NAG Library Routine Document

D02NNF

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

D02NNF is a reverse communication routine for integrating stiff systems of implicit ordinary differential equations coupled with algebraic equations.

2 Specification

```
SUBROUTINE D02NNF (NEQ, LDYSAV, T, TOUT, Y, YDOT, RWORK, RTOL, ATOL,        &
                   ITOL, INFORM, YSAV, SDYSAV, WKJAC, NWKJAC, JACPVT,       &
                   NJCPVT, IMON, INLN, IRES, IREVCM, LDERIV, ITASK,         &
                   ITRACE, IFAIL)
INTEGER            NEQ, LDYSAV, ITOL, INFORM(23), SDYSAV, NWKJAC,           &
                   JACPVT(NJCPVT), NJCPVT, IMON, INLN, IRES, IREVCM,        &
                   ITASK, ITRACE, IFAIL
REAL (KIND=nag_wp) T, TOUT, Y(NEQ), YDOT(NEQ), RWORK(50+4*NEQ),             &
                   RTOL(*), ATOL(*), YSAV(LDYSAV,SDYSAV),                   &
                   WKJAC(NWKJAC)
LOGICAL            LDERIV(2)
```

3 Description

D02NNF is a general purpose routine for integrating the initial value problem for a stiff system of implicit ordinary differential equations coupled with algebraic equations, written in the form

$$A(t,y)y' = g(t,y).$$

An outline of a typical calling program is given below:

```
     !     Declarations

           call linear algebra setup routine
           call integrator setup routine
           IREVCM=0
     1000  CALL D02NNF(NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
           ATOL, ITOL, INFORM, YSAVE, NY2DIM, WKJAC, NWKJAC, JACPVT,
           NJCPVT, IMON, INLN, IRES, IREVCM, LDERIV,
           ITASK, ITRACE, IFAIL)

           IF (IREVCM.GT.0) THEN
             IF (IREVCM.GT.7 .AND. IREVCM.LT.11) THEN
               IF (IREVCM.EQ.8) THEN
                 supply the Jacobian matrix                          (i)
               ELSE IF (IREVCM.EQ.9) THEN
                 perform monitoring tasks requested by the user      (ii)
               ELSE IF (IREVCM.EQ.10) THEN
                 indicates an unsuccessful step
               END IF
             ELSE
               evaluate the residual                                 (iii)
             ENDIF
             GO TO 1000
           END IF

      !    post processing (optional linear algebra diagnostic call
      !    (sparse case only), optional integrator diagnostic call)
```

```
      STOP
      END
```

There are three major operations that may be required of the calling subroutine on an intermediate return (IREVCM $\neq 0$) from D02NNF; these are denoted (i), (ii) and (iii).

The following sections describe in greater detail exactly what is required of each of these operations.

(i) **Supply the Jacobian matrix**

You need only provide this facility if the argument JCEVAL = 'A' (or JCEVAL = 'F' if using sparse matrix linear algebra) in a call to the linear algebra setup routine (see JCEVAL in D02NUF). If the Jacobian matrix is to be evaluated numerically by the integrator, then the remainder of section (i) can be ignored.

We must define the system of nonlinear equations which is solved internally by the integrator. The time derivative, y', has the form

$$y' = (y - z)/(hd),$$

where h is the current step size and d is an argument that depends on the integration method in use. The vector y is the current solution and the vector z depends on information from previous time steps. This means that $\frac{d}{dy'}(\ ) = (hd)\frac{d}{dy}(\ )$.

The system of nonlinear equations that is solved has the form

$$A(t, y)y' - g(t, y) = 0$$

but is solved in the form

$$f(t, y) = 0,$$

where f is the function defined by

$$f(t, y) = (hd)(A(t, y)(y - z)/(hd) - g(t, y)).$$

It is the Jacobian matrix $\dfrac{\partial r}{\partial y}$ that you must supply as follows:

$$\frac{\partial f_i}{\partial y_j} = a_{ij}(t, y) + hd\frac{\partial}{\partial y_j}\left(\sum_{k=1}^{\mathrm{NEQ}} a_{ik}(t, y){y'}_k - g_i(t, y)\right),$$

where t, h and d are located in RWORK(19), RWORK(16) and RWORK(20) respectively and the arrays Y and YDOT contain the current solution and time derivatives respectively. Only the nonzero elements of the Jacobian need be set, since the locations where it is to be stored are preset to zero.

Hereafter in this document this operation will be referred to as JAC.

