NAG Library Routine Document

D02BGF

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

D02BGF integrates a system of first-order ordinary differential equations over an interval with suitable initial conditions, using a Runge–Kutta–Merson method, until a specified component attains a given value.

2 Specification

```
SUBROUTINE D02BGF (X, XEND, N, Y, TOL, HMAX, M, VAL, FCN, W, IFAIL)
INTEGER             N, M, IFAIL
REAL (KIND=nag_wp) X, XEND, Y(N), TOL, HMAX, VAL, W(N,10)
EXTERNAL            FCN
```

3 Description

D02BGF advances the solution of a system of ordinary differential equations

$$y_i' = f_i(x, y_1, y_2, \ldots, y_n), \quad i = 1, 2, \ldots, n,$$

from $x = \mathrm{X}$ towards $x = \mathrm{XEND}$ using a Merson form of the Runge–Kutta method. The system is defined by FCN, which evaluates f_i in terms of x and $y_1, y_2, \ldots, y_n$ (see Section 5), and the values of $y_1, y_2, \ldots, y_n$ must be given at $x = \mathrm{X}$.

As the integration proceeds, a check is made on the specified component y_m of the solution to determine an interval where it attains a given value α. The position where this value is attained is then determined accurately by interpolation on the solution and its derivative. It is assumed that the solution of $y_m = \alpha$ can be determined by searching for a change in sign in the function $y_m - \alpha$.

The accuracy of the integration and, indirectly, of the determination of the position where $y_m = \alpha$ is controlled by the argument TOL.

For a description of Runge–Kutta methods and their practical implementation see Hall and Watt (1976).

4 References

Hall G and Watt J M (ed.) (1976) *Modern Numerical Methods for Ordinary Differential Equations* Clarendon Press, Oxford

5 Arguments

1: X – REAL (KIND=nag_wp) *Input/Output*

On entry: must be set to the initial value of the independent variable x.

On exit: the point where the component y_m attains the value α unless an error has occurred, when it contains the value of x at the error. In particular, if $y_m \neq \alpha$ anywhere on the range $x = \mathrm{X}$ to $x = \mathrm{XEND}$, it will contain XEND on exit.

2: XEND – REAL (KIND=nag_wp) *Input*

On entry: the final value of the independent variable x.

If $\mathrm{XEND} < \mathrm{X}$ on entry integration will proceed in the negative direction.

3: N – INTEGER *Input*

On entry: n, the number of differential equations.

Constraint: N > 0.

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

On entry: the initial values of the solution $y_1, y_2, \ldots, y_n$.

On exit: the computed values of the solution at a point near the solution X, unless an error has occurred when they contain the computed values at the final value of X.

5: TOL – REAL (KIND=nag_wp) *Input/Output*

On entry: must be set to a positive tolerance for controlling the error in the integration and in the determination of the position where $y_m = \alpha$.

D02BGF has been designed so that, for most problems, a reduction in TOL leads to an approximately proportional reduction in the error in the solution obtained in the integration. The relation between changes in TOL and the error in the determination of the position where $y_m = \alpha$ is less clear, but for TOL small enough the error should be approximately proportional to TOL. However, the actual relation between TOL and the accuracy cannot be guaranteed. You are strongly recommended to call D02BGF with more than one value for TOL and to compare the results obtained to estimate their accuracy. In the absence of any prior knowledge you might compare results obtained by calling D02BGF with TOL $= 10.0^{-p}$ and TOL $= 10.0^{-p-1}$ if p correct decimal digits in the solution are required.

Constraint: TOL > 0.0.

On exit: normally unchanged. However if the range from X to the position where $y_m = \alpha$ (or to the final value of X if an error occurs) is so short that a small change in TOL is unlikely to make any change in the computed solution then, on return, TOL has its sign changed. To check results returned with TOL < 0.0, D02BGF should be called again with a positive value of TOL whose magnitude is considerably smaller than that of the previous call.

6: HMAX – REAL (KIND=nag_wp) *Input*

On entry: controls how the sign of $y_m - \alpha$ is checked.

HMAX $= 0.0$
: $y_m - \alpha$ is checked at every internal integration step.

HMAX $\neq 0.0$
: The computed solution is checked for a change in sign of $y_m - \alpha$ at steps of not greater than |HMAX|. This facility should be used if there is any chance of 'missing' the change in sign by checking too infrequently. For example, if two changes of sign of $y_m - \alpha$ are expected within a distance h, say, of each other then a suitable value for HMAX might be HMAX $= h/2$. If only one change of sign in $y_m - \alpha$ is expected on the range X to XEND then HMAX $= 0.0$ is most appropriate.

