1$.

2: T – REAL (KIND=nag_wp) *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 – REAL (KIND=nag_wp) *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) – REAL (KIND=nag_wp) 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) – REAL (KIND=nag_wp) 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($*$) – REAL (KIND=nag_wp) array *Input/Output*

**Note**: the dimension of the array RTOL 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..

*On entry*: the relative local error tolerance.

*Constraint*: RTOL($i$) $\geq 0.0$, 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($*$) – REAL (KIND=nag_wp) array *Input/Output*

**Note**: the dimension of the array ATOL 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..

*On entry*: the absolute local error tolerance.

*Constraint*: ATOL($i$) $\geq 0.0$, 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(*)
REAL (KIND=nag_wp) T, Y(NEQ), YDOT(NEQ), R(NEQ), RUSER(*)
```

1: NEQ – INTEGER *Input*

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

2: T – REAL (KIND=nag_wp) *Input*

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

3: Y(NEQ) – REAL (KIND=nag_wp) array *Input*

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

4: YDOT(NEQ) – REAL (KIND=nag_wp) array *Input*

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

5: R(NEQ) – REAL (KIND=nag_wp) array *Output*

*On exit*: R$(i)$ must contain the $i$th 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(∗) – REAL (KIND=nag_wp) array *User Workspace*

RES is called with the arguments IUSER and RUSER as supplied to D02NEF. You should use the arrays IUSER and RUSER to supply information to RES.

RES must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which D02NEF is called. Arguments 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

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

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 argument 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(*)
REAL (KIND=nag_wp) T, Y(NEQ), YDOT(NEQ), PD(*), CJ, RUSER(*)
```

1: NEQ – INTEGER *Input*

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

2: T – REAL (KIND=nag_wp) *Input*

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

3: Y(NEQ) – REAL (KIND=nag_wp) array *Input*

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

4: YDOT(NEQ) – REAL (KIND=nag_wp) array *Input*

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

5: PD(∗) – REAL (KIND=nag_wp) array *Input/Output*

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

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

6: CJ – REAL (KIND=nag_wp) *Input*

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

7: IUSER(∗) – INTEGER array *User Workspace*
8: RUSER(∗) – REAL (KIND=nag_wp) array *User Workspace*

JAC is called with the arguments IUSER and RUSER as supplied to D02NEF. You should use the arrays IUSER and RUSER to supply information to JAC.

JAC must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which D02NEF is called. Arguments 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) – REAL (KIND=nag_wp) 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 $\geq 40 + (maxorder + 4) \times \text{NEQ} + \text{NEQ} \times p + q$ where $maxorder$ is the maximum order that can be used by the integration method (see MAXORD in D02MWF); $p = \text{NEQ}$ 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(∗) – REAL (KIND=nag_wp) array *User Workspace*

IUSER and RUSER are not used by D02NEF, but are passed directly to RES and JAC and should be used to pass information to these routines.

16: IFAIL – INTEGER *Input/Output*

*On entry*: IFAIL must be set to 0, $-1$ or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

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 arguments 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.**

*On exit*: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

## 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 $= \langle value \rangle$.
Constraint: NEQ $\geq 1$.

IFAIL $= 3$

On entry, T $= \langle value \rangle$.
Constraint: TOUT $\neq$ T.

TOUT is behind T in the direction of $h$: TOUT $-$ T $= \langle value \rangle$, $h = \langle value \rangle$.

TOUT is too close to T to start integration: TOUT $-$ T $= \langle value \rangle$: $hmin = \langle value \rangle$.

IFAIL $= 6$

Some element of RTOL is less than zero.

IFAIL $= 7$

Some element of ATOL is less than zero.

IFAIL $= 8$

A previous call to this routine returned with ITASK $= \langle value \rangle$ and no appropriate action was taken.

IFAIL $= 11$

Either the initialization routine has not been called prior to the first call of this routine or a communication array has become corrupted.

IFAIL $= 12$

Either the initialization routine has not been called prior to the first call of this routine or a communication array has become corrupted.

IFAIL $= 13$

COM array is of insufficient length; length required $= \langle value \rangle$; actual length LCOM $= \langle value \rangle$.

IFAIL $= 14$

All elements of RTOL and ATOL are zero.

IFAIL $= 15$

Maximum number of steps taken on this call before reaching TOUT: T $= \langle value \rangle$, maximum number of steps $= \langle value \rangle$.