(ii) **Perform tasks requested by you**

This operation is essentially a monitoring function and additionally provides the opportunity of changing the current values of Y, YDOT, HNEXT (the step size that the integrator proposes to take on the next step), HMIN (the minimum step size to be taken on the next step), and HMAX (the maximum step size to be taken on the next step). The scaled local error at the end of a time step may be obtained by calling the real function D02ZAF as follows:

```
      IFAIL = 1
       ERRLOC = D02ZAF(NEQ,ROWK(51+NEQMAX),RWORK(51),IFAIL)
!      CHECK IFAIL BEFORE PROCEEDING
```

The following gives details of the location within the array RWORK of variables that may be of interest to you:

Variable	Specification	Location
TCURR	the current value of the independent variable	RWORK(19)
HLAST	last step size successfully used by the integrator	RWORK(15)
HNEXT	step size that the integrator proposes to take on the next step	RWORK(16)

HMIN	minimum step size to be taken on the next step	RWORK(17)
HMAX	maximum step size to be taken on the next step	RWORK(18)
NQU	the order of the integrator used on the last step	RWORK(10)

You are advised to consult the description of MONITR in D02NGF for details on what optional input can be made.

If either Y or YDOT are changed, then IMON must be set to 2 before return to D02NNF. If either of the values HMIN or HMAX are changed, then IMON must be set ≥ 3 before return to D02NNF. If HNEXT is changed, then IMON must be set to 4 before return to D02NNF.

In addition you can force D02NNF to evaluate the residual vector

$$A(t,y)y' - g(t,y)$$

by setting IMON $= 0$ and INLN $= 3$ and then returning to D02NNF; on return to this monitoring operation the residual vector will be stored in RWORK$(50 + 2 \times \text{NEQ} + i)$, for $i = 1, 2, \ldots, \text{NEQ}$.

Hereafter in this document this operation will be referred to as MONITR.

(iii) **Evaluate the residual**

This operation must evaluate the residual

$$-r = g(t,y) - A(t,y)y' \tag{1}$$

in one case and the reduced residual

$$-\hat{r} = -A(t,y)y' \tag{2}$$

in another, where t is located in RWORK(19). The form of the residual that is returned is determined by the value of IRES returned by D02NNF. If IRES $= -1$, then the residual defined by equation (2) above must be returned; if IRES $= 1$, then the residual returned by equation (1) above must be returned.

Hereafter in this document this operation will be referred to as RESID.

4 References

See the D02M–N Sub-chapter Introduction.

5 Arguments

Note: this routine uses **reverse communication.** Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the argument **IREVCM**. Between intermediate exits and re-entries, **all arguments other than YDOT, RWORK, WKJAC, IMON, INLN and IRES must remain unchanged**.

1: NEQ – INTEGER *Input*

On initial entry: the number of equations to be solved.

Constraint: NEQ ≥ 1.

2: LDYSAV – INTEGER *Input*

On initial entry: a bound on the maximum number of equations to be solved during the integration.

Constraint: LDYSAV $\geq$ NEQ.

3: T – REAL (KIND=nag_wp) *Input/Output*

On initial entry: t, the value of the independent variable. The input value of T is used only on the first call as the initial point of the integration.

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

4: TOUT – REAL (KIND=nag_wp) *Input/Output*

On initial entry: the next value of t at which a computed solution is desired. For the initial t, the input value of TOUT is used to determine the direction of integration. Integration is permitted in either direction (see also ITASK).

Constraint: TOUT $\neq$ T.

On exit: is unaltered unless ITASK $= 6$ and LDERIV(2) = .TRUE. on entry (see also ITASK and LDERIV) in which case TOUT will be set to the result of taking a small step at the start of the integration.

5: Y(NEQ) – REAL (KIND=nag_wp) array *Input/Output*

On initial entry: the values of the dependent variables (solution). On the first call the first NEQ elements of y must contain the vector of initial values.

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

6: YDOT(NEQ) – REAL (KIND=nag_wp) array *Input/Output*

On initial entry: if LDERIV(1) = .TRUE., YDOT must contain approximations to the time derivatives y' of the vector y. If LDERIV(1) = .FALSE., then YDOT need not be set on entry.

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

7: RWORK$(50 + 4 \times$ NEQ$)$ – REAL (KIND=nag_wp) array *Communication Array*

On initial entry: must be the same array as used by one of the method setup routines D02MVF, D02NVF or D02NWF, and by one of the storage setup routines D02NSF, D02NTF or D02NUF. The contents of RWORK must not be changed between any call to a setup routine and the first call to D02NNF.

On intermediate re-entry: must contain residual evaluations as described under the argument IREVCM.

On intermediate exit: contains information for JAC, RESID and MONITR operations as described under Section 3 and the argument IREVCM.

8: RTOL$(*)$ – REAL (KIND=nag_wp) array *Input*

Note: the dimension of the array RTOL must be at least 1 if ITOL $= 1$ or 2, and at least NEQ otherwise.

On initial entry: the relative local error tolerance.

Constraint: RTOL$(i) \geq 0.0$ for all relevant i (see ITOL).

