NAG Library Routine Document

D02NJF

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

D02NJF is a forward communication routine for integrating stiff systems of implicit ordinary differential equations coupled with algebraic equations when the Jacobian is a sparse matrix.

2 Specification

```
 SUBROUTINE D02NJF(NEQ, LDYSAV, T, TOUT, Y, YDOT, RWORK, RTOL, ATOL,
1                  ITOL, INFORM, RESID, YSAV, SDYSAV, JAC, WKJAC, NWKJAC,
2                  JACPVT, NJCPVT, MONITR, LDERIV, ITASK, ITRACE, IFAIL)
 INTEGER           NEQ, LDYSAV, ITOL, INFORM(23), SDYSAV, NWKJAC,
1                  JACPVT(NJCPVT), NJCPVT, ITASK, ITRACE, IFAIL
 double precision  T, TOUT, Y(NEQ), YDOT(NEQ), RWORK(50+4*NEQ), RTOL(*),
1                  ATOL(*), YSAV(LDYSAV,SDYSAV), WKJAC(NWKJAC)
 LOGICAL           LDERIV(2)
 EXTERNAL          RESID, JAC, MONITR
```

3 Description

D02NJF 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

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

It is designed specifically for the case where the resulting Jacobian is a sparse matrix (see the description of JAC).

Both interval and step oriented modes of operation are available and also modes designed to permit intermediate output within an interval oriented mode.

An outline of a typical calling program for D02NJF is given below. It calls the sparse matrix linear algebra setup routine D02NUF, the Backward Differentiation Formula (BDF) integrator setup routine D02NVF, its diagnostic counterpart D02NYF, and the sparse matrix linear algebra diagnostic routine D02NXF.

```
C
C     declarations
C
      EXTERNAL RESID, JAC, MONITR
          .
          .
          .
      IFAIL = 0
      CALL D02NVF(...,IFAIL)
      CALL D02NUF(NEQ, NEQMAX, JCEVAL, NWKJAC, IA, NIA, JA, NJA,
     + JACPVT, NJCPVT, SENS, U, ETA, LBLOCK, ISPLIT, RWORK,
     + IFAIL)
      IFAIL = -1
      CALL D02NJF(NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
     + ATOL, ITOL, INFORM, RESID, YSAVE, NY2DIM, JAC, WKJAC,
     + NWKJAC, JACPVT, NJCPVT, MONITR, LDERIV, ITASK, ITRACE,
     + IFAIL)
      IF(IFAIL.EQ.1 .OR. IFAIL.GE.14) STOP
      IFAIL = 0
      CALL D02NXF(...)
      CALL D02NYF(...)
```

```
      .
      .
      .
STOP
END
```

The linear algebra setup routine D02NUF and one of the integrator setup routines, D02MVF, D02NVF or D02NWF, must be called prior to the call of D02NJF. Either or both of the integrator diagnostic routine D02NYF, or the sparse matrix linear algebra diagnostic routine D02NXF, may be called after the call to D02NJF. There is also a routine, D02NZF, designed to permit you to change step size on a continuation call to D02NJF without restarting the integration process.

4 References

See the D02M–N sub-chapter Introduction.

5 Parameters

1: NEQ – INTEGER *Input*

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

Constraint: NEQ ≥ 1.

2: LDYSAV – INTEGER *Input*

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

Constraint: LDYSAV $\geq$ NEQ.

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

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

4: TOUT – ***double precision*** *Input/Output*

On 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).

On exit: normally unchanged. However, when ITASK $= 6$, then TOUT contains the value of T at which initial values have been computed without performing any integration. See descriptions of ITASK and LDERIV.

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

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

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

On entry: if LDERIV(1) = .TRUE., YDOT must contain approximations to the time derivatives y' of the vector y.

If LDERIV(1) = .FALSE., YDOT need not be set on entry.

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

7: RWORK$(50 + 4 \times$ NEQ$)$ – ***double precision*** array *Communication Array*

8: RTOL$(*)$ – ***double precision*** array *Input*

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

On entry: the relative local error tolerance.