IFAIL $= 16$

Too much accuracy requested for precision of machine. RTOL and ATOL were increased by scale factor $R$. Try running again with these scaled tolerances. T $= \langle value \rangle$, $R = \langle value \rangle$.

IFAIL $= 17$

A solution component has become zero when a purely relative tolerance (zero absolute tolerance) was selected for that component. T $= \langle value \rangle$, Y$(I) = \langle value \rangle$ for component $I = \langle value \rangle$.

IFAIL $= 18$

The error test failed repeatedly with $|h| = hmin$. T $= \langle value \rangle$. Stepsize $h = \langle value \rangle$.

IFAIL $= 19$

The corrector repeatedly failed to converge with $|h| = hmin$. T $= \langle value \rangle$. Stepsize $h = \langle value \rangle$.

IFAIL $= 20$

The iteration matrix is singular. T $= \langle value \rangle$. Stepsize $h = \langle value \rangle$.

IFAIL $= 21$

The corrector could not converge and the error test failed repeatedly. T $= \langle value \rangle$. Stepsize $h = \langle value \rangle$.

IFAIL $= 22$

IRES was set to $-1$ during a call to RES and could not be resolved. T $= \langle value \rangle$. Stepsize $h = \langle value \rangle$.

IFAIL $= 23$

IRES was set to $-2$ during a call to RES. T $= \langle value \rangle$. Stepsize $= \langle value \rangle$.

IFAIL $= 24$

The initial YDOT could not be computed. T $= \langle value \rangle$. Stepsize $h = \langle value \rangle$.

IFAIL $= 25$

Repeated occurrences of input constraint violations have been detected. This could result in a potential infinite loop: ITASK $= \langle value \rangle$. Current violation corresponds to exit with IFAIL $= \langle value \rangle$.

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

## 7 Accuracy

The accuracy of the numerical solution may be controlled by a careful choice of the arguments 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 Parallelism and Performance

D02NEF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

D02NEF makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this routine. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

## 9 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.

## 10 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

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

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 $t = 1$.

