NAG Library Routine Document

D02MWF

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

D02MWF is a setup routine which must be called prior to the integrator D02NEF, if the DASSL implementation of Backward Differentiation Formulae (BDF) is to be used.

2 Specification

```
SUBROUTINE D02MWF (NEQ, MAXORD, JCEVAL, HMAX, H0, ITOL, ICOM, LICOM, COM,       &
                   LCOM, IFAIL)

INTEGER            NEQ, MAXORD, ITOL, ICOM(LICOM), LICOM, LCOM, IFAIL
REAL (KIND=nag_wp) HMAX, H0, COM(LCOM)
CHARACTER(1)       JCEVAL
```

3 Description

This integrator setup routine must be called before the first call to the integrator D02NEF. This setup routine D02MWF permits you to define options for the DASSL integrator, such as: whether the Jacobian is to be provided or is to be approximated numerically by the integrator; the initial and maximum step-sizes for the integration; whether relative and absolute tolerances are system wide or per system equation; and the maximum order of BDF method permitted.

4 References

None.

5 Parameters

1: NEQ – INTEGER *Input*

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

Constraint: NEQ ≥ 1.

2: MAXORD – INTEGER *Input*

On entry: the maximum order to be used for the BDF method. Orders up to 5th order are available; setting MAXORD > 5 means that the maximum order used will be 5.

Constraint: $1 \leq$ MAXORD.

3: JCEVAL – CHARACTER(1) *Input*

On entry: specifies the technique to be used to compute the Jacobian.

JCEVAL = 'N'
The Jacobian is to be evaluated numerically by the integrator.

JCEVAL = 'A'
You must supply a subroutine to evaluate the Jacobian on a call to the integrator.

Only the first character of the actual paramater JCEVAL is passed to D02MWF; hence it is permissible for the actual argument to be more descriptive, e.g., ‘Numerical’ or ‘Analytical’, on a call to D02MWF.

Constraint: JCEVAL = 'N' or 'A'.

4: HMAX – REAL (KIND=nag_wp) *Input*

On entry: the maximum absolute step size to be allowed. Set HMAX = 0.0 if this option is not required.

Constraint: HMAX $\geq$ 0.0.

5: H0 – REAL (KIND=nag_wp) *Input*

On entry: the step size to be attempted on the first step. Set H0 = 0.0 if the initial step size is calculated internally.

6: ITOL – INTEGER *Input*

On entry: a value to indicate the form of the local error test.

ITOL = 0
RTOL and ATOL are single element vectors.

ITOL = 1
RTOL and ATOL are vectors. This should be chosen if you want to apply different tolerances to each equation in the system.

See D02NEF.

Note: The tolerances must either both be single element vectors or both be vectors of length NEQ.

Constraint: ITOL = 0 or 1.

7: ICOM(LICOM) – INTEGER array *Communication Array*

On exit: used to communicate details of the task to be carried out to the integration routine D02NEF.

8: LICOM – INTEGER *Input*

On entry: the dimension of the array ICOM as declared in the (sub)program from which D02MWF is called.

Constraint: LICOM $\geq$ NEQ + 50.

9: COM(LCOM) – REAL (KIND=nag_wp) array *Communication Array*

On exit: used to communicate problem parameters to the integration routine D02NEF. This must be the same communication array as the array COM supplied to D02NEF. In particular, the values of HMAX and H0 are contained in COM.

10: LCOM – INTEGER *Input*

On entry: the dimension of the array COM as declared in the (sub)program from which D02MWF is called.

Constraints:

the array COM must be large enough for the requirements of D02NEF. That is:

if the system Jacobian is dense, LCOM $\geq 40 + (\mathrm{MAXORD} + 4) \times \mathrm{NEQ} + \mathrm{NEQ}^2$;
if the system Jacobian is banded,
LCOM $\geq 40 + (\mathrm{MAXORD} + 4) \times \mathrm{NEQ} + (2 \times \mathrm{ML} + \mathrm{MU} + 1) \times \mathrm{NEQ} + 2 \times (\mathrm{NEQ}/(\mathrm{ML} + \mathrm{MU} + 1) + 1)$.