7: M – INTEGER *Input*

On entry: the index m of the component of the solution whose value is to be checked.

Constraint: $1 \leq$ M $\leq$ N.

8: VAL – REAL (KIND=nag_wp) *Input*

On entry: the value of α in the equation $y_m = \alpha$ to be solved for X.

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

FCN must evaluate the functions f_i (i.e., the derivatives y_i') for given values of its arguments $x, y_1, \ldots, y_n$.

The specification of FCN is:

```
SUBROUTINE FCN (X, Y, F)
REAL (KIND=nag_wp) X, Y(*), F(*)
```

In the description of the arguments of D02BGF below, n denotes the actual value of N in the call of D02BGF.

1: X – REAL (KIND=nag_wp) *Input*

On entry: x, the value of the argument.

2: Y$(*)$ – REAL (KIND=nag_wp) array *Input*

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

3: F$(*)$ – REAL (KIND=nag_wp) array *Output*

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

FCN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which D02BGF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

10: W$(\mathrm{N}, 10)$ – REAL (KIND=nag_wp) array *Workspace*

11: 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, if you are not familiar with this argument, 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, TOL ≤ 0.0,
or N ≤ 0,
or M ≤ 0,
or M $>$ N.

IFAIL $= 2$

With the given value of TOL, no further progress can be made across the integration range from the current point $x =$ X, or dependence of the error on TOL would be lost if further progress across the integration range were attempted (see Section 9 for a discussion of this error exit). The components Y(1), Y(2), $\ldots$, Y(n) contain the computed values of the solution at the current point $x =$ X. No point at which $y_m - \alpha$ changes sign has been located up to the point $x =$ X.

IFAIL $= 3$

TOL is too small for the routine to take an initial step (see Section 9). X and $\mathrm{Y}(1), \mathrm{Y}(2), \ldots, \mathrm{Y}(n)$ retain their initial values.

IFAIL $= 4$

At no point in the range X to XEND did the function $y_m - \alpha$ change sign. It is assumed that $y_m - \alpha$ has no solution.

IFAIL $= 5$ (C05AZF)

A serious error has occurred in an internal call to the specified routine. Check all subroutine calls and array dimensions. Seek expert help.

IFAIL $= 6$

A serious error has occurred in an internal call to an integration routine. Check all subroutine calls and array dimensions. Seek expert help.

IFAIL $= 7$

A serious error has occurred in an internal call to an interpolation routine. Check all (sub) program calls and array dimensions. Seek expert help.

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 depends on TOL, on the mathematical properties of the differential system, on the position where $y_m = \alpha$ and on the method. It can be controlled by varying TOL but the approximate proportionality of the error to TOL holds only for a restricted range of values of TOL. For TOL too large, the underlying theory may break down and the result of varying TOL may be unpredictable. For TOL too small, rounding error may affect the solution significantly and an error exit with IFAIL $= 2$ or 3 is possible.

8 Parallelism and Performance

D02BGF is not threaded in any implementation.

9 Further Comments

The time taken by D02BGF depends on the complexity and mathematical properties of the system of differential equations defined by FCN, on the range, the position of solution and the tolerance. There is also an overhead of the form $a + b \times n$ where a and b are machine-dependent computing times.

For some problems it is possible that D02BGF will exit with IFAIL $= 4$ due to inaccuracy of the computed value y_m. For example, consider a case where the component y_m has a maximum in the

integration range and α is close to the maximum value. If TOL is too large, it is possible that the maximum might be estimated as less than α, or even that the integration step length chosen might be so long that the maximum of y_m and the (two) positions where $y_m = \alpha$ are all in the same step and so the position where $y_m = \alpha$ remains undetected. Both these difficulties can be overcome by reducing TOL sufficiently and, if necessary, by choosing HMAX sufficiently small. For similar reasons, care should be taken when choosing XEND. If possible, you should choose XEND well beyond the point where y_m is expected to equal α, for example $|\mathrm{XEND} - \mathrm{X}|$ should be made about 50% longer than the expected range. As a simple check, if, with XEND fixed, a change in TOL does not lead to a significant change in y_m at XEND, then inaccuracy is not a likely source of error.

If D02BGF fails with IFAIL $= 3$, then it could be called again with a larger value of TOL if this has not already been tried. If the accuracy requested is really needed and cannot be obtained with this routine, the system may be very stiff (see below) or so badly scaled that it cannot be solved to the required accuracy.

