NAG Library Routine Document

D02QGF

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

D02QGF is a reverse communication 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 D02QGF (NEQF, T, Y, TOUT, NEQG, ROOT, IREVCM, TRVCM, YRVCM,      &
                   YPRVCM, GRVCM, KGRVCM, RWORK, LRWORK, IWORK, LIWORK,     &
                   IFAIL)
INTEGER            NEQF, NEQG, IREVCM, YRVCM, YPRVCM, KGRVCM, LRWORK,       &
                   IWORK(LIWORK), LIWORK, IFAIL
REAL (KIND=nag_wp) T, Y(NEQF), TOUT, TRVCM, GRVCM, RWORK(LRWORK)
LOGICAL            ROOT
```

3 Description

Given the initial values $x, y_1, y_2, \ldots, y_{\mathrm{NEQF}}$ D02QGF integrates a non-stiff system of first-order differential equations of the type

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

from $x = \mathrm{T}$ to $x = \mathrm{TOUT}$ using a variable-order variable-step Adams' method. You define the system by reverse communication, evaluating 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 $x = \mathrm{T}$ by D02QGF. The routine is capable of finding roots (values of x) of prescribed event functions of the form

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

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. You also define each g_j by reverse communication. 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).

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 Arguments

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

1: NEQF – INTEGER *Input*

On initial entry: the number of first-order ordinary differential equations to be solved by D02QGF. It must contain the same value as the argument NEQF used in the prior call to D02QWF.

Constraint: NEQF ≥ 1.

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

On initial entry: that is after a call to D02QWF with STATEF $=$ 'S', T must be set to the initial value of the independent variable x.

On final 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.

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

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

On final 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.

4: TOUT – REAL (KIND=nag_wp) *Input*

On initial 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.

5: NEQG – INTEGER *Input*

On initial 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 value as the argument NEQG used in the prior call to D02QWF.

6: ROOT – LOGICAL *Output*

On final exit: if root-finding was required (NEQG > 0 on entry), then ROOT specifies whether or not the output value of the argument 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 ROOT $=$ .FALSE..

7: IREVCM – INTEGER *Input/Output*

On initial entry: must have the value 0.

On intermediate exit: specifies what action you must take before re-entering D02QGF **with** IREVCM **unchanged**.

IREVCM $= 1$, 2, 3, 4, 5, 6 or 7

Indicates that you must supply $y' = f(x,y)$, where x is given by TRVCM and y_i is returned in Y(i), for $i = 1, 2, \ldots, \mathrm{NEQF}$ when YRVCM $= 0$ and RWORK(YRVCM $+ i - 1$), for $i = 1, 2, \ldots, \mathrm{NEQF}$ when YRVCM $\neq 0$. y'_i should be placed in location RWORK(YPRVCM $+ i - 1$), for $i = 1, 2, \ldots, \mathrm{NEQF}$.

IREVCM $= 8$

Indicates that the current step was not successful due to error test failure. The only information supplied to you on this return is the current value of the independent variable T, as given by TRVCM. No values must be changed before re-entering D02QGF. This facility enables you to determine the number of unsuccessful steps.

IREVCM $= 9$, 10, 11 or 12

Indicates that you must supply $g_k(x, y, y')$, where k is given by KGRVCM, x is given by TRVCM, y_i is given by $\mathrm{Y}(i)$ and y'_i is given by RWORK$(\mathrm{YPRVCM} - 1 + i)$. The result g_k should be placed in the variable GRVCM.

On final exit: has the value 0, which indicates that an output point or root has been reached or an error has occurred (see IFAIL).

8: TRVCM – REAL (KIND=nag_wp) *Output*

On intermediate exit: the current value of the independent variable.

9: YRVCM – INTEGER *Output*

On intermediate exit: with IREVCM $= 1$, 2, 3, 4, 5, 6, 7, 9, 10, 11 or 12, YRVCM specifies the locations of the dependent variables y for use in evaluating the differential system or the event functions.