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

9: ATOL$(*)$ – ***double precision*** array *Input*

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

On entry: the absolute local error tolerance.

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

10: ITOL – INTEGER *Input*

On entry: a value to indicate the form of the local error test. ITOL indicates to D02NJF 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	RTOL$(1) \times \|y_i\| +$ ATOL(1)
2	scalar	vector	RTOL$(1) \times \|y_i\| +$ ATOL(i)
3	vector	scalar	RTOL$(i) \times \|y_i\| +$ ATOL(1)
4	vector	vector	RTOL$(i) \times \|y_i\| +$ 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: RESID – SUBROUTINE, supplied by the user. *External Procedure*

RESID must evaluate the residual

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

in one case and

$$r = -A(t, y)y'$$

in another.

The specification of RESID is:

```
SUBROUTINE RESID(NEQ, T, Y, YDOT, R, IRES)
INTEGER           NEQ, IRES
double precision  T, Y(NEQ), YDOT(NEQ), R(NEQ)
```

1: NEQ – INTEGER *Input*

On entry: the number of 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: the value of y_i, for $i = 1, 2, \ldots,$ NEQ.

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

On entry: the value of y_i' at t, for $i = 1, 2, \ldots, \text{NEQ}$.

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

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

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

or

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

and where the definition of r is determined by the input value of IRES.

6: IRES – INTEGER *Input/Output*

On entry: the form of the residual that must be returned in array R.

IRES $= -1$

The residual defined in equation (2) must be returned.

IRES $= 1$

The residual defined in equation (1) must be returned.

On exit: should be unchanged unless one of the following actions is required of the integrator, in which case IRES should be set accordingly.

IRES $= 2$

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

IRES $= 3$

Indicates to the integrator 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, the integrator returns to the calling (sub)program with the error indicator set to IFAIL $= 7$.

IRES $= 4$

Indicates to the integrator to stop its current operation and to enter MONITR immediately with parameter IMON $= -2$.

RESID must be declared as EXTERNAL in the (sub)program from which D02NJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: YSAV(LDYSAV,SDYSAV) – ***double precision*** array *Communication Array*

14: SDYSAV – INTEGER *Input*

On entry: the second dimension of the array YSAV as declared in the (sub)program from which D02NJF 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.

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

JAC must evaluate the Jacobian of the system. If this option is not required, the actual argument for JAC must be the dummy routine D02NJZ. D02NJZ 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 sparse linear algebra setup routine D02NUF.

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

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

where h is the current step size and d is a parameter 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 it is solved in the form

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

where r is the function defined by

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

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

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

The specification of JAC is:

```
SUBROUTINE JAC(NEQ, T, Y, YDOT, H, D, J, PDJ)
INTEGER          NEQ, J
double precision T, Y(NEQ), YDOT(NEQ), H, D, PDJ(NEQ)
```

1: NEQ – INTEGER *Input*

On entry: the number of 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: H – ***double precision*** *Input*

On entry: the current step size.

6: D – ***double precision*** *Input*

On entry: the parameter d which depends on the integration method.

7: J – INTEGER *Input*

On entry: the column of the Jacobian that JAC must return in the array PDJ.

8: PDJ(NEQ) – ***double precision*** array *Input/Output*

On entry: is set to zero.

On exit: PDJ(i) should be set to the (i,j)th element of the Jacobian, where j is given by J. Only nonzero elements of this array need be set, since it is preset to zero before the call to JAC.

JAC must be declared as EXTERNAL in the (sub)program from which D02NJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

16: WKJAC(NWKJAC) – ***double precision*** array *Communication Array*
17: NWKJAC – INTEGER *Input*

On entry: the dimension of the array WKJAC as declared in the (sub)program from which D02NJF is called. The actual size depends on whether the sparsity structure is supplied or whether it is to be estimated. An appropriate value for NWKJAC is described in the specification of the linear algebra setup routine D02NUF. This value must be the same as that supplied to D02NUF.