Here ML and MU are the lower and upper bandwidths respectively that are to be specified in a subsequent call to D02NPF.

11: 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.

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, if you are not familiar with this parameter, the recommended value is 0. **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, NEQ < 1.

IFAIL $= 2$

On entry, MAXORD < 1,
or MAXORD > 5.

IFAIL $= 3$

On entry, JCEVAL $\neq$ 'N' or 'A'.

IFAIL $= 4$

On entry, HMAX < 0.0.

IFAIL $= 6$

On entry, ITOL $\neq 0$ or 1.

IFAIL $= 8$

On entry, LICOM $<$ NEQ $+ 50$.

7 Accuracy

Not applicable.

8 Further Comments

None.

9 Example

This example solves the plane pendulum problem, defined by the following equations:

$$\begin{aligned} x' &= u \\ y' &= v \\ u' &= -\lambda x \\ v' &= -\lambda y - 1 \\ x^2 + y^2 &= 1. \end{aligned}$$

Differentiating the algebraic constraint once, a new algebraic constraint is obtained

$$xu + yv = 0.$$

Differentiating the algebraic constraint one more time, substituting for x', y', u', v' and using $x^2 + y^2 - 1 = 0$, the corresponding DAE system includes the differential equations and the algebraic equation in λ:

$$u^2 + v^2 - \lambda - y = 0.$$

We solve the reformulated DAE system

$$\begin{aligned} y_1' &= y_3 \\ y_2' &= y_4 \\ y_3' &= -y_5 \times y_1 \\ y_4' &= -y_5 \times y_2 - 1 \\ y_3^2 + y_4^2 - y_5 - y_2 &= 0. \end{aligned}$$

For our experiments, we take consistent initial values

$$y_1(0) = 1,\ y_2(0) = 0,\ y_3(0) = 0,\ y_4(0) = 1 \text{ and } y_5(0) = 1$$

at $t = 0$.