9: ATOL$(*)$ – REAL (KIND=nag_wp) array *Input*

Note: the dimension of the array ATOL must be at least 1 if ITOL $= 1$ or 3, and at least NEQ otherwise.

On initial entry: the absolute local error tolerance.

Constraint: ATOL$(i) \geq 0.0$ for all relevant i (see ITOL).

10: ITOL – INTEGER *Input*

On initial entry: a value to indicate the form of the local error test. ITOL indicates to D02NNF whether to interpret either or both of RTOL or ATOL as a vector or a scalar. The error test to be satisfied is $\|e_i/w_i\| < 1.0$, where w_i is defined as follows:

ITOL	RTOL	ATOL	w_i
1	scalar	scalar	$\text{RTOL}(1) \times \lvert y_i \rvert + \text{ATOL}(1)$
2	scalar	vector	$\text{RTOL}(1) \times \lvert y_i \rvert + \text{ATOL}(i)$
3	vector	scalar	$\text{RTOL}(i) \times \lvert y_i \rvert + \text{ATOL}(1)$
4	vector	vector	$\text{RTOL}(i) \times \lvert y_i \rvert + \text{ATOL}(i)$

e_i is an estimate of the local error in y_i, computed internally, and the choice of norm to be used is defined by a previous call to an integrator setup routine.

Constraint: ITOL $= 1$, 2, 3 or 4.

11: INFORM(23) – INTEGER array *Communication Array*

12: YSAV(LDYSAV, SDYSAV) – REAL (KIND=nag_wp) array *Communication Array*

13: SDYSAV – INTEGER *Input*

On initial entry: the second dimension of the array YSAV as declared in the (sub)program from which D02NNF is called. An appropriate value for SDYSAV is described in the specifications of the integrator setup routines D02MVF, D02NVF and D02NWF. This value must be the same as that supplied to the integrator setup routine.

14: WKJAC(NWKJAC) – REAL (KIND=nag_wp) array *Input/Output*

On intermediate re-entry: elements of the Jacobian as defined under the description of IREVCM. If a numerical Jacobian was requested then WKJAC is used for workspace.

On intermediate exit: the Jacobian is overwritten.

15: NWKJAC – INTEGER *Input*

On initial entry: the dimension of the array WKJAC as declared in the (sub)program from which D02NNF is called. The actual size depends on the linear algebra method used. An appropriate value for NWKJAC is described in the specifications of the linear algebra setup routines D02NSF, D02NTF and D02NUF for full, banded and sparse matrix linear algebra respectively. This value must be the same as that supplied to the linear algebra setup routine.

16: JACPVT(NJCPVT) – INTEGER array *Communication Array*

17: NJCPVT – INTEGER *Input*

On initial entry: the dimension of the array JACPVT as declared in the (sub)program from which D02NNF is called. The actual size depends on the linear algebra method used. An appropriate value for NJCPVT is described in the specifications of the linear algebra setup routines D02NTF and D02NUF for banded and sparse matrix linear algebra respectively. This value must be the same as that supplied to the linear algebra setup routine. When full matrix linear algebra is chosen, the array JACPVT is not used and hence NJCPVT should be set to 1.

18: IMON – INTEGER *Input/Output*

On intermediate exit: used to pass information between D02NNF and the MONITR operation (see Section 3). With IREVCM $= 9$, IMON contains a flag indicating under what circumstances the return from D02NNF occurred:

IMON $= -2$
: Exit from D02NNF after IRES $= 4$ (set in the RESID operation (see Section 3) caused an early termination (this facility could be used to locate discontinuities).

IMON $= -1$
: The current step failed repeatedly.

IMON $= 0$
: Exit from D02NNF after a call to the internal nonlinear equation solver.

IMON $= 1$
The current step was successful.

On intermediate re-entry: may be reset to determine subsequent action in D02NNF.

IMON $= -2$
Integration is to be halted. A return will be made from D02NNF to the calling (sub) program with IFAIL $= 12$.

IMON $= -1$
Allow D02NNF to continue with its own internal strategy. The integrator will try up to three restarts unless IMON $\neq -1$.

IMON $= 0$
Return to the internal nonlinear equation solver, where the action taken is determined by the value of INLN.

IMON $= 1$
Normal exit to D02NNF to continue integration.

IMON $= 2$
Restart the integration at the current time point. The integrator will restart from order 1 when this option is used. The internal initialization module solves for new values of y and y' by using the values supplied in Y and YDOT by the MONITR operation (see Section 3) as initial estimates.

IMON $= 3$
Try to continue with the same step size and order as was to be used before entering the MONITR operation (see Section 3). HMIN and HMAX may be altered if desired.

IMON $= 4$
Continue the integration but using a new value of HNEXT and possibly new values of HMIN and HMAX.