18: JACPVT(NJCPVT) – INTEGER array *Communication Array*
19: NJCPVT – INTEGER *Input*

On entry: the dimension of the array JACPVT as declared in the (sub)program from which D02NJF is called. The actual size depends on whether the sparsity structure is supplied or whether it is to be estimated. An appropriate value for NJCPVT is described in the specification for the linear algebra setup routine D02NUF. This value must be same as that supplied to D02NUF.

20: MONITR – SUBROUTINE, supplied by the NAG Library or the user. *External Procedure*

MONITR performs tasks requested by you. If this option is not required, then the actual argument for MONITR must be the dummy routine D02NBY. D02NBY is included in the NAG Library.

The specification of MONITR is:

```
      SUBROUTINE MONITR(NEQ, LDYSAV, T, HLAST, HNEXT, Y, YDOT, YSAV, R,
     1                  ACOR, IMON, INLN, HMIN, HMAX, NQU)
      INTEGER            NEQ, LDYSAV, IMON, INLN, NQU
      double precision   T, HLAST, HNEXT, Y(NEQ), YDOT(NEQ),
     1                   YSAV(LDYSAV,sdysav), R(NEQ), ACOR(NEQ,2), HMIN,
     2                   HMAX
```

where $sdysav$ is the numerical value of SDYSAV in the call of D02NJF.

1: NEQ – INTEGER *Input*

On entry: the number of equations being solved.

2: LDYSAV – INTEGER *Input*

On entry: an upper bound on the number of equations to be solved.

3: T – ***double precision*** *Input*

On entry: the current value of the independent variable.

4: HLAST – ***double precision*** *Input*

On entry: the last step size successfully used by the integrator.

5: HNEXT – ***double precision*** *Input/Output*

On entry: the step size that the integrator proposes to take on the next step.

On exit: the next step size to be used. If this is different from the input value, then IMON must be set to 4.

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

On entry: y, the values of the dependent variables evaluated at t.

On exit: these values must not be changed unless IMON is set to 2.

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

On entry: the time derivatives y' of the vector y.

8: YSAV(LDYSAV,$sdysav$) – ***double precision*** array *Input*

On entry: workspace to enable you to carry out interpolation using either of the routines D02XJF or D02XKF.

9: R(NEQ) – ***double precision*** array *Input*

On entry: if IMON $= 0$ and INLN $= 3$, then the first NEQ elements contain the residual vector $A(t,y)y' - g(t,y)$.

10: ACOR(NEQ,2) – ***double precision*** array *Input*

On entry: with IMON $= 1$, ACOR$(i,1)$ contains the weight used for the ith equation when the norm is evaluated, and ACOR$(i,2)$ contains the estimated local error for the ith equation. The scaled local error at the end of a timestep may be obtained by calling the ***double precision*** function D02ZAF as follows:

```
      IFAIL = 1
      ERRLOC = D02ZAF(NEQ, ACOR(1,2), ACOR(1,1), IFAIL)
C     CHECK IFAIL BEFORE PROCEEDING
```

11: IMON – INTEGER *Input/Output*

On entry: a flag indicating under what circumstances MONITR was called:

IMON $= -2$
: Entry from the integrator after IRES $= 4$ (set in RESID) caused an early termination (this facility could be used to locate discontinuities).

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

IMON $= 0$
: Entry after a call to the internal nonlinear equation solver (see INLN).

IMON $= 1$
: The current step was successful.

On exit: may be reset to determine subsequent action in D02NJF.

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

IMON $= -1$
: Allow the integrator to continue with its own internal strategy. The integrator will try up to three restarts unless IMON is set $\neq -1$ on exit.

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

IMON $= 1$
: Normal exit to the integrator 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 solution Y, provided by MONITR, will be used for the initial conditions.

IMON $= 3$
: Try to continue with the same step size and order as was to be used before the call to MONITR. 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.