YRVCM $= 0$

y_i is given by $\mathrm{Y}(i)$, for $i = 1, 2, \ldots, \mathrm{NEQF}$.

YRVCM $\neq 0$

y_i is given by RWORK$(\mathrm{YRVCM} + i - 1)$, for $i = 1, 2, \ldots, \mathrm{NEQF}$.

10: YPRVCM – INTEGER *Output*

On intermediate exit: with IREVCM $= 1$, 2, 3, 4, 5, 6 or 7, YPRVCM specifies the positions in RWORK at which you should place the derivatives y'. y'_i should be placed in location RWORK$(\mathrm{YPRVCM} + i - 1)$, for $i = 1, 2, \ldots, \mathrm{NEQF}$.

With IREVCM $= 9$, 10, 11 or 12, YPRVCM specifies the locations of the derivatives y' for use in evaluating the event functions. y'_i is given by RWORK$(\mathrm{YPRVCM} + i - 1)$, for $i = 1, 2, \ldots, \mathrm{NEQF}$.

11: GRVCM – REAL (KIND=nag_wp) *Input*

On initial entry: need not be set.

On intermediate re-entry: with IREVCM $= 9$, 10, 11 or 12, GRVCM must contain the value of $g_k(x, y, y')$, where k is given by KGRVCM.

12: KGRVCM – INTEGER *Input/Output*

On intermediate re-entry: with IREVCM $= 9$, 10, 11 or 12, KGRVCM must remain unchanged from a previous call to D02QGF.

On intermediate exit: with IREVCM $= 9$, 10, 11 or 12, KGRVCM specifies which event function $g_k(x, y, y')$ you must evaluate.

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

This **must** be the same argument RWORK as supplied to D02QWF. It is used to pass information from D02QWF to D02QGF, and from D02QGF to D02QXF, D02QYF and D02QZF. Therefore the contents of this array **must not** be changed before the call to D02QGF or calling any of the routines D02QXF, D02QYF and D02QZF.

14: LRWORK – INTEGER *Input*

On initial entry: the dimension of the array RWORK as declared in the (sub)program from which D02QGF is called.

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

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

This **must** be the same argument IWORK as supplied to D02QWF. It is used to pass information from D02QWF to D02QGF, and from D02QGF to D02QXF, D02QYF and D02QZF. Therefore the contents of this array **must not** be changed before the call to D02QGF or calling any of the routines D02QXF, D02QYF and D02QZF.

16: LIWORK – INTEGER *Input*

On initial entry: the dimension of the array IWORK as declared in the (sub)program from which D02QGF is called.

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

17: IFAIL – INTEGER *Input/Output*

On initial 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 final 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 derivative evaluations per step). (See argument 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 argument TOLFAC in D02QXF.)

IFAIL $= 4$

Some error weight w_i became zero during the integration (see arguments 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 argument 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.

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 integration is determined by the arguments VECTOL, RTOL and ATOL in a prior call to D02QWF. Note that only the local error at each step is controlled by these arguments. 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 D02QGF 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 arguments VECTOL, RTOL and ATOL 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 Parallelism and Performance

D02QGF 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.

D02QGF is not threaded in any implementation.

9 Further Comments

If D02QGF 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. D02QGF 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 argument NFAIL in D02QXF) may indicate stiffness but there may be other reasons for this phenomenon.

D02QGF 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 D02QGF.

10 Example

This example solves the following system (for a projectile)

$$\begin{aligned} y' &= \tan\phi \\ v' &= \frac{-0.032\tan\phi}{v} - \frac{0.02v}{\cos\phi} \\ \phi' &= \frac{-0.032}{v^2} \end{aligned}$$

over an interval $[0.0, 10.0]$ starting with values $y = 0.5$, $v = 0.5$ and $\phi = \pi/5$ using scalar error control (VECTOL = .FALSE.) until the first point where $y = 0.0$ is encountered.

Also, D02QGF is used to produce output at intervals of 2.0.

10.1 Program Text

```
    Program d02qgfe

!     D02QGF Example Program Text

!     Mark 26 Release. NAG Copyright 2016.

