NAG Library Routine Document

D02QFF

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

D02QFF is a routine for integrating a non-stiff system of first-order ordinary differential equations using a variable-order variable-step Adams method. A root-finding facility is provided.

2 Specification

```
SUBROUTINE D02QFF (FCN, NEQF, T, Y, TOUT, G, NEQG, ROOT, RWORK, LRWORK,         &
                   IWORK, LIWORK, IFAIL)

INTEGER             NEQF, NEQG, LRWORK, IWORK(LIWORK), LIWORK, IFAIL
REAL (KIND=nag_wp) T, Y(NEQF), TOUT, G, RWORK(LRWORK)
LOGICAL             ROOT
EXTERNAL            FCN, G
```

3 Description

Given the initial values $x, y_1, y_2, \ldots, y_{\mathrm{NEQF}}$ D02QFF integrates a non-stiff system of first-order differential equations of the type

$$y_i' = f_i\big(x, y_1, y_2, \ldots, y_{\mathrm{NEQF}}\big), \qquad i = 1, 2, \ldots, \mathrm{NEQF},$$

from $x = \mathrm{T}$ to $x = \mathrm{TOUT}$ using a variable-order variable-step Adams method. The system is defined by FCN, which evaluates f_i in terms of x and $y_1, y_2, \ldots, y_{\mathrm{NEQF}}$, and $y_1, y_2, \ldots, y_{\mathrm{NEQF}}$ are supplied at $x = \mathrm{T}$. The routine is capable of finding roots (values of x) of prescribed event functions of the form

$$g_j\big(x, y, y'\big) = 0, \qquad j = 1, 2, \ldots, \mathrm{NEQG}.$$

(See D02QWF for the specification of NEQG.)

Each g_j is considered to be independent of the others so that roots are sought of each g_j individually. The root reported by the routine will be the first root encountered by any g_j. Two techniques for determining the presence of a root in an integration step are available: the sophisticated method described in Watts (1985) and a simplified method whereby sign changes in each g_j are looked for at the ends of each integration step. The event functions are defined by G supplied by you which evaluates g_j in terms of $x, y_1, \ldots, y_{\mathrm{NEQF}}$ and $y_1', \ldots, y_{\mathrm{NEQF}}'$. In one-step mode the routine returns an approximation to the solution at each integration point. In interval mode this value is returned at the end of the integration range. If a root is detected this approximation is given at the root. You select the mode of operation, the error control, the root-finding technique and various optional inputs by a prior call to the setup routine D02QWF.

For a description of the practical implementation of an Adams formula see Shampine and Gordon (1975) and Shampine and Watts (1979).

4 References

Shampine L F and Gordon M K (1975) *Computer Solution of Ordinary Differential Equations – The Initial Value Problem* W H Freeman & Co., San Francisco

Shampine L F and Watts H A (1979) DEPAC – design of a user oriented package of ODE solvers *Report SAND79-2374* Sandia National Laboratory

Watts H A (1985) RDEAM – An Adams ODE code with root solving capability *Report SAND85-1595* Sandia National Laboratory

5 Parameters

1: FCN – SUBROUTINE, supplied by the user. *External Procedure*

FCN must evaluate the functions f_i (that is the first derivatives y_i') for given values of its arguments x, $y_1, y_2, \ldots, y_{\mathrm{NEQF}}$.

The specification of FCN is:

```
SUBROUTINE FCN (NEQF, X, Y, F)

INTEGER            NEQF
REAL (KIND=nag_wp) X, Y(NEQF), F(NEQF)
```

1: NEQF – INTEGER *Input*

On entry: the number of differential equations.

2: X – REAL (KIND=nag_wp) *Input*

On entry: the current value of the argument x.

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

On entry: y_i, for $i = 1, 2, \ldots,$ NEQF, the current value of the argument.

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

On exit: the value of f_i, for $i = 1, 2, \ldots,$ NEQF.

FCN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02QFF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: NEQF – INTEGER *Input*

On entry: the number of first-order ordinary differential equations to be solved by D02QFF. It must contain the same value as the parameter NEQF used in a prior call to D02QWF.

Constraint: NEQF ≥ 1.

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

On entry: after a call to D02QWF with STATEF = 'S' (i.e., an initial entry), T must be set to the initial value of the independent variable x.

On exit: the value of x at which y has been computed. This may be an intermediate output point, a root, TOUT or a point at which an error has occurred. If the integration is to be continued, possibly with a new value for TOUT, T must not be changed.

4: Y(NEQF) – REAL (KIND=nag_wp) array *Input/Output*

On entry: the initial values of the solution $y_1, y_2, \ldots, y_{\mathrm{NEQF}}$.