12: INLN – INTEGER *Output*

On exit: the action to be taken by the internal nonlinear equation solver when MONITR is exited with IMON = 0. By setting INLN = 3 and returning to the integrator, the residual vector is evaluated and placed in the array R, and then MONITR is called again. At present this is the only option available: INLN must not be set to any other value.

13: HMIN – ***double precision*** *Input/Output*

On entry: the minimum step size to be taken on the next step.

On exit: the minimum step size to be used. If this is different from the input value, then IMON must be set to 3 or 4.

14: HMAX – ***double precision*** *Input/Output*

On entry: the maximum step size to be taken on the next step.

On exit: the maximum step size to be used. If this is different from the input value, then IMON must be set to 3 or 4. If HMAX is set to zero, no limit is assumed.

15: NQU – INTEGER *Input*

On entry: the order of the integrator used on the last step. This is supplied to enable you to carry out interpolation using either of the routines D02XJF or D02XKF.

MONITR must be declared as EXTERNAL in the (sub)program from which D02NJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

On 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 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..

22: ITASK – INTEGER *Input*

On 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 prior to the first call to the integrator, or specified in the optional input routine prior to a continuation call. TCRIT 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. TCRIT must be specified as 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 D02NJF 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.

23: ITRACE – INTEGER *Input*

On 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.

24: 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).

Errors or warnings detected by the routine:

IFAIL $= 1$

An illegal input was detected on entry, or after an internal call to MONITR. If ITRACE > -1, then 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 $= 7$

RESID set its 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

RESID signalled the integrator to halt the integration and return (IRES = 2). Integration was successful as far as T.

IFAIL = 12

MONITR 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 ≠ 2 or 5.)

IFAIL = 14

The values of RTOL and ATOL are so small that D02NJF is unable to start the integration.

IFAIL = 15

The linear algebra setup routine D02NUF was not called before the call to D02NJF.

7 Accuracy

The accuracy of the numerical solution may be controlled by a careful choice of the parameters 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 Further Comments

Since numerical stability and memory are often conflicting requirements when solving ordinary differential systems where the Jacobian matrix is sparse we provide a diagnostic routine, D02NXF, whose aim is to inform you how much memory is required to solve the problem and to give you some indicators of numerical stability.

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

9 Example

This example solves the well-known stiff Robertson problem written as a mixed differential/algebraic system in implicit form

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

exploiting the fact that, from the initial conditions $a = 1.0$ and $b = c = 0.0$, we know that $a + b + c = 1$ for all time. We integrate over the range $[0, 10.0]$ with vector relative error control and scalar absolute error control (ITOL = 3) and using the BDF integrator (setup routine D02NVF) and a modified Newton method. The Jacobian is evaluated, in turn, using the 'A' (Analytical) and 'F' (Full information) options. We provide a monitor routine to terminate the integration when the value of the component a falls below 0.9.