19: INLN – INTEGER *Input/Output*

On intermediate re-entry: with IMON $= 0$ and IREVCM $= 9$, INLN specifies the action to be taken by the internal nonlinear equation solver. By setting INLN $= 3$ and returning to D02NNF, the residual vector is evaluated and placed in RWORK$(50 + 2 \times \text{NEQ} + i)$, for $i = 1, 2, \ldots, \text{NEQ}$ and then the MONITR operation (see Section 3) is invoked again. At present this is the only option available: INLN must not be set to any other value.

On intermediate exit: contains a flag indicating the action to be taken, if any, by the internal nonlinear equation solver.

20: IRES – INTEGER *Input/Output*

On intermediate exit: with IREVCM $= 1$, 2, 3, 4, 5, 6, 7 or 11, IRES specifies the form of the residual to be returned by the RESID operation (see Section 3).

If IRES $= 1$, then $-r = g(t, y) - A(t, y)y'$ must be returned.

If IRES $= -1$, then $-\hat{r} = -A(t, y)y'$ must be returned.

On intermediate re-entry: should be unchanged unless one of the following actions is required of D02NNF in which case IRES should be set accordingly.

IRES $= 2$
Indicates to D02NNF that control should be passed back immediately to the calling (sub) program with the error indicator set to IFAIL $= 11$.

IRES $= 3$
Indicates to D02NNF that an error condition has occurred in the solution vector, its time derivative or in the value of t. The integrator will use a smaller time step to try to avoid this condition. If this is not possible D02NNF returns to the calling (sub)program with the error indicator set to IFAIL $= 7$.

IRES = 4
: Indicates to D02NNF to stop its current operation and to enter the MONITR operation (see Section 3) immediately.

21: IREVCM – INTEGER *Input/Output*

On initial entry: must contain 0.

On intermediate re-entry: should remain unchanged.

On intermediate exit: indicates what action you must take before re-entering D02NNF. The possible exit values of IREVCM are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 or 11 which should be interpreted as follows:

IREVCM = 1, 2, 3, 4, 5, 6, 7 or 11
: Indicates that a RESID operation (see Section 3) is required: you must supply the residual of the system. For each of these values of IREVCM, y_i is located in Y(i), for $i = 1, 2, \ldots,$ NEQ.

For IREVCM = 1, 3, 6 or 11, y'_i is located in YDOT(i) and r_i should be stored in RWORK($50 + 2 \times$ NEQ $+ i$), for $i = 1, 2, \ldots,$ NEQ.

For IREVCM = 2, y'_i is located in RWORK($50 +$ NEQ $+ i$) and r_i should be stored in RWORK($50 + 2 \times$ NEQ $+ i$), for $i = 1, 2, \ldots,$ NEQ.

For IREVCM = 4 or 7, y'_i is located in YDOT(i) and r_i should be stored in RWORK($50 +$ NEQ $+ i$), for $i = 1, 2, \ldots,$ NEQ.

For IREVCM = 5, y'_i is located in RWORK($50 + 2 \times$ NEQ $+ i$) and r_i should be stored in YDOT(i), for $i = 1, 2, \ldots,$ NEQ.

IREVCM = 8
: Indicates that a JAC operation (see Section 3) is required: you must supply the Jacobian matrix.

If full matrix linear algebra is being used, then the (i, j)th element of the Jacobian must be stored in WKJAC($(j - 1) \times$ NEQ $+ i$).

If banded matrix linear algebra is being used, then the (i, j)th element of the Jacobian must be stored in WKJAC($(i - 1) \times m_B + k$), where $m_B = m_L + m_U + 1$ and $k = \min(m_L - i + 1, 0) + j$; here m_L and m_U are the number of subdiagonals and superdiagonals, respectively, in the band.

If sparse matrix linear algebra is being used, then D02NRF must be called to determine which column of the Jacobian is required and where it should be stored.

```
CALL D02NRF(J, IPLACE, INFORM)
```

: will return in J the number of the column of the Jacobian that is required and will set IPLACE = 1 or 2 (see D02NRF). If IPLACE = 1, you must store the nonzero element (i, j) of the Jacobian in RWORK($50 + 2 \times$ NEQ $+ i$); otherwise it must be stored in RWORK($50 +$ NEQ $+ i$).

IREVCM = 9
: Indicates that a MONITR operation (see Section 3) can be performed.

IREVCM = 10
: Indicates that the current step was not successful, due to error test failure or convergence test failure. The only information supplied to you on this return is the current value of the variable t, located in RWORK(19). No values must be changed before re-entering D02NNF; this facility enables you to determine the number of unsuccessful steps.

On final exit: IREVCM = 0 indicating that the user-specified task has been completed or an error has been encountered (see the descriptions for ITASK and IFAIL).

Constraint: $0 \leq$ IREVCM ≤ 11.

