NAG Library Routine Document

D02NDF

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

D02NDF is a direct communication routine for integrating stiff systems of explicit ordinary differential equations when the Jacobian is a sparse matrix.

2 Specification

```
SUBROUTINE D02NDF (NEQ, LDYSAV, T, TOUT, Y, YDOT, RWORK, RTOL, ATOL,       &
                   ITOL, INFORM, FCN, YSAV, SDYSAV, JAC, WKJAC, NWKJAC,    &
                   JACPVT, NJCPVT, MONITR, ITASK, ITRACE, IFAIL)

INTEGER            NEQ, LDYSAV, ITOL, INFORM(23), SDYSAV, NWKJAC,          &
                   JACPVT(NJCPVT), NJCPVT, ITASK, ITRACE, IFAIL
REAL (KIND=nag_wp) T, TOUT, Y(NEQ), YDOT(NEQ), RWORK(50+4*NEQ),            &
                   RTOL(*), ATOL(*), YSAV(LDYSAV,SDYSAV),                  &
                   WKJAC(NWKJAC)
EXTERNAL           FCN, JAC, MONITR
```

3 Description

D02NDF is a general purpose routine for integrating the initial value problem for a stiff system of explicit ordinary differential equations,

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

It is designed specifically for the case where the Jacobian $\dfrac{\partial g}{\partial y}$ is a sparse matrix.

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 D02NDF 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 linear algebra diagnostic routine D02NXF.

```
!     Declarations

      EXTERNAL FCN, 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 D02NDF(NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,    &
                  ATOL, ITOL, INFORM, FCN, YSAVE, NY2DIM, JAC,   &
                  WKJAC,NWKJAC, JACPVT, NJCPVT, MONITR, 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, D02NVF or D02NWF, must be called prior to the call of D02NDF. 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 D02NDF. There is also a routine, D02NZF, designed to permit you to change step size on a continuation call to D02NDF without restarting the integration process.

4 References

See the D02M–N Sub-chapter Introduction.

5 Arguments

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 differential equations to be solved during the integration.

Constraint: LDYSAV $\geq$ NEQ.

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

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).

Constraint: TOUT $\neq$ T.

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

On exit: 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*

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 entry: the relative local error tolerance.

Constraint: RTOL(i) ≥ 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 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 D02NDF 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: FCN – SUBROUTINE, supplied by the user. *External Procedure*

FCN must evaluate the derivative vector for the explicit ordinary differential equation system, defined by $y' = g(t, y)$.

The specification of FCN is:

```
SUBROUTINE FCN (NEQ, T, Y, F, IRES)
INTEGER            NEQ, IRES
REAL (KIND=nag_wp) T, Y(NEQ), F(NEQ)
```

1: NEQ – INTEGER *Input*

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

4: F(NEQ) – REAL (KIND=nag_wp) array *Output*

On exit: the value y'_i, given by $y'_i = g_i(t, y)$, for $i = 1, 2, \ldots,$ NEQ.

5: IRES – INTEGER *Input/Output*

On entry: IRES $= 1$.

On exit: you may set IRES as follows to indicate certain conditions in FCN to the integrator:

IRES $= 1$
: Indicates a normal return from FCN, that is IRES has not been altered by you and integration continues.

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 argument IMON $= -2$.

FCN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which D02NDF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

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

14: SDYSAV – INTEGER *Input*

On entry: the second dimension of the array YSAV as declared in the (sub)program from which D02NDF is called. An appropriate value for SDYSAV is described in the specification of the integrator setup routines 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 D02NDZ. (D02NDZ 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 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 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

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

but it is solved in the form

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

where r is the function defined by

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

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

$$\frac{\partial r_i}{\partial y_j} = 1 - (hd)\frac{\partial g_i}{\partial y_j}, \quad \text{if } i = j,$$

$$\frac{\partial r_i}{\partial y_j} = -(hd)\frac{\partial g_i}{\partial y_j}, \qquad \text{otherwise.}$$

The specification of JAC is:

```
SUBROUTINE JAC (NEQ, T, Y, H, D, J, PDJ)
INTEGER                 NEQ, J
REAL (KIND=nag_wp) T, Y(NEQ), H, D, PDJ(NEQ)
```

1: NEQ – INTEGER *Input*

On entry: the number of differential 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: H – REAL (KIND=nag_wp) *Input*

On entry: the current step size.

5: D – REAL (KIND=nag_wp) *Input*

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

6: J – INTEGER *Input*

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

7: PDJ(NEQ) – REAL (KIND=nag_wp) 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 either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which D02NDF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

16: WKJAC(NWKJAC) – REAL (KIND=nag_wp) array *Communication Array*

17: NWKJAC – INTEGER *Input*

On entry: the dimension of the array WKJAC as declared in the (sub)program from which D02NDF 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 D02NDF 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 of the linear algebra setup routine D02NUF. This value must be the 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, ACOR, IMON, INLN, HMIN, HMAX, NQU)

INTEGER            NEQ, LDYSAV, IMON, INLN, NQU
REAL (KIND=nag_wp) T, HLAST, HNEXT, Y(NEQ), YDOT(NEQ),               &
                   YSAV(LDYSAV,sdysav), R(NEQ), ACOR(NEQ,2), HMIN,   &
                   HMAX
```

where $sdysav$ is the numerical value of SDYSAV in the call of D02NDF.