## 10.1 Program Text

```
!   D02NEF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module d02nefe_mod

!     D02NEF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                             :: jac1, jac2, res1, res2
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter      :: alpha = 0.04_nag_wp
      Real (Kind=nag_wp), Parameter      :: beta = 1.0E4_nag_wp
      Real (Kind=nag_wp), Parameter      :: gamma = 3.0E7_nag_wp
      Real (Kind=nag_wp), Parameter      :: two = 2.0_nag_wp
      Integer, Parameter, Public         :: ml = 1, mu = 2, neq1 = 3, neq2 = 1,  &
                                            nin = 5, nout = 6
    Contains
      Subroutine myjac1(neq,ml,mu,t,y,ydot,pd,cj)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: cj, t
        Integer, Intent (In)            :: ml, mu, neq
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: pd(2*ml+mu+1,neq)
        Real (Kind=nag_wp), Intent (In) :: 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) = -alpha - cj
        pd(md,2) = -beta*y(3) - two*gamma*y(2) - cj
        pd(md,3) = -cj
!       1 Subdiagonal pdfull(i+1:i), i=1,neq-1
        ms = md + 1
        pd(ms,1) = alpha
        pd(ms,2) = two*gamma*y(2)
!       First superdiagonal pdfull(i-1,i), i=2, neq
        ms = md - 1
        pd(ms,2) = beta*y(3)
        pd(ms,3) = -beta*y(2)
!       Second superdiagonal pdfull(i-2,i), i=3, neq
        ms = md - 2
        pd(ms,3) = beta*y(2)

        Return
      End Subroutine myjac1
      Subroutine myjac2(neq,t,y,ydot,pd,cj)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: cj, t
        Integer, Intent (In)            :: neq
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: pd(neq*neq)
        Real (Kind=nag_wp), Intent (In) :: y(neq), ydot(neq)
!       .. Intrinsic Procedures ..
        Intrinsic                       :: exp
!       .. Executable Statements ..
        pd(1) = -two*y(1) + 0.1E0_nag_wp*t*y(1)*exp(y(1))

        Return
      End Subroutine myjac2
```

```
      Subroutine res1(neq,t,y,ydot,r,ires,iuser,ruser)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: t
        Integer, Intent (Inout)        :: ires
        Integer, Intent (In)           :: neq
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: r(neq)
        Real (Kind=nag_wp), Intent (Inout) :: ruser(*)
        Real (Kind=nag_wp), Intent (In) :: y(neq), ydot(neq)
        Integer, Intent (Inout)        :: iuser(*)
!       .. Executable Statements ..
        r(1) = -alpha*y(1) + beta*y(2)*y(3) - ydot(1)
        r(2) = alpha*y(1) - beta*y(2)*y(3) - gamma*y(2)*y(2) - ydot(2)
        r(3) = gamma*y(2)*y(2) - ydot(3)
        Return
      End Subroutine res1
      Subroutine jac1(neq,t,y,ydot,pd,cj,iuser,ruser)

!       .. Use Statements ..
        Use nag_library, Only: d02nez
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: cj, t
        Integer, Intent (In)           :: neq
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout) :: pd(*), ruser(*)
        Real (Kind=nag_wp), Intent (In) :: y(neq), ydot(neq)
        Integer, Intent (Inout)        :: iuser(*)
!       .. Local Scalars ..
        Integer                        :: ijac, ml, mu
!       .. Executable Statements ..
        ml = iuser(1)
        mu = iuser(2)
        ijac = iuser(3)

        If (ijac==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 jac1
      Subroutine res2(neq,t,y,ydot,r,ires,iuser,ruser)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: t
        Integer, Intent (Inout)        :: ires
        Integer, Intent (In)           :: neq
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: r(neq)
        Real (Kind=nag_wp), Intent (Inout) :: ruser(*)
        Real (Kind=nag_wp), Intent (In) :: y(neq), ydot(neq)
        Integer, Intent (Inout)        :: iuser(*)
!       .. Intrinsic Procedures ..
        Intrinsic                      :: exp
!       .. Executable Statements ..
        r(1) = 4.0_nag_wp - y(1)**2 + t*0.1E0_nag_wp*exp(y(1))
        Return
      End Subroutine res2
      Subroutine jac2(neq,t,y,ydot,pd,cj,iuser,ruser)

!       .. Use Statements ..
        Use nag_library, Only: d02nez
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: cj, t
        Integer, Intent (In)           :: neq
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout) :: pd(*), ruser(*)
        Real (Kind=nag_wp), Intent (In) :: y(neq), ydot(neq)
        Integer, Intent (Inout)        :: iuser(*)
```

```
!       .. Local Scalars ..
        Integer                          :: ijac
!       .. Executable Statements ..
        ijac = iuser(1)

        If (ijac==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 jac2
    End Module d02nefe_mod
    Program d02nefe

!     D02NEF Example Main Program

!     .. Use Statements ..
      Use d02nefe_mod, Only: nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Executable Statements ..
      Write (nout,*) 'D02NEF Example Program Results'

      Call ex1

      Call ex2

    Contains
      Subroutine ex1

!       .. Use Statements ..
        Use nag_library, Only: d02mcf, d02mwf, d02nef, d02npf, nag_wp
        Use d02nefe_mod, Only: jac1, ml, mu, neq1, nin, res1
!       .. Local Scalars ..
        Real (Kind=nag_wp)               :: h0, hmax, t, tout
        Integer                          :: i, ifail, ijac, itask, itol, j,      &
                                            lcom, licom, maxord, neq