9.1 Program Text

```
*     D02NJF Example Program Text
*     Mark 14 Revised. NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, NEQMAX, NRW, NINF, NELTS, NJCPVT, NWKJAC,
     +                 NIA, NJA, MAXORD, SDYSAV, MAXSTP, MXHNIL, LDYSAV
      PARAMETER        (NEQ=3,NEQMAX=NEQ,NRW=50+4*NEQMAX,NINF=23,
     +                 NELTS=8,NJCPVT=150,NWKJAC=100,NIA=NEQ+1,
     +                 NJA=NELTS,MAXORD=5,SDYSAV=MAXORD+1,MAXSTP=200,
     +                 MXHNIL=5,LDYSAV=NEQ)
      DOUBLE PRECISION H0, HMAX, HMIN, TCRIT
      PARAMETER        (H0=0.0D0,HMAX=10.0D0,HMIN=1.0D-10,TCRIT=0.0D0)
      LOGICAL          PETZLD
      PARAMETER        (PETZLD=.TRUE.)
      DOUBLE PRECISION ETA, U, SENS
      PARAMETER        (ETA=1.0D-4,U=0.1D0,SENS=1.0D-6)
      LOGICAL          LBLOCK
      PARAMETER        (LBLOCK=.TRUE.)
*     .. Local Scalars ..
      DOUBLE PRECISION H, HU, T, TCUR, TOLSF, TOUT
      INTEGER          I, ICALL, IFAIL, IGROW, IMXER, ISPLIT, ITASK,
     +                 ITOL, ITRACE, LIWREQ, LIWUSD, LRWREQ, LRWUSD,
     +                 NBLOCK, NGP, NITER, NJE, NLU, NNZ, NQ, NQU, NRE,
     +                 NST, OUTCHN
*     .. Local Arrays ..
      DOUBLE PRECISION ATOL(NEQ), CONST(6), RTOL(NEQ), RWORK(NRW),
     +                 WKJAC(NWKJAC), Y(NEQ), YDOT(NEQ),
     +                 YSAV(LDYSAV,SDYSAV)
      INTEGER          IA(NIA), INFORM(NINF), JA(NJA), JACPVT(NJCPVT)
      LOGICAL          ALGEQU(NEQ), LDERIV(2)
*     .. External Subroutines ..
      EXTERNAL         D02NJF, D02NUF, D02NVF, D02NXF, D02NYF, JAC,
     +                 MONITR, RESID, X04ABF
*     .. Data statements ..
      DATA             IA/1, 3, 6, 9/, JA/1, 2, 1, 2, 3, 1, 2, 3/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02NJF Example Program Results'
      OUTCHN = NOUT
      CALL X04ABF(1,OUTCHN)
*
*     First case. Integrate to TOUT by overshooting (ITASK=1) using
*     B.D.F formulae with a Newton method. Also set PETZLD to
*     .TRUE. so that the Petzold error test is used (since an algebraic
*     equation is defined in the system). Default values for the
*     array CONST are used. Employ vector relative tolerance and scalar
*     absolute tolerance. The Jacobian is supplied by JAC and its
*     structure is determined internally by calls to JAC.
*     The MONITR routine is used to force a return when the first
*     component of the system falls below the value 0.9.
*
      T = 0.0D0
      TOUT = 10.0D0
      ITASK = 1
      Y(1) = 1.0D0
      Y(2) = 0.0D0
      Y(3) = 0.0D0
      LDERIV(1) = .FALSE.
      LDERIV(2) = .FALSE.
      ITOL = 3
      RTOL(1) = 1.0D-4
      RTOL(2) = 1.0D-3
      RTOL(3) = 1.0D-4
      ATOL(1) = 1.0D-7
      DO 20 I = 1, 6
         CONST(I) = 0.0D0
   20 CONTINUE
      ISPLIT = 0
      IFAIL = 1
```

```
*
      CALL D02NVF(NEQMAX,SDYSAV,MAXORD,'Newton',PETZLD,CONST,TCRIT,HMIN,
     +            HMAX,H0,MAXSTP,MXHNIL,'Average-L2',RWORK,IFAIL)
      IF (IFAIL.NE.0) THEN
         WRITE (NOUT,99988) IFAIL
         GO TO 40
      END IF
      IFAIL = 0
      CALL D02NUF(NEQ,NEQMAX,'Analytical',NWKJAC,IA,NIA,JA,NJA,JACPVT,
     +            NJCPVT,SENS,U,ETA,LBLOCK,ISPLIT,RWORK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '  Analytic Jacobian, structure not supplied'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '    X          Y(1)           Y(2)           Y(3)'
      WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
*
*     Soft fail and error messages only
      ITRACE = 0
      IFAIL = 1
*
      CALL D02NJF(NEQ,LDYSAV,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            RESID,YSAV,SDYSAV,JAC,WKJAC,NWKJAC,JACPVT,NJCPVT,
     +            MONITR,LDERIV,ITASK,ITRACE,IFAIL)
*
      IF (IFAIL.EQ.0 .OR. IFAIL.EQ.12) THEN
         WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
         IFAIL = 0
*
         CALL D02NYF(NEQ,NEQMAX,HU,H,TCUR,TOLSF,RWORK,NST,NRE,NJE,NQU,
     +               NQ,NITER,IMXER,ALGEQU,INFORM,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99997) HU, H, TCUR
         WRITE (NOUT,99996) NST, NRE, NJE
         WRITE (NOUT,99995) NQU, NQ, NITER
         WRITE (NOUT,99994) ' Max err comp = ', IMXER
         ICALL = 0
*
         CALL D02NXF(ICALL,LIWREQ,LIWUSD,LRWREQ,LRWUSD,NLU,NNZ,NGP,
     +               ISPLIT,IGROW,LBLOCK,NBLOCK,INFORM)
*
         WRITE (NOUT,*)
         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.EQ.10) THEN
         ICALL = 1
*
         CALL D02NXF(ICALL,LIWREQ,LIWUSD,LRWREQ,LRWUSD,NLU,NNZ,NGP,
     +               ISPLIT,IGROW,LBLOCK,NBLOCK,INFORM)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99993) LIWREQ, LIWUSD
         WRITE (NOUT,99992) LRWREQ, LRWUSD
      ELSE
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Exit D02NJF with IFAIL = ', IFAIL,