1: NEQ – INTEGER *Input*

On entry: the number of differential equations being solved.

2: LDYSAV – INTEGER *Input*

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

3: T – REAL (KIND=nag_wp) *Input*

On entry: the current value of the independent variable.

4: HLAST – REAL (KIND=nag_wp) *Input*

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

5: HNEXT – REAL (KIND=nag_wp) *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) – REAL (KIND=nag_wp) 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) – REAL (KIND=nag_wp) array *Input*

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

8: YSAV(LDYSAV, $sdysav$) – REAL (KIND=nag_wp) array *Input*

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

9: R(NEQ) – REAL (KIND=nag_wp) array *Input*

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

10: ACOR(NEQ, 2) – REAL (KIND=nag_wp) 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 real function D02ZAF as follows:

```
IFAIL = 1
 ERRLOC = D02ZAF(NEQ, ACOR(1,2), ACOR(1,1), IFAIL)
 ! 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 FCN) 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 D02NDF.

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 – REAL (KIND=nag_wp) *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 – REAL (KIND=nag_wp) *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 either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02NDF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

21: 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 (e.g., see D02MVF). 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 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 D02MVF). TCRIT must be specified as under ITASK $= 4$.

Constraint: ITASK $= 1$, 2, 3, 4 or 5.

22: 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.

23: 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 $= 1$

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 $= 7$

FCN set its error flag (IRES $= 3$) continually despite repeated attempts by the integrator to avoid this.

IFAIL $= 8$

Not used for the integrator.

IFAIL $= 9$

A singular Jacobian $\dfrac{\partial r}{\partial y}$ has been encountered. This error exit is unlikely to be taken when solving explicit ordinary differential equations. 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 $= 11$

FCN 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 $\neq 2$ or 5.)

IFAIL $= 14$

The values of RTOL and ATOL are so small that D02NDF is unable to start the integration.

IFAIL $= 15$

The linear algebra setup routine D02NUF was not called prior to calling D02NDF.

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

D02NDF 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.

D02NDF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

D02NDF 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

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 indication of numerical stability.

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 $\dfrac{\partial g}{\partial y}$ has complex eigenvalues near the imaginary axis for some part of the integration, you should try the BLEND option (setup routine D02NWF).

10 Example

This example solves the well-known stiff Robertson problem

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

over the range $[0, 10.0]$ with initial conditions $a = 1.0$ and $b = c = 0.0$ using scalar error control (ITOL $= 1$). The solution is computed up to 10.0 by overshooting and interpolating (ITASK $= 1$) and the intermediate solution computed on an equispaced mesh through MONITR. The integration algorithm used is the BDF method (setup routine D02NVF) and a modified Newton method is also used. The use of the `N' (Numerical) and `S' (Structural) options are illustrated in turn for calculating the Jacobian.

10.1 Program Text

```
!   D02NDF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module d02ndfe_mod

!     D02NDF 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                             :: fcn, monitr
!     .. Parameters ..
      Integer, Parameter, Public         :: iset = 1, neq = 3
```

```
      Integer, Parameter, Public       :: nia = neq + 1
      Integer, Parameter, Public       :: nin = 5
      Integer, Parameter, Public       :: njcpvt = 20*neq + 100
      Integer, Parameter, Public       :: nout = 6
      Integer, Parameter, Public       :: nrw = 50 + 4*neq
      Integer, Parameter, Public       :: nwkjac = 4*neq + 100
      Integer, Parameter, Public       :: sdysav = 6
      Integer, Parameter, Public       :: ldysav = neq
    Contains
      Subroutine fcn(neq,t,y,f,ires)

!       .. 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
!       .. 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) :: f(neq)
        Real (Kind=nag_wp), Intent (In) :: y(neq)
!       .. Executable Statements ..
        f(1) = -alpha*y(1) + beta*y(2)*y(3)
        f(2) = alpha*y(1) - beta*y(2)*y(3) - gamma*y(2)*y(2)
        f(3) = gamma*y(2)*y(2)
        Return
      End Subroutine fcn
      Subroutine monitr(neq,ldysav,t,hlast,hnext,y,ydot,ysav,r,acor,imon,inln, &
        hmin,hmax,nqu)