22: LDERIV(2) – LOGICAL array *Input/Output*

On initial entry: LDERIV(1) must be set to .TRUE. if you have supplied both an initial y and an initial y'. LDERIV(1) must be set to .FALSE. if only the initial y has been supplied.

LDERIV(2) must be set to .TRUE. if the integrator is to use a modified Newton method to evaluate the initial y and y'. Note that y and y', if supplied, are used as initial estimates. This method involves taking a small step at the start of the integration, and if ITASK $= 6$ on entry, T and TOUT will be set to the result of taking this small step. LDERIV(2) must be set to .FALSE. if the integrator is to use functional iteration to evaluate the initial y and y', and if this fails a modified Newton method will then be attempted. LDERIV(2) = .TRUE. is recommended if there are implicit equations or the initial y and y' are zero.

On final exit: LDERIV(1) is normally unchanged. However if ITASK $= 6$ and internal initialization was successful then LDERIV(1) = .TRUE..

LDERIV(2) = .TRUE., if implicit equations were detected. Otherwise LDERIV(2) = .FALSE..

23: ITASK – INTEGER *Input*

On initial entry: the task to be performed by the integrator.

ITASK $= 1$
: Normal computation of output values of $y(t)$ at $t =$ TOUT (by overshooting and interpolating).

ITASK $= 2$
: Take one step only and return.

ITASK $= 3$
: Stop at the first internal integration point at or beyond $t =$ TOUT and return.

ITASK $= 4$
: Normal computation of output values of $y(t)$ at $t =$ TOUT but without overshooting $t =$ TCRIT. TCRIT must be specified as an option in one of the integrator setup routines before the first call to the integrator, or specified in the optional input routine before a continuation call. TCRIT (e.g., see D02NVF) may be equal to or beyond TOUT, but not before it in the direction of integration.

ITASK $= 5$
: Take one step only and return, without passing TCRIT (e.g., see D02NVF). TCRIT must be specified under ITASK $= 4$.

ITASK $= 6$
: The integrator will solve for the initial values of y and y' only and then return to the calling (sub)program without doing the integration. This option can be used to check the initial values of y and y'. Functional iteration or a 'small' backward Euler method used in conjunction with a damped Newton iteration is used to calculate these values (see LDERIV). Note that if a backward Euler step is used then the value of t will have been advanced a short distance from the initial point.

Note: if D02NNF is recalled with a different value of ITASK (and TOUT altered) then the initialization procedure is repeated, possibly leading to different initial conditions.

Constraint: $1 \leq$ ITASK ≤ 6.

24: ITRACE – INTEGER *Input*

On initial entry: the level of output that is printed by the integrator. ITRACE may take the value -1, 0, 1, 2 or 3.

ITRACE < -1
: -1 is assumed and similarly if ITRACE > 3, then 3 is assumed.

ITRACE $= -1$
: No output is generated.

ITRACE $= 0$

Only warning messages are printed on the current error message unit (see X04AAF).

ITRACE > 0

Warning messages are printed as above, and on the current advisory message unit (see X04ABF) output is generated which details Jacobian entries, the nonlinear iteration and the time integration. The advisory messages are given in greater detail the larger the value of ITRACE.

25: 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).

Errors or warnings detected by the routine:

IFAIL $= 1$

On entry, the integrator detected an illegal input, or that a linear algebra and/or integrator setup routine has not been called prior to the call to the integrator. If ITRACE ≥ 0, the form of the error will be detailed on the current error message unit (see X04AAF).

IFAIL $= 2$

The maximum number of steps specified has been taken (see the description of optional inputs in the integrator setup routines and the optional input continuation routine, D02NZF).

IFAIL $= 3$

With the given values of RTOL and ATOL no further progress can be made across the integration range from the current point T. The components $\mathrm{Y}(1), \mathrm{Y}(2), \ldots, \mathrm{Y}(\mathrm{NEQ})$ contain the computed values of the solution at the current point T.

IFAIL $= 4$

There were repeated error test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. The problem may have a singularity, or the local error requirements may be inappropriate.

IFAIL $= 5$

There were repeated convergence test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. This may be caused by an inaccurate Jacobian matrix or one which is incorrectly computed.

IFAIL $= 6$

Some error weight w_i became zero during the integration (see the description of ITOL). Pure relative error control ($\mathrm{ATOL}(i) = 0.0$) was requested on a variable (the ith) which has now vanished. The integration was successful as far as T.

IFAIL $= 7$

The RESID operation (see Section 3) set the error flag IRES $= 3$ continually despite repeated attempts by the integrator to avoid this.

IFAIL $= 8$

LDERIV(1) = .FALSE. on entry but the internal initialization routine was unable to initialize y' (more detailed information may be directed to the current error message unit, see X04AAF).

IFAIL $= 9$

A singular Jacobian $\dfrac{\partial r}{\partial y}$ has been encountered. You should check the problem formulation and Jacobian calculation.