     +     '  and T = ', T
      END IF
*
*     Second case. Integrate to TOUT by overshooting (ITASK=1) using
*     B.D.F formulae with a Newton method. Also set PETZLD to
*     .TRUE. so that the Petzold error test is used (since an algebraic
*     equation is defined in the system). Default values for the
*     array CONST are used. Employ vector relative tolerance and scalar
*     absolute tolerance. The Jacobian is supplied by JAC and its
*     structure is also supplied.
*     The MONITR routine is used to force a return when the first
```

```
*     component of the system falls below the value 0.9.
*
      T = 0.0D0
      Y(1) = 1.0D0
      Y(2) = 0.0D0
      Y(3) = 0.0D0
*
      ISPLIT = 0
      IFAIL = 1
*
      CALL D02NVF(NEQMAX,SDYSAV,MAXORD,'Newton',PETZLD,CONST,TCRIT,HMIN,
     +            HMAX,H0,MAXSTP,MXHNIL,'Average-L2',RWORK,IFAIL)
*
      IF (IFAIL.NE.0) THEN
         WRITE (NOUT,99993) IFAIL
         GO TO 40
      END IF
      IFAIL = 0
      CALL D02NUF(NEQ,NEQMAX,'Full information',NWKJAC,IA,NIA,JA,NJA,
     +            JACPVT,NJCPVT,SENS,U,ETA,LBLOCK,ISPLIT,RWORK,IFAIL)
*
      LDERIV(1) = .FALSE.
      LDERIV(2) = .FALSE.
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '  Analytic Jacobian, structure supplied'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '    X          Y(1)           Y(2)           Y(3)'
      WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
      IFAIL = 1
*
      CALL D02NJF(NEQ,LDYSAV,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            RESID,YSAV,SDYSAV,JAC,WKJAC,NWKJAC,JACPVT,NJCPVT,
     +            MONITR,LDERIV,ITASK,ITRACE,IFAIL)
*
      IF (IFAIL.EQ.0 .OR. IFAIL.EQ.12) THEN
         WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
         IFAIL = 0
*
         CALL D02NYF(NEQ,NEQMAX,HU,H,TCUR,TOLSF,RWORK,NST,NRE,NJE,NQU,
     +               NQ,NITER,IMXER,ALGEQU,INFORM,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99997) HU, H, TCUR
         WRITE (NOUT,99996) NST, NRE, NJE
         WRITE (NOUT,99995) NQU, NQ, NITER
         WRITE (NOUT,99994) ' Max err comp = ', IMXER
         WRITE (NOUT,*)
         ICALL = 0
*
         CALL D02NXF(ICALL,LIWREQ,LIWUSD,LRWREQ,LRWUSD,NLU,NNZ,NGP,
     +               ISPLIT,IGROW,LBLOCK,NBLOCK,INFORM)
*
         WRITE (NOUT,*)
         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.EQ.10) THEN
         ICALL = 1
*
         CALL D02NXF(ICALL,LIWREQ,LIWUSD,LRWREQ,LRWUSD,NLU,NNZ,NGP,
     +               ISPLIT,IGROW,LBLOCK,NBLOCK,INFORM)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99993) LIWREQ, LIWUSD
         WRITE (NOUT,99992) LRWREQ, LRWUSD
      ELSE
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Exit D02NJF with IFAIL = ', IFAIL,
```

```
     +     '  and T = ', T
      END IF
   40 CONTINUE
*
99999 FORMAT (1X,F8.3,3(F13.5,2X))
99998 FORMAT (1X,A,I2,A,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,A,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 ISPLI',
     +       'T ',I4)
99989 FORMAT (1X,' Growth est ',I8,'  No. of blocks on diagonal ',I4)
99988 FORMAT (1X,/1X,' ** D02NVF returned with IFAIL = ',I5)
      END
*
      SUBROUTINE RESID(NEQ,T,Y,YDOT,R,IRES)
*     .. Scalar Arguments ..
      DOUBLE PRECISION T
      INTEGER          IRES, NEQ
*     .. Array Arguments ..
      DOUBLE PRECISION R(NEQ), Y(NEQ), YDOT(NEQ)
*     .. Executable Statements ..
      R(1) = 0.0D0
      R(2) = -YDOT(2)
      R(3) = -YDOT(3)
      IF (IRES.EQ.1) THEN
         R(1) = Y(1) + Y(2) + Y(3) - 1.0D0 + R(1)
         R(2) = 0.04D0*Y(1) - 1.0D4*Y(2)*Y(3) - 3.0D7*Y(2)*Y(2) + R(2)
         R(3) = 3.0D7*Y(2)*Y(2) + R(3)
      END IF
      RETURN
      END
*
      SUBROUTINE JAC(NEQ,T,Y,YDOT,H,D,J,PDJ)
*     .. Scalar Arguments ..
      DOUBLE PRECISION D, H, T
      INTEGER          J, NEQ
*     .. Array Arguments ..
      DOUBLE PRECISION PDJ(NEQ), Y(NEQ), YDOT(NEQ)
*     .. Local Scalars ..
      DOUBLE PRECISION HXD
*     .. Executable Statements ..
      HXD = H*D
      IF (J.EQ.1) THEN
         PDJ(1) = 0.0D0 - HXD*(1.0D0)
         PDJ(2) = 0.0D0 - HXD*(0.04D0)
*        PDJ(3) = 0.0 - HXD*(0.)
      ELSE IF (J.EQ.2) THEN
         PDJ(1) = 0.0D0 - HXD*(1.0D0)
         PDJ(2) = 1.0D0 - HXD*(-1.0D4*Y(3)-6.0D7*Y(2))
         PDJ(3) = 0.0D0 - HXD*(6.0D7*Y(2))
      ELSE IF (J.EQ.3) THEN
         PDJ(1) = 0.0D0 - HXD*(1.0D0)
         PDJ(2) = 0.0D0 - HXD*(-1.0D4*Y(2))
         PDJ(3) = 1.0D0 - HXD*(0.0D0)
      END IF
      RETURN
      END
*
      SUBROUTINE MONITR(N,LDYSAV,T,HLAST,H,Y,YDOT,YSAV,R,ACOR,IMON,INLN,
     +                  HMIN,HMXI,NQU)
*     .. Scalar Arguments ..
      DOUBLE PRECISION H, HLAST, HMIN, HMXI, T
      INTEGER          IMON, INLN, LDYSAV, N, NQU
*     .. Array Arguments ..
```