!       .. Use Statements ..
        Use nag_library, Only: d02xkf
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: hlast, t
        Real (Kind=nag_wp), Intent (Inout) :: hmax, hmin, hnext
        Integer, Intent (Inout)        :: imon
        Integer, Intent (Out)          :: inln
        Integer, Intent (In)           :: ldysav, neq, nqu
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In) :: acor(neq,2), r(neq), ydot(neq),     &
                                           ysav(ldysav,*)
        Real (Kind=nag_wp), Intent (Inout) :: y(neq)
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: xout
        Integer                        :: i, ifail
!       .. Local Arrays ..
        Real (Kind=nag_wp), Allocatable :: z(:)
!       .. Executable Statements ..
        inln = 3

        If (imon==1) Then
          Allocate (z(neq))

!         Interpolate at multiples of 2.0 between t-hlast and t
          xout = 2.0E0_nag_wp
          Do While (xout<=t-hlast)
            xout = xout + 2.0E0_nag_wp
          End Do
loop:     Do While (t-hlast<xout .And. xout<=t)

!           C1 interpolation

            ifail = 1
            Call d02xkf(xout,z,neq,ysav,ldysav,sdysav,acor(1,2),neq,t,nqu,     &
              hlast,hnext,ifail)

            If (ifail/=0) Then
              imon = -2
              Exit loop
            End If
```

```
            Write (nout,99999) xout, (z(i),i=1,neq)
            xout = xout + 2.0E0_nag_wp

            If (xout>=10.0E0_nag_wp) Then
              Exit loop
            End If

          End Do loop

        End If

        Deallocate (z)
        Return

99999   Format (1X,F8.3,3(F13.5,2X))
      End Subroutine monitr
    End Module d02ndfe_mod

    Program d02ndfe

!     D02NDF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02ndf, d02ndz, d02nuf, d02nvf, d02nxf, d02nyf,   &
                             nag_wp, x04abf
      Use d02ndfe_mod, Only: fcn, iset, ldysav, monitr, neq, nia, nin, njcpvt, &
                             nout, nrw, nwkjac, sdysav
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)              :: eta, h, h0, hmax, hmin, hu, sens, t, &
                                         tcrit, tcur, tinit, tolsf, tout, u
      Integer                         :: i, icall, icase, ifail, igrow,       &
                                         imxer, isplit, isplt, itask, itol,   &
                                         itrace, liwreq, liwusd, lrwreq,      &
                                         lrwusd, maxord, maxstp, mxhnil,      &
                                         nblock, ngp, niter, nja, nje, nlu,   &
                                         nnz, nq, nqu, nre, nst, outchn
      Logical                         :: lblock, petzld
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: atol(:), rtol(:), rwork(:),          &
                                         wkjac(:), y(:), ydot(:), yinit(:),   &
                                         ysav(:,:)
      Real (Kind=nag_wp)              :: con(6)
      Integer, Allocatable            :: ia(:), ja(:), jacpvt(:)
      Integer                         :: inform(23)
      Logical, Allocatable            :: algequ(:)
!     .. Executable Statements ..
      Write (nout,*) 'D02NDF Example Program Results'
!     Skip heading in data file
      Read (nin,*)

!     Allocations based on number of differential equations (neq)
      Allocate (atol(neq),rtol(neq),rwork(nrw),wkjac(nwkjac),y(neq),           &
        yinit(neq),ydot(neq),ysav(ldysav,sdysav),ia(nia),jacpvt(njcpvt),       &
        algequ(neq))

      Read (nin,*) maxord, maxstp, mxhnil
      Read (nin,*) ia(1:nia)

      nja = ia(nia) - 1
      Allocate (ja(nja))
      Read (nin,*) ja(1:nja)

!     Read algorithmic parameters
      Read (nin,*) hmin, hmax, h0, tcrit
      Read (nin,*) eta, sens, u
      Read (nin,*) lblock, petzld
      Read (nin,*) tinit, tout
      Read (nin,*) itol, isplt
```

```
      Read (nin,*) yinit(1:neq)
      Read (nin,*) rtol(1), atol(1)

      outchn = nout
      Call x04abf(iset,outchn)

      Do icase = 1, 2
        t = tinit
        isplit = isplt
        y(1:neq) = yinit(1:neq)

!       In both cases: integrate to tout by overshooting (itask=1) using BDF
!       with Newton; use default con values, scalar tolerances and numerical
!       Jacobian; interpolate using MONITR and D02XKF.