On exit: the computed values of the solution at the exit value of T. If the integration is to be continued, possibly with a new value for TOUT, these values must not be changed.

5: TOUT – REAL (KIND=nag_wp) *Input*

On entry: the next value of x at which a computed solution is required. For the initial T, the input value of TOUT is used to determine the direction of integration. Integration is permitted in either direction. If TOUT = T on exit, TOUT must be reset beyond T **in the direction of integration**, before any continuation call.

6: G – REAL (KIND=nag_wp) FUNCTION, supplied by the user. *External Procedure*

G must evaluate a given component of $g(x, y, y')$ at a specified point.

If root-finding is not required the actual argument for G must be the dummy routine D02QFZ. (D02QFZ is included in the NAG Library.)

The specification of G is:

```
FUNCTION G (NEQF, X, Y, YP, K)
REAL (KIND=nag_wp) G
INTEGER            NEQF, K
REAL (KIND=nag_wp) X, Y(NEQF), YP(NEQF)
```

1: NEQF – INTEGER *Input*

On entry: the number of differential equations being solved.

2: X – REAL (KIND=nag_wp) *Input*

On entry: the current value of the independent variable.

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

On entry: the current values of the dependent variables.

4: YP(NEQF) – REAL (KIND=nag_wp) array *Input*

On entry: the current values of the derivatives of the dependent variables.

5: K – INTEGER *Input*

On entry: the component of g which must be evaluated.

G must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02QFF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: NEQG – INTEGER *Input*

On entry: the number of event functions which you are defining for root-finding. If root-finding is not required the value for NEQG must be ≤ 0. Otherwise it must be the same parameter NEQG used in the prior call to D02QWF.

8: ROOT – LOGICAL *Output*

On exit: if root-finding was required (NEQG > 0 on entry), then ROOT specifies whether or not the output value of the parameter T is a root of one of the event functions. If ROOT = .FALSE., then no root was detected, whereas ROOT = .TRUE. indicates a root and you should make a call to D02QYF for further information.

If root-finding was not required (NEQG $= 0$ on entry), then on exit ROOT = .FALSE..

9: RWORK(LRWORK) – REAL (KIND=nag_wp) array *Communication Array*

This **must** be the same parameter RWORK as supplied to D02QWF. It is used to pass information from D02QWF to D02QFF, and from D02QFF to D02QXF, D02QYF and D02QZF. Therefore the contents of this array **must not** be changed before the call to D02QFF or calling any of the routines D02QXF, D02QYF and D02QZF.

10: LRWORK – INTEGER *Input*

On entry: the dimension of the array RWORK as declared in the (sub)program from which D02QFF is called.

This must be the same parameter LRWORK as supplied to D02QWF.

11: IWORK(LIWORK) – INTEGER array *Communication Array*

This **must** be the same parameter IWORK as supplied to D02QWF. It is used to pass information from D02QWF to D02QFF, and from D02QFF to D02QXF, D02QYF and D02QZF. Therefore the contents of this array **must not** be changed before the call to D02QFF or calling any of the routines D02QXF, D02QYF and D02QZF.

12: LIWORK – INTEGER *Input*

On entry: the dimension of the array IWORK as declared in the (sub)program from which D02QFF is called.

This must be the same parameter LIWORK as supplied to D02QWF.

13: 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, 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.**

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 D02QWF has not been called before the call to the integrator.

This error may be caused by overwriting elements of RWORK and IWORK.

IFAIL $= 2$

The maximum number of steps has been attempted (at a cost of about 2 calls to FCN per step). (See parameter MAXSTP in D02QWF.) If integration is to be continued then you need only reset IFAIL and call the routine again and a further MAXSTP steps will be attempted.

IFAIL $= 3$

The step size needed to satisfy the error requirements is too small for the ***machine precision*** being used. (See parameter TOLFAC in D02QXF.)

IFAIL $= 4$

Some error weight w_i became zero during the integration (see parameters VECTOL, RTOL and ATOL in D02QWF.) Pure relative error control (ATOL $= 0.0$) was requested on a variable (the ith) which has now become zero. (See parameter BADCMP in D02QXF.) The integration was successful as far as T.

IFAIL = 5

The problem appears to be stiff (see the D02 Chapter Introduction for a discussion of the term 'stiff'). Although it is inefficient to use this integrator to solve stiff problems, integration may be continued by resetting IFAIL and calling the routine again.

IFAIL = 6

A change in sign of an event function has been detected but the root-finding process appears to have converged to a singular point T rather than a root. Integration may be continued by resetting IFAIL and calling the routine again.

IFAIL = 7

The code has detected two successive error exits at the current value of T and cannot proceed. Check all input variables.