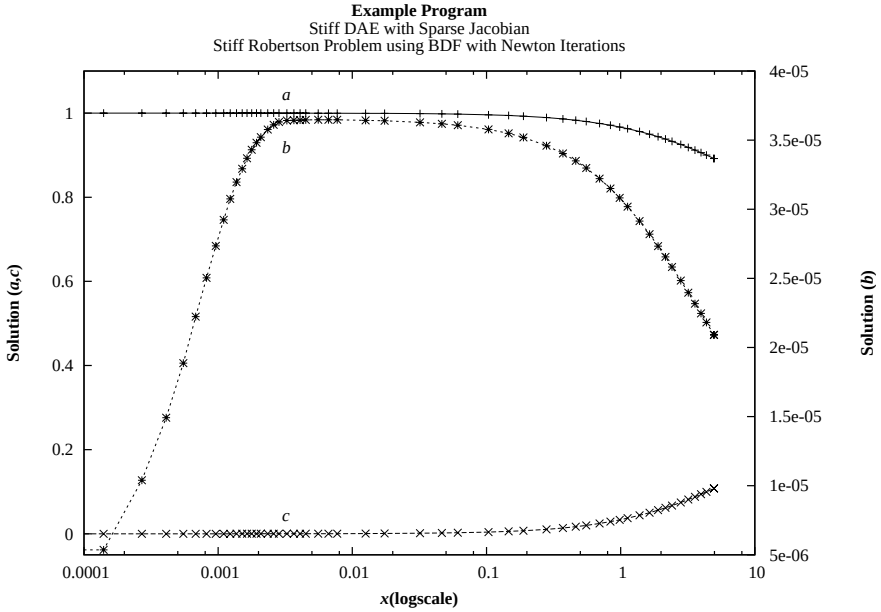
```
      DOUBLE PRECISION ACOR(N,2), R(N), Y(N), YDOT(N), YSAV(LDYSAV,*)
*     .. Executable Statements ..
      IF (Y(1).LE.0.9D0) IMON = -2
      RETURN
      END
```