If D02BGF fails with IFAIL $= 2$, it is likely that it has been called with a value of TOL which is so small that a solution cannot be obtained on the range X to XEND. This can happen for well-behaved systems and very small values of TOL. You should, however, consider whether there is a more fundamental difficulty. For example:

(a) in the region of a singularity (infinite value) of the solution, the routine will usually stop with IFAIL $= 2$, unless overflow occurs first. If overflow occurs using D02BGF, routine D02PFF can be used instead to detect the increasing solution before overflow occurs. In any case, numerical integration cannot be continued through a singularity, and analytical treatment should be considered;

(b) for 'stiff' equations, where the solution contains rapidly decaying components the routine will use very small steps in x (internally to D02BGF) to preserve stability. This will usually exhibit itself by making the computing time excessively long, or occasionally by an exit with IFAIL $= 2$. Merson's method is not efficient in such cases, and you should try the method D02EJF which uses a Backward Differentiation Formula. To determine whether a problem is stiff, D02PEF may be used.

For well-behaved systems with no difficulties such as stiffness or singularities, the Merson method should work well for low accuracy calculations (three or four figures). For high accuracy calculations or where FCN is costly to evaluate, Merson's method may not be appropriate and a computationally less expensive method may be D02CJF which uses an Adams' method.

For problems for which D02BGF is not sufficiently general, you should consider the routines D02BHF and D02PFF. Routine D02BHF can be used to solve an equation involving the components $y_1, y_2, \ldots, y_n$ and their derivatives (for example, to find where a component passes through zero or to find the maximum value of a component). It also permits a more general form of error control and may be preferred to D02BGF if the component whose value is to be determined is very small in modulus on the integration range. D02BHF can always be used in place of D02BGF, but will usually be computationally more expensive for solving the same problem. D02PFF is a more general routine with many facilities including a more general error control criterion. D02PFF can be combined with the root-finder C05AZF and the interpolation routine D02PSF to solve equations involving $y_1, y_2, \ldots, y_n$ and their derivatives.

This routine is only intended to be used to locate the **first** zero of the function $y_m - \alpha$. If later zeros are required you are strongly advised to construct your own more general root-finding routines as discussed above.

10 Example

This example finds the value $X > 0.0$ where $y = 0.0$, where y, v, ϕ, are defined by

$$y' = \tan\phi$$

$$v' = \frac{-0.032\tan\phi}{v} - \frac{0.02v}{\cos\phi}$$

$$\phi\prime = \frac{-0.032}{v^2}$$

and where at $X = 0.0$ we are given $y = 0.5$, $v = 0.5$ and $\phi = \pi/5$. We write $y = Y(1)$, $v = Y(2)$ and $\phi = Y(3)$ and we set TOL $= 1.0E{-}4$ and TOL $= 1.0E{-}5$ in turn so that we can compare the solutions obtained. We expect the solution $X \simeq 7.3$ and we set XEND $= 10.0$ so that the point where $y = 0.0$ is not too near the end of the range of integration. The initial values and range are read from a data file.

10.1 Program Text

```
!   D02BGF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module d02bgfe_mod

!     D02BGF 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
!     .. Parameters ..
      Integer, Parameter, Public         :: n = 3, nin = 5, nout = 6
!     n: number of differential equations
    Contains
      Subroutine fcn(x,y,f)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: x
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: f(*)
        Real (Kind=nag_wp), Intent (In) :: y(*)
!       .. Intrinsic Procedures ..
        Intrinsic                        :: cos, tan
!       .. Executable Statements ..
        f(1) = tan(y(3))
        f(2) = -0.032E0_nag_wp*tan(y(3))/y(2) - 0.02E0_nag_wp*y(2)/cos(y(3))
        f(3) = -0.032E0_nag_wp/y(2)**2
        Return
      End Subroutine fcn
    End Module d02bgfe_mod

    Program d02bgfe

!     D02BGF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02bgf, nag_wp
      Use d02bgfe_mod, Only: fcn, n, nin, nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                 :: alpha, hmax, tol, val, x, xend,      &
                                            xinit
      Integer                            :: i, ifail, m
```

```
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: w(:,:), y(:), yinit(:)
!     .. Executable Statements ..
      Write (nout,*) 'D02BGF Example Program Results'
!     Skip heading in data file
      Read (nin,*)
!     m: index of mode of solution to attain value alpha
      Read (nin,*) m
      Allocate (w(n,10),y(n),yinit(n))
!     xinit: initial x value