        con(1:6) = 0.0_nag_wp
        itask = 1
        ifail = 0
        Call d02nvf(neq,sdysav,maxord,'Newton',petzld,con,tcrit,hmin,hmax,h0,  &
          maxstp,mxhnil,'Average-L2',rwork,ifail)
        Write (nout,*)

        Select Case (icase)
        Case (1)
!         No Jacobian Structure Supplied.
          ifail = 0
          Call d02nuf(neq,neq,'Numerical',nwkjac,ia,nia,ja,nja,jacpvt,njcpvt,  &
            sens,u,eta,lblock,isplit,rwork,ifail)
          Write (nout,*) '  Numerical Jacobian, structure not supplied'
        Case (2)
!         Jacobian Structure Supplied.
          ifail = 0
          Call d02nuf(neq,neq,'Structural',nwkjac,ia,nia,ja,nja,jacpvt,njcpvt, &
            sens,u,eta,lblock,isplit,rwork,ifail)
          Write (nout,*) '  Numerical Jacobian, structure supplied'
        End Select

        Write (nout,99988)(i,i=1,neq)
        Write (nout,99999) t, (y(i),i=1,neq)

!       Soft fail and error messages only
        itrace = 0

        ifail = -1
        Call d02ndf(neq,ldysav,t,tout,y,ydot,rwork,rtol,atol,itol,inform,fcn,  &
          ysav,sdysav,d02ndz,wkjac,nwkjac,jacpvt,njcpvt,monitr,itask,itrace,   &
          ifail)

        If (ifail==0) Then

          ifail = 0
          Call d02nyf(neq,neq,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==10) Then
          Write (nout,*)
          Write (nout,99998) 'Exit D02NDF with IFAIL = ', ifail, '  and T = ', &
            t

          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 D02NDF with IFAIL = ', ifail, '  and T = ', &
            t
        End If
      End Do

99999 Format (1X,F8.3,3(F13.5,2X))
99998 Format (1X,A,I5,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 ISPLIT ',I4)
99989 Format (1X,' Growth est ',I8,'  No. of blocks on diagonal ',I4)
99988 Format (/,1X,'    X ',3('         Y(',I1,')  '))
    End Program d02ndfe
```

10.2 Program Data

```
D02NDF Example Program Data
   5  200  5                        : maxord, maxstp, mxhnil
   1  3  6  9                       : ia
   1  2  1  2  3  1  2  3           : ja
   1.0E-10  10.0  0.0  0.0          : hmin, hmax, h0, tcrit
   1.0E-4  0.0  0.1                 : eta, sens, u
   .TRUE. .FALSE.                   : lblock, petzld
   0.0  10.0                        : t, tout
   1  0                             : itol, isplit
   1.0  0.0  0.0                    : y
   1.0E-4  1.0E-7                   : rtol, atol
```

10.3 Program Results

```
 D02NDF Example Program Results

   Numerical Jacobian, structure not supplied

     X          Y(1)           Y(2)           Y(3)
    0.000      1.00000        0.00000        0.00000
    2.000      0.94161        0.00003        0.05836
    4.000      0.90552        0.00002        0.09446
    6.000      0.87927        0.00002        0.12072
    8.000      0.85855        0.00002        0.14144
   10.000      0.84137        0.00002        0.15863

  HUSED =  0.90236E+00  HNEXT =  0.90236E+00  TCUR =  0.10769E+02
  NST =     55    NRE =    136    NJE =     16
  NQU =      4    NQ  =      4  NITER =     78
  Max err comp =    3


  NJCPVT (required  100  used      160)
  NWKJAC (required   29  used       78)
  No. of LU-decomps   16  No. of nonzeros        7
```

```
No. of FCN calls to form Jacobian    3  Try ISPLIT   73
Growth est     1108  No. of blocks on diagonal    1

 Numerical Jacobian, structure supplied

   X          Y(1)           Y(2)           Y(3)
 0.000      1.00000        0.00000        0.00000
 2.000      0.94161        0.00003        0.05836
 4.000      0.90551        0.00002        0.09446
 6.000      0.87926        0.00002        0.12072
 8.000      0.85854        0.00002        0.14144
10.000      0.84135        0.00002        0.15863

HUSED =  0.90178E+00  HNEXT =  0.90178E+00  TCUR =  0.10766E+02
NST =     55    NRE =    129    NJE =     16
NQU =      4    NQ  =      4  NITER =     78
Max err comp =    3


NJCPVT (required  106  used      160)
NWKJAC (required   31  used       77)
No. of LU-decomps   16  No. of nonzeros        8
No. of FCN calls to form Jacobian    3  Try ISPLIT   73
Growth est   277504  No. of blocks on diagonal    1
```