IFAIL $= 10$

An error occurred during Jacobian formulation or back-substitution (a more detailed error description may be directed to the current error message unit, see X04AAF).

IFAIL $= 11$

The RESID operation (see Section 3) signalled the integrator to halt the integration and return by setting IRES $= 2$. Integration was successful as far as T.

IFAIL $= 12$

The MONITR operation (see Section 3) set IMON $= -2$ and so forced a return but the integration was successful as far as T.

IFAIL $= 13$

The requested task has been completed, but it is estimated that a small change in RTOL and ATOL is unlikely to produce any change in the computed solution. (Only applies when you are not operating in one step mode, that is when ITASK $\neq 2$ or 5.)

IFAIL $= 14$

The values of RTOL and ATOL are so small that D02NNF is unable to start the integration.

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, and to a much lesser extent by the choice of norm. 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 $= 1$ with ATOL(1) small but positive).

8 Parallelism and Performance

D02NNF is not thread safe and should not be called from a multithreaded user program. Please see Section 3.12.1 in How to Use the NAG Library and its Documentation for more information on thread safety.

D02NNF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

D02NNF 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; also on the type of linear algebra being used. For further details see Section 9 in D02NGF, D02NHF and D02NJF of the documents for D02NGF (full matrix), D02NHF (banded matrix) or D02NJF (sparse matrix).

In general, you are advised to choose the Backward Differentiation Formula option (setup routine D02NVF) but if efficiency is of great importance and especially if it is suspected that $\dfrac{\partial}{\partial y}(A^{-1}g)$ has complex eigenvalues near the imaginary axis for some part of the integration, you should try the BLEND option (setup routine D02NWF).

10 Example

We solve the well-known stiff Robertson problem written as a differential system in implicit form

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

over the range $[0, 10]$ with initial conditions $a = 1.0$ and $b = c = 0.0$ and with scalar error control (ITOL $= 1$). We integrate to the first internal integration point past TOUT $= 10.0$ (ITASK $= 3$), using a BDF method (setup routine D02MVF) and a modified Newton method. We treat the Jacobian as sparse (setup routine D02NUF) and we calculate it analytically. In this program we also illustrate the monitoring of step failures (IREVCM $= 10$) and the forcing of a return when the component falls below 0.9 in the evaluation of the residual by setting IRES $= 2$.