        Character (8)                    :: jceval
!       .. Local Arrays ..
        Real (Kind=nag_wp), Allocatable :: atol(:), com(:), rtol(:), y(:),     &
                                           ydot(:)
        Real (Kind=nag_wp)               :: ruser(1)
        Integer, Allocatable             :: icom(:)
        Integer                          :: iuser(3)
!       .. Executable Statements ..
        Write (nout,*)
        Write (nout,*) 'D02NEF Example 1'
!       Skip heading in data file
        Read (nin,*)
        Read (nin,*) maxord
        neq = neq1
        lcom = 40 + (maxord+4)*neq + (2*ml+mu+1)*neq + 2*(neq/(ml+mu+1)+1)
        licom = 50 + neq
        Allocate (atol(neq),com(lcom),rtol(neq),y(neq),ydot(neq),icom(licom))
        Read (nin,*) ijac, itol
        Read (nin,*) rtol(1:neq)
        Read (nin,*) atol(1:neq)
        Read (nin,*) ydot(1:neq)
        If (ijac==1) Then
          jceval = 'Analytic'
        Else
          jceval = 'Numeric'
        End If
!       Set initial values
        Read (nin,*) y(1:neq)

!       Initialize the problem, specifying that the Jacobian is to be
!       evaluated analytically using the provided routine jac.
```

```
        Read (nin,*) hmax, h0
        Read (nin,*) t, tout

!       ifail: behaviour on error exit
!              =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
        ifail = 0
        Call d02mwf(neq,maxord,jceval,hmax,h0,itol,icom,licom,com,lcom,ifail)

!       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,99999)(i,i=1,neq)
        Write (nout,99998) t, (y(i),i=1,neq)
        itask = 0

!       Obtain the solution at 5 equally spaced values of T.

loop:   Do 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/=0) Then
            Write (nout,99997) ifail
            Exit loop
          End If
          tout = tout + 0.02_nag_wp
          Call d02mcf(icom)
        End Do loop

        Write (nout,*)
        Write (nout,99996) itask

99999   Format (/,1X,'    t ',5X,3('      Y(',I1,')  '))
99998   Format (1X,F8.4,3X,3(F12.6))
99997   Format (1X,' ** D02NEF returned with IFAIL = ',I5)
99996   Format (1X,'The integrator completed task, ITASK = ',I4)
      End Subroutine ex1
      Subroutine ex2

!       .. Use Statements ..
        Use nag_library, Only: d02mcf, d02mwf, d02nef, nag_wp
        Use d02nefe_mod, Only: jac2, neq2, nin, res2
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: h0, hmax, t, tout
        Integer                        :: i, ifail, ijac, itask, itol, j,     &
                                          lcom, licom, maxord, neq
        Character (8)                  :: jceval
!       .. Local Arrays ..
        Real (Kind=nag_wp), Allocatable :: atol(:), com(:), rtol(:), y(:),    &
                                           ydot(:)
        Real (Kind=nag_wp)             :: ruser(1)
        Integer, Allocatable           :: icom(:)
        Integer                        :: iuser(1)
!       .. Executable Statements ..
        Write (nout,*)
        Write (nout,*) 'D02NEF Example 2'
        Write (nout,*)
        Read (nin,*)
        Read (nin,*) maxord
        neq = neq2
        lcom = 40 + (maxord+4)*neq + neq*neq
```

```
        licom = 50 + neq
        Allocate (atol(neq),com(lcom),rtol(neq),y(neq),ydot(neq),icom(licom))
        Read (nin,*) ijac, itol
        Read (nin,*) rtol(1:neq)
        Read (nin,*) atol(1:neq)
        Read (nin,*) ydot(1:neq)
        If (ijac==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.

        Read (nin,*) y(1:neq)
        Read (nin,*) hmax, h0
        Read (nin,*) t, tout

        ifail = 0
        Call d02mwf(neq,maxord,jceval,hmax,h0,itol,icom,licom,com,lcom,ifail)

!       Use the iuser array to pass whether numerical or analytic Jacobian
!       is to be used.

        iuser(1) = ijac

        Write (nout,99999)(i,i=1,neq)
        Write (nout,99998) t, y(1:neq)
        itask = 0

!       Obtain the solution at 5 equally spaced values of t.

loop:   Do 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(1:neq)
          If (ifail/=0) Then
            Write (nout,99997) ifail
            Exit loop
          End If
          tout = tout + 0.2_nag_wp
          Call d02mcf(icom)
        End Do loop

        Write (nout,*)
        Write (nout,99996) itask

99999   Format (/,1X,'    t            y(',I1,')')
99998   Format (1X,F8.4,3X,3(F12.6))
99997   Format (1X,' ** D02NEF returned with IFAIL = ',I5)
99996   Format (1X,'The integrator completed task, ITASK = ',I4)
      End Subroutine ex2
    End Program d02nefe
```

## 10.2 Program Data

```
D02NEF Example Program Data
   5                        : ex1   : maxord
   1  1                             : ijac, itol
   1.0E-3 1.0E-3 1.0E-3            : rtol
   1.0E-6 1.0E-6 1.0E-6            : atol
   0.0  0.0  0.0                   : ydot
   1.0  0.0  0.0                   : y
   0.0  0.0                        : hmax, h0
   0.0  0.02                       : t, tout
```

```
  5                         : ex2   : maxord
  1  1                              : ijac, itol
  0.0                            : rtol
  1.0E-8                         : atol
  0.0                            : ydot
  2.0                            : y
  0.0  0.0                       : hmax, h0
  0.0  0.2                       : t, tout
```

## 10.3 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
```