7 Accuracy

The accuracy of integration is determined by the parameters VECTOL, RTOL and ATOL in a prior call to D02QWF. Note that only the local error at each step is controlled by these parameters. The error estimates obtained are not strict bounds but are usually reliable over one step. Over a number of steps the overall error may accumulate in various ways, depending on the properties of the differential equation system. The code is designed so that a reduction in the tolerances should lead to an approximately proportional reduction in the error. You are strongly recommended to call D02QFF with more than one set of tolerances and to compare the results obtained to estimate their accuracy.

The accuracy obtained depends on the type of error test used. If the solution oscillates around zero a relative error test should be avoided, whereas if the solution is exponentially increasing an absolute error test should not be used. If different accuracies are required for different components of the solution then a component-wise error test should be used. For a description of the error test see the specifications of the parameters VECTOL, ATOL and RTOL in the routine document for D02QWF.

The accuracy of any roots located will depend on the accuracy of integration and may also be restricted by the numerical properties of $g(x, y, y')$. When evaluating g you should try to write the code so that unnecessary cancellation errors will be avoided.

8 Further Comments

If D02QFF fails with IFAIL = 3 then the combination of ATOL and RTOL may be so small that a solution cannot be obtained, in which case the routine should be called again with larger values for RTOL and/or ATOL (see D02QWF). If the accuracy requested is really needed then you should consider whether there is a more fundamental difficulty. For example:

(a) in the region of a singularity the solution components will usually be of a large magnitude. D02QFF could be used in one-step mode to monitor the size of the solution with the aim of trapping the solution before the singularity. In any case numerical integration cannot be continued through a singularity, and analytical treatment may be necessary;

(b) for 'stiff' equations, where the solution contains rapidly decaying components, the routine will require a very small step size to preserve stability. This will usually be exhibited by excessive computing time and sometimes an error exit with IFAIL = 3, but usually an error exit with IFAIL = 2 or 5. The Adams methods are not efficient in such cases and you should consider using a routine from the sub-chapter D02M–N. A high proportion of failed steps (see parameter NFAIL) may indicate stiffness but there may be other reasons for this phenomenon.

D02QFF can be used for producing results at short intervals (for example, for graph plotting); you should set CRIT = .TRUE. and TCRIT to the last output point required in a prior call to D02QWF and then set TOUT appropriately for each output point in turn in the call to D02QFF.

9 Example

This example solves the equation

$$y'' = -y, \qquad y(0) = 0, \qquad y'(0) = 1$$

reposed as

$$y_1' = y_2$$

$$y_2' = -y_1$$

over the range $[0, 10.0]$ with initial conditions $y_1 = 0.0$ and $y_2 = 1.0$ using vector error control (VECTOL = .TRUE.) and computation of the solution at TOUT = 10.0 with TCRIT = 10.0 (CRIT = .TRUE.). Also, we use D02QFF to locate the positions where $y_1 = 0.0$ or where the first component has a turning-point, that is $y_1' = 0.0$.