10.1 Program Text

```
    Program d02nnfe

!     D02NNF Example Program Text

!     Mark 26 Release. NAG Copyright 2016.

!     .. Use Statements ..
      Use nag_library, Only: d02mzf, d02nnf, d02nrf, d02nuf, d02nvf, d02nxf,   &
                             d02nyf, nag_wp, x04abf
!     .. Implicit None Statement ..
      Implicit None
!     .. 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    :: one = 1.0_nag_wp
      Real (Kind=nag_wp), Parameter    :: zero = 0.0_nag_wp
      Real (Kind=nag_wp), Parameter    :: gamm2 = 2.0_nag_wp*gamma
      Integer, Parameter               :: iset = 1, itrace = 0, nelts = 8,     &
                                          neq = 3, nia = 1, nin = 5, nja = 1
      Integer, Parameter               :: njcpvt = 20*neq + 12*nelts
      Integer, Parameter               :: nout = 6
      Integer, Parameter               :: nrw = 50 + 4*neq
      Integer, Parameter               :: nwkjac = 4*neq + 12*nelts
      Integer, Parameter               :: ldysav = neq
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: eta, h, h0, hmax, hmin, hu, hxd,     &
                                          sens, t, tcrit, tcur, tolsf, tout, u
      Integer                          :: i, icall, ifail, igrow, imon, imxer, &
                                          indd, indr, inln, iplace, ires,      &
                                          irevcm, isplit, itask, itol, j,      &
                                          liwreq, liwusd, lrwreq, lrwusd,      &
                                          maxord, maxstp, mxhnil, nblock,      &
                                          nfails, ngp, niter, nje, nlu, nnz,   &
                                          nq, nqu, nre, nst, outchn, sdysav
      Logical                          :: lblock, petzld
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: atol(:), rtol(:), rwork(:),          &
                                          wkjac(:), y(:), ydot(:), ysav(:,:)
      Real (Kind=nag_wp)               :: con(6)
      Integer                          :: ia(nia), inform(23), ja(nja)
      Integer, Allocatable             :: jacpvt(:)
      Logical, Allocatable             :: algequ(:)
      Logical                          :: lderiv(2)
!     .. Executable Statements ..
      Write (nout,*) 'D02NNF Example Program Results'
!     Skip heading in data file
      Read (nin,*)
!     neq: number of differential equations
      Read (nin,*) maxord, maxstp, mxhnil
      sdysav = maxord + 1
      Allocate (atol(neq),rtol(neq),rwork(nrw),wkjac(nwkjac),y(neq),ydot(neq), &
        ysav(ldysav,sdysav),jacpvt(njcpvt),algequ(neq))

      outchn = nout
      Write (nout,*)
      Call x04abf(iset,outchn)

!     Integrate towards tout stopping at the first mesh point beyond
!     tout (itask=3) using the B.D.F. formulae with a Newton method.
!     Employ scalar tolerances and the Jacobian is supplied, but its
!     structure is evaluated internally by calls to the Jacobian
!     forming part of the program (irevcm=8). Default values for the
!     array con are used. Also count the number of step failures
!     (irevcm=10). The solution is interpolated using D02MZF to give
!     the solution at tout.

      Read (nin,*) hmin, hmax, h0, tcrit
      Read (nin,*) eta, sens, u
      Read (nin,*) lblock, petzld
      Read (nin,*) t, tout
      Read (nin,*) itol, isplit
      Read (nin,*) y(1:neq)
      Select Case (itol)
      Case (1)
        Read (nin,*) rtol(1), atol(1)
      Case (2)
        Read (nin,*) rtol(1), atol(1:neq)
      Case (3)
        Read (nin,*) rtol(1:neq), atol(1)
      Case (4)
        Read (nin,*) rtol(1:neq), atol(1:neq)
      End Select
```

```
      itask = 3
      lderiv(1:2) = .False.
      con(1:6) = zero
      nfails = 0

!     ifail: behaviour on error exit
!             =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
      ifail = 0
      Call d02nvf(neq,sdysav,maxord,'Newton',petzld,con,tcrit,hmin,hmax,h0,    &
        maxstp,mxhnil,'Average-l2',rwork,ifail)

      ifail = 0
      Call d02nuf(neq,neq,'Analytical',nwkjac,ia,nia,ja,nja,jacpvt,njcpvt,     &
        sens,u,eta,lblock,isplit,rwork,ifail)

      irevcm = 0
      Write (nout,*) '    X          Y(1)          Y(2)          Y(3)'
      Write (nout,99999) t, (y(i),i=1,neq)
      Flush (nout)

revcm: Do
        ifail = -1
        Call d02nnf(neq,ldysav,t,tout,y,ydot,rwork,rtol,atol,itol,inform,ysav, &
          sdysav,wkjac,nwkjac,jacpvt,njcpvt,imon,inln,ires,irevcm,lderiv,      &
          itask,itrace,ifail)

        Select Case (irevcm)
        Case (0)
!         Final exit.
          Exit revcm
        Case (1,3,4,6,7,11)
          If (irevcm==4 .Or. irevcm==7) Then
            indr = 50 + neq
          Else
            indr = 50 + 2*neq
          End If
!         Return residual in rwork(indr+1:indr+neq) using y' in ydot.
          rwork(indr+1) = -ydot(1) - ydot(2) - ydot(3)
          rwork(indr+2) = -ydot(2)
          rwork(indr+3) = -ydot(3)
          If (ires==1) Then
            rwork(indr+1) = rwork(indr+1) + zero
            rwork(indr+2) = rwork(indr+2) + alpha*y(1) - beta*y(2)*y(3) -      &
              gamma*y(2)*y(2)
            rwork(indr+3) = rwork(indr+3) + gamma*y(2)*y(2)
          End If
        Case (2)
!         Return residual in rwork(51+2*neq:) using y' in rwork(51+neq:).
          indd = 50 + neq
          indr = 50 + 2*neq
          rwork(indr+1) = -rwork(indd+1) - rwork(indd+2) - rwork(indd+3)
          rwork(indr+2) = -rwork(indd+2)
          rwork(indr+3) = -rwork(indd+3)
        Case (5)
!         Return residual in ydot, using y' in rwork(51+2*neq:).
          indd = 50 + 2*neq
          ydot(1) = -rwork(indd+1) - rwork(indd+2) - rwork(indd+3)
          ydot(2) = -rwork(indd+2)
          ydot(3) = -rwork(indd+3)
          ydot(1) = ydot(1) + zero
          ydot(2) = ydot(2) + alpha*y(1) - beta*y(2)*y(3) - gamma*y(2)*y(2)
          ydot(3) = ydot(3) + gamma*y(2)*y(2)
        Case (8)
!         Return Jacobian in rwork(51+neq:) or rwork(51+2*neq:).

!         Get index J for Jacoban evaluation.
          Call d02nrf(j,iplace,inform)

          hxd = rwork(16)*rwork(20)
          If (iplace<2) Then
!           return Jacobian in rwork(51+2*neq:).
```

```
            indr = 50 + 2*neq
          Else
!           return Jacobian in rwork(51+neq:).
            indr = 50 + neq
          End If
!         8 nonzero elements in Jacobian.
          If (j<2) Then
            rwork(indr+1) = one - hxd*(zero)
            rwork(indr+2) = zero - hxd*(alpha)
!           rwork(indr+3) = zero - hxd*(zero)
          Else If (j==2) Then
            rwork(indr+1) = one - hxd*(zero)
            rwork(indr+2) = one - hxd*(-beta*y(3)-gamm2*y(2))
            rwork(indr+3) = zero - hxd*(gamm2*y(2))
          Else If (j>2) Then
            rwork(indr+1) = one - hxd*(zero)
            rwork(indr+2) = zero - hxd*(-beta*y(2))
            rwork(indr+3) = one - hxd*(zero)
          End If
        Case (10)
!         Step failure
          nfails = nfails + 1
        End Select
      End Do revcm

!     Print solution and statistics.
      If (ifail==0) Then

        Call d02nyf(neq,neq,hu,h,tcur,tolsf,rwork,nst,nre,nje,nqu,nq,niter,    &
          imxer,algequ,inform,ifail)

        ifail = 0
        Call d02mzf(tout,y,neq,ldysav,neq,ysav,sdysav,rwork,ifail)

        Write (nout,99999) tout, (y(i),i=1,neq)
        Write (nout,99997) hu, h, tcur
        Write (nout,99996) nst, nre, nje
        Write (nout,99995) nqu, nq, niter
        Write (nout,99994) imxer, nfails
        icall = 0

        Call d02nxf(icall,liwreq,liwusd,lrwreq,lrwusd,nlu,nnz,ngp,isplit,      &
          igrow,lblock,nblock,inform)

        Write (nout,99993) liwreq, liwusd
        Write (nout,99992) lrwreq, lrwusd
        Write (nout,99991) nlu, nnz
        Write (nout,99990) ngp, isplit
        Write (nout,99989) igrow, nblock
      Else If (ifail==10) Then
        icall = 1

        Call d02nxf(icall,liwreq,liwusd,lrwreq,lrwusd,nlu,nnz,ngp,isplit,      &
          igrow,lblock,nblock,inform)

        Write (nout,99993) liwreq, liwusd
        Write (nout,99992) lrwreq, lrwusd
      Else
        Write (nout,99998) ifail, t
      End If

99999 Format (1X,F8.3,3(F13.5,2X))
99998 Format (/,1X,'Exit D02NNF with IFAIL = ',I5,'  and T = ',E12.5)
99997 Format (/,1X,' HUSED = ',E12.5,'  HNEXT = ',E12.5,'  TCUR = ',E12.5)
99996 Format (1X,' NST = ',I6,'    NRE = ',I6,'    NJE = ',I6)
99995 Format (1X,' NQU = ',I6,'    NQ  = ',I6,'  NITER = ',I6)
99994 Format (1X,' Max err comp = ',I4,'   No. of failed steps = ',I4)
99993 Format (/,1X,' NJCPVT (required ',I4,'  used ',I8,')')
```

```
99992 Format (1X,' NWKJAC (required ',I4,'  used ',I8,')')
99991 Format (1X,' No. of LU-decomps ',I4,'  No. of nonzeros ',I8)
99990 Format (1X,' No. of FCN calls to form Jacobian ',I4,'  Try ISPLIT ',I4)
99989 Format (1X,' Growth est ',I8,'  No. of blocks on diagonal ',I4)
    End Program d02nnfe
```