!     .. Use Statements ..
      Use nag_library, Only: d02qgf, d02qwf, nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter     :: alpha = -0.032_nag_wp
      Real (Kind=nag_wp), Parameter     :: beta = -0.02_nag_wp
      Integer, Parameter                :: neqf = 3, neqg = 1, nin = 5,         &
                                           nout = 6
      Integer, Parameter                :: liwork = 21 + 4*neqg
      Integer, Parameter                :: lrwork = 23 + 23*neqf + 14*neqg
!     .. Local Scalars ..
      Real (Kind=nag_wp)                :: grvcm, hmax, t, tcrit, tinc, tout,   &
                                           trvcm, tstart
      Integer                           :: i, ifail, irevcm, j, kgrvcm, latol,  &
                                           lrtol, maxstp, yprvcm, yrvcm
      Logical                           :: alterg, crit, onestp, root, sophst,  &
```

```
                                          vectol
      Character (1)                    :: statef
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: atol(:), rtol(:), rwork(:), y(:)
      Integer, Allocatable             :: iwork(:)
!     .. Intrinsic Procedures ..
      Intrinsic                        :: cos, tan
!     .. Executable Statements ..
      Write (nout,*) 'D02QGF Example Program Results'
!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) latol, lrtol

      Allocate (atol(latol),rtol(lrtol),rwork(lrwork),y(neqf),iwork(liwork))

      Read (nin,*) hmax, tstart, tcrit, tinc
      Read (nin,*) statef
      Read (nin,*) vectol, onestp, crit, sophst
      Read (nin,*) maxstp
      Read (nin,*) rtol(1:lrtol), atol(1:latol)
      Read (nin,*) y(1:neqf)

      t = tstart

!     ifail: behaviour on error exit
!             =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
      ifail = 0
      Call d02qwf(statef,neqf,vectol,atol,latol,rtol,lrtol,onestp,crit,tcrit,  &
        hmax,maxstp,neqg,alterg,sophst,rwork,lrwork,iwork,liwork,ifail)

      Write (nout,*)
      Write (nout,*) '  T         Y(1)     Y(2)     Y(3)'
      Write (nout,99999) t, (y(i),i=1,neqf)
      j = 1
      tout = tinc
      irevcm = 0

revcm: Do
        ifail = -1
        Call d02qgf(neqf,t,y,tout,neqg,root,irevcm,trvcm,yrvcm,yprvcm,grvcm,   &
          kgrvcm,rwork,lrwork,iwork,liwork,ifail)

        Select Case (irevcm)
        Case (0)
          If (ifail==0) Then
!           Print solution at current t
            Write (nout,99999) t, (y(i),i=1,neqf)
            If (t==tout .And. j<5) Then
!             Increment tout and cycle to find solution at this new time.
              j = j + 1
              tout = tout + tinc
              Cycle revcm
            End If
          End If
          Exit revcm
        Case (1:7)
          If (yrvcm==0) Then
            rwork(yprvcm) = tan(y(3))
            rwork(yprvcm+1) = alpha*tan(y(3))/y(2) + beta*y(2)/cos(y(3))
            rwork(yprvcm+2) = alpha/y(2)**2
          Else
            rwork(yprvcm) = tan(rwork(yrvcm+2))
            rwork(yprvcm+1) = alpha*tan(rwork(yrvcm+2))/rwork(yrvcm+1) +       &
              beta*rwork(yrvcm+1)/cos(rwork(yrvcm+2))
            rwork(yprvcm+2) = alpha/rwork(yrvcm+1)**2
          End If
        Case (9:)
          grvcm = y(1)
        End Select
      End Do revcm