9.1 Program Text

```
!   D02MWF Example Program Text
!   Mark 24 Release. NAG Copyright 2012.

    Module d02mwfe_mod

!     D02MWF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter                   :: iset = 1, neq = 5, nin = 5,      &
                                              nout = 6
      Integer, Parameter                   :: licom = 50 + neq
    Contains
      Subroutine res(neq,t,y,ydot,r,ires,iuser,ruser)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: t
        Integer, Intent (Inout)              :: ires
        Integer, Intent (In)                 :: neq
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: r(neq)
        Real (Kind=nag_wp), Intent (Inout)   :: ruser(*)
        Real (Kind=nag_wp), Intent (In)      :: y(neq), ydot(neq)
        Integer, Intent (Inout)              :: iuser(*)
!       .. Executable Statements ..
        r(1) = y(3) - ydot(1)
        r(2) = y(4) - ydot(2)
        r(3) = -y(5)*y(1) - ydot(3)
        r(4) = -y(5)*y(2) - 1.0_nag_wp - ydot(4)
        r(5) = y(3)**2 + y(4)**2 - y(5) - y(2)
        Return
      End Subroutine res

      Subroutine jac(neq,t,y,ydot,pd,cj,iuser,ruser)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: cj, t
        Integer, Intent (In)                 :: neq
!       .. Array Arguments ..
```

```
        Real (Kind=nag_wp), Intent (Inout)   :: pd(*), ruser(*)
        Real (Kind=nag_wp), Intent (In)      :: y(neq), ydot(neq)
        Integer, Intent (Inout)              :: iuser(*)
!       .. Executable Statements ..
        pd(1) = -cj
        pd(3) = -y(5)
        pd(7) = -cj
        pd(9) = -y(5)
        pd(10) = -1.0_nag_wp
        pd(11) = 1.0_nag_wp
        pd(13) = -cj
        pd(15) = 2.0_nag_wp*y(3)
        pd(17) = 1.0_nag_wp
        pd(19) = -cj
        pd(20) = 2.0_nag_wp*y(4)
        pd(23) = -y(1)
        pd(24) = -y(2)
        pd(25) = -1.0_nag_wp
        Return
      End Subroutine jac
    End Module d02mwfe_mod

    Program d02mwfe

!     D02MWF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02mwf, d02nef, nag_wp, x04abf
      Use d02mwfe_mod, Only: iset, jac, licom, neq, nin, nout, res
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                  :: g1, g2, h0, hmax, t, tout
      Integer                             :: i, ifail, ijac, itask, itol,     &
                                             lcom, maxord, nadv
      Character (8)                       :: jceval
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable     :: atol(:), com(:), rtol(:), y(:),  &
                                             ydot(:)
      Real (Kind=nag_wp)                  :: ruser(1)
      Integer, Allocatable                :: icom(:)
      Integer                             :: iuser(1)
!     .. Executable Statements ..
      Write (nout,*) 'D02MWF Example Program Results'
      Write (nout,*)
!     Skip heading in data file
      Read (nin,*)
!     neq: number of differential-algebraic equations
      Read (nin,*) maxord
      lcom = 40 + (maxord+4)*neq + neq**2
      Allocate (atol(neq),com(lcom),rtol(neq),y(neq),ydot(neq),icom(licom))
      nadv = nout
      Call x04abf(iset,nadv)
      Read (nin,*) t, tout
      Read (nin,*) itol, itask
      Read (nin,*) rtol(1:neq)
      Read (nin,*) atol(1:neq)
!     Set initial values
      Read (nin,*) y(1:neq)
      Read (nin,*) hmax, h0
      Read (nin,*) ijac
      If (ijac==1) Then
        jceval = 'Analytic'
      Else
        jceval = 'Numeric'
      End If

!     ifail: behaviour on error exit
!             =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
      ifail = 0
      Call d02mwf(neq,maxord,jceval,hmax,h0,itol,icom,licom,com,lcom,ifail)
      Write (nout,99995)(i,i=1,neq)
      Write (nout,99998) t, y(1:neq)
      ydot(1:neq) = 0.0_nag_wp

loop: Do
        Call d02nef(neq,t,tout,y,ydot,rtol,atol,itask,res,jac,icom,com,lcom, &
          iuser,ruser,ifail)

        Write (nout,99998) t, y(1:neq)
        Write (nout,99999) itask
        If ((itask>=0) .And. (itask<=3)) Then
          If (t>=tout) Then
            g1 = y(1)**2 + y(2)**2 - 1.0_nag_wp
            g2 = y(1)*y(3) + y(2)*y(4)
            Write (nout,99997) g1
            Write (nout,99996) g2
            Exit loop
          End If
        Else
          Exit loop
        End If
      End Do loop

99999 Format (/' D02NEF returned with ITASK = ',I4/)
99998 Format (1X,F7.4,2X,5(F11.6))
99997 Format (1X,'The position-level constraint G1 = ',E12.4)
99996 Format (1X,'The velocity-level constraint G2 = ',E12.4)
99995 Format (/1X,'  t ',3X,5('       y(',I1,')'))
    End Program d02mwfe
```

9.2 Program Data

```
D02MWF Example Program Data
   5                                      : maxord
   0.0  1.0                               : t, tout
   1 0                                    : itol, itask
   1.0E-8 1.0E-8 1.0E-8 1.0E-8 1.0E-8     : rtol
   1.0E-8 1.0E-8 1.0E-8 1.0E-8 1.0E-8     : atol
   1.0  0.0  0.0  1.0  1.0                : y
   0.0  0.0                               : hmax, h0
   1                                      : ijac
```

9.3 Program Results

```
 D02MWF Example Program Results


   t          y(1)       y(2)       y(3)       y(4)       y(5)
  0.0000     1.000000   0.000000   0.000000   1.000000   1.000000
  1.0000     0.867349   0.497701  -0.033748   0.058813  -0.493103

 D02NEF returned with ITASK =    3

 The position-level constraint G1 =  -0.8580E-08
 The velocity-level constraint G2 =  -0.3005E-07
```