10.2 Program Data

```
D02NNF Example Program Data
   5  200  5                        : maxord, maxstp, mxhnil
   1.0E-10  10.0  1.0E-4  0.0       : hmin, hmax, h0, tcrit
   1.0E-4  1.0E-6  0.1              : eta, sens, u
   .TRUE. .TRUE.                    : lblock, petzld
   0.0  10.0                        : t, tout
   1  0                             : itol, isplit
   1.0  0.0  0.0                    : y(1:neq)
   1.0E-4  1.0E-7                   : rtol(1), atol(1)
```

10.3 Program Results

```
 D02NNF Example Program Results

     X          Y(1)           Y(2)           Y(3)
    0.000      1.00000        0.00000        0.00000
 Warning: Equation(=i1) and possibly other equations are
 implicit and in calculating the initial values the
 equations will be treated as implicit.
 In above message i1 =         1
   10.000      0.84136        0.00002        0.15863

  HUSED =  0.81503E+00  HNEXT =  0.12467E+01  TCUR =  0.10409E+02
  NST =     51    NRE =    130    NJE =     14
  NQU =      4    NQ  =      4  NITER =    121
  Max err comp =    3   No. of failed steps =    0

  NJCPVT (required  105  used      156)
  NWKJAC (required   34  used       79)
  No. of LU-decomps   14  No. of nonzeros        9
  No. of FCN calls to form Jacobian    0  Try ISPLIT   73
  Growth est     1386  No. of blocks on diagonal    1
```