99999 Format (1X,F7.4,2X,3(F7.4,2X))
    End Program d02qgfe
```

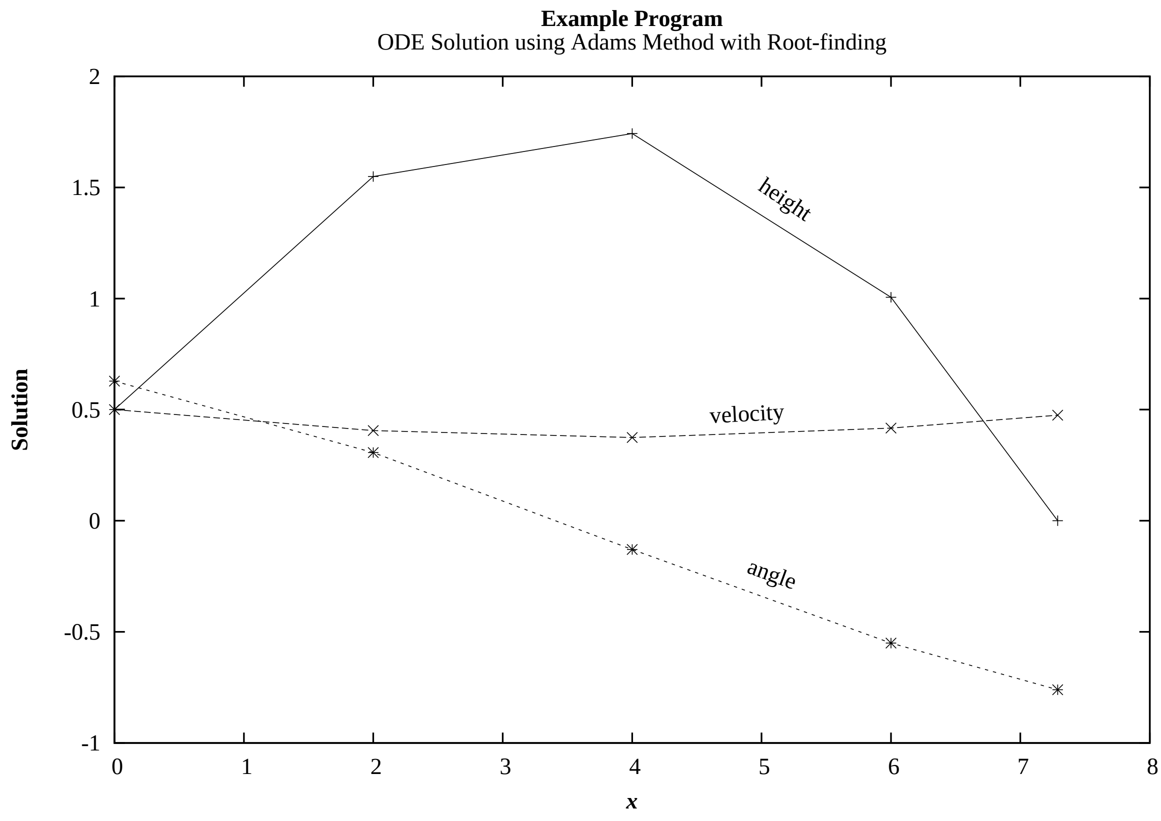
10.2 Program Data

```
D02QGF Example Program Data
   1  1                                   : latol, lrtol
   2.0  0.0  10.0  2.0                    : hmax, tstart, tcrit, tinc
   S                                      : statef
   .FALSE. .FALSE. .TRUE. .TRUE.          : vectol, onestp, crit, sophst
   500                                    : maxstp
   1.0E-4  1.0E-7                         : rtol, atol
   0.5  0.5  6.28318530717958647692E-1    : y
```

10.3 Program Results

```
 D02QGF Example Program Results

   T         Y(1)     Y(2)     Y(3)
  0.0000   0.5000   0.5000   0.6283
  2.0000   1.5493   0.4055   0.3066
  4.0000   1.7423   0.3743  -0.1289
  6.0000   1.0055   0.4173  -0.5507
  7.2883  -0.0000   0.4749  -0.7601
```


Example Program
ODE Solution using Adams Method with Root-finding
Solution
x
height
velocity
angle