9.1 Program Text

```
!   D02QFF Example Program Text
!   Mark 24 Release. NAG Copyright 2012.

    Module d02qffe_mod

!     D02QFF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter                   :: neqf = 2, neqg = 2, nin = 5,     &
                                              nout = 6
      Integer, Parameter                   :: latol = neqf
      Integer, Parameter                   :: liwork = 21 + 4*neqg
      Integer, Parameter                   :: lrtol = neqf
      Integer, Parameter                   :: lrwork = 23 + 23*neqf + 14*neqg
    Contains
      Subroutine fcn(neqf,x,y,f)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: x
        Integer, Intent (In)                 :: neqf
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: f(neqf)
        Real (Kind=nag_wp), Intent (In)      :: y(neqf)
!       .. Executable Statements ..
        f(1) = y(2)
        f(2) = -y(1)
        Return
      End Subroutine fcn

      Function g(neqf,x,y,yp,k)

!       .. Function Return Value ..
        Real (Kind=nag_wp)                   :: g
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: x
        Integer, Intent (In)                 :: k, neqf
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: y(neqf), yp(neqf)
!       .. Executable Statements ..
        If (k==1) Then
          g = yp(1)
        Else
          g = y(1)
        End If
        Return
```

```
      End Function g
    End Module d02qffe_mod

    Program d02qffe

!     D02QFF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02qff, d02qwf, d02qxf, d02qyf, nag_wp
      Use d02qffe_mod, Only: fcn, g, latol, liwork, lrtol, lrwork, neqf, neqg, &
                             nin, nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                  :: hlast, hmax, hnext, t, tcrit,    &
                                             tcurr, tolfac, tout, tstart
      Integer                             :: badcmp, i, ifail, index, maxstp, &
                                             nfail, nsucc, odlast, odnext, type
      Logical                             :: alterg, crit, onestp, root,      &
                                             sophst, vectol
      Character (1)                       :: statef
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable     :: atol(:), resids(:), rtol(:),     &
                                             rwork(:), y(:), yp(:)
      Integer, Allocatable                :: events(:), iwork(:)
!     .. Executable Statements ..
      Write (nout,*) 'D02QFF Example Program Results'
!     Skip heading in data file
      Read (nin,*)
      Allocate (atol(latol),resids(neqg),rtol(lrtol),rwork(lrwork),y(neqf), &
        yp(neqf),events(neqg),iwork(liwork))
      Read (nin,*) hmax, tstart, tcrit
      Read (nin,*) statef
      Read (nin,*) vectol, onestp, crit, sophst
      Read (nin,*) maxstp
      Read (nin,*) rtol(1:neqf)
      Read (nin,*) atol(1:neqf)

!     Initialize
      ifail = 0
      Call d02qwf(statef,neqf,vectol,atol,latol,rtol,lrtol,onestp,crit,tcrit, &
        hmax,maxstp,neqg,alterg,sophst,rwork,lrwork,iwork,liwork,ifail)

      t = tstart
      tout = tcrit
      Read (nin,*) y(1:neqf)

!      Cycle through roots and print info when encountered.
findr: Do
        ifail = -1
        Call d02qff(fcn,neqf,t,y,tout,g,neqg,root,rwork,lrwork,iwork,liwork, &
          ifail)

        If (ifail/=0) Exit findr

        ifail = 0
        Call d02qxf(neqf,yp,tcurr,hlast,hnext,odlast,odnext,nsucc,nfail, &
          tolfac,badcmp,rwork,lrwork,iwork,liwork,ifail)

        If (.Not. root) Exit findr

        ifail = 0
        Call d02qyf(neqg,index,type,events,resids,rwork,lrwork,iwork,liwork, &
          ifail)

        Write (nout,99999) t
        Write (nout,99998) index, type, resids(index)
        Write (nout,99997) y(1), yp(1)

        Do i = 1, neqg
          If (i/=index) Then
```

```
            If (events(i)/=0) Then
              Write (nout,99996) i, events(i), resids(i)
            End If
          End If
        End Do

        If (tcurr>=tout) Exit findr

      End Do findr

99999 Format (/1X,'Root at ',1P,E13.5)
99998 Format (1X,'for event equation ',I2,' with type',I3,' and residual ',1P, &
        E13.5)
99997 Format (1X,' Y(1) = ',1P,E13.5,'   Y''(1) = ',1P,E13.5)
99996 Format (1X,'and also for event equation ',I2,' with type',I3, &
        ' and residual ',1P,E13.5)
    End Program d02qffe
```

9.2 Program Data

```
D02QFF Example Program Data
   0.0  0.0  10.0                          : hmax, tstart, tcrit
   S                                       : statef
   .TRUE. .FALSE. .TRUE. .TRUE.            : vectol, onestp, crit, sophst
   0                                       : maxstp
   1.0E-4  1.0E-4                          : rtol
   1.0E-6  1.0E-6                          : atol
   0.0  1.0                                : y
```

9.3 Program Results

```
 D02QFF Example Program Results

 Root at   0.00000E+00
 for event equation  2 with type  1 and residual   0.00000E+00
  Y(1) =   0.00000E+00   Y'(1) =   1.00000E+00

 Root at   1.57076E+00
 for event equation  1 with type  1 and residual  -5.20417E-17
  Y(1) =   1.00003E+00   Y'(1) =  -5.20417E-17

 Root at   3.14151E+00
 for event equation  2 with type  1 and residual  -1.27676E-15
  Y(1) =  -1.27676E-15   Y'(1) =  -1.00012E+00

 Root at   4.71228E+00
 for event equation  1 with type  1 and residual   1.67921E-15
  Y(1) =  -1.00010E+00   Y'(1) =   1.67921E-15

 Root at   6.28306E+00
 for event equation  2 with type  1 and residual   2.65066E-15
  Y(1) =   2.65066E-15   Y'(1) =   9.99979E-01

 Root at   7.85379E+00
 for event equation  1 with type  4 and residual  -7.63278E-17
  Y(1) =   9.99970E-01   Y'(1) =  -7.63278E-17

 Root at   9.42469E+00
 for event equation  2 with type  4 and residual  -6.86950E-16
  Y(1) =  -6.86950E-16   Y'(1) =  -9.99854E-01
```