9.2 Program Data

None.

9.3 Program Results

```
 D02NJF Example Program Results

   Analytic Jacobian, structure not supplied

     X          Y(1)           Y(2)           Y(3)
    0.000      1.00000        0.00000        0.00000
    4.862      0.89332        0.00002        0.10666

  HUSED =  0.61574E+00  HNEXT =  0.61574E+00  TCUR =  0.48624E+01
  NST =     50    NRE =    144    NJE =     15
  NQU =      4    NQ  =      4  NITER =    129
  Max err comp =    3

  NJCPVT (required   93  used      150)
  NWKJAC (required   29  used       76)
  No. of LU-decomps   15  No. of nonzeros        7
  No. of FCN calls to form Jacobian    0  Try ISPLIT   73
  Growth est   140290  No. of blocks on diagonal    1

   Analytic Jacobian, structure supplied

     X          Y(1)           Y(2)           Y(3)
    0.000      1.00000        0.00000        0.00000
    4.957      0.89208        0.00002        0.10790

  HUSED =  0.59971E+00  HNEXT =  0.59971E+00  TCUR =  0.49566E+01
  NST =     52    NRE =    131    NJE =     12
  NQU =      4    NQ  =      4  NITER =    117
  Max err comp =    3


  NJCPVT (required   99  used      150)
  NWKJAC (required   31  used       75)
  No. of LU-decomps   12  No. of nonzeros        8
  No. of FCN calls to form Jacobian    0  Try ISPLIT   73
  Growth est     1034  No. of blocks on diagonal    1
```


Example Program
Stiff DAE with Sparse Jacobian
Stiff Robertson Problem using BDF with Newton Iterations
a
b
c
Solution (a,c)
Solution (b)
x(logscale)
0
0.2
0.4
0.6
0.8
1
5e-06
1e-05
1.5e-05
2e-05
2.5e-05
3e-05
3.5e-05
4e-05
0.0001
0.001
0.01
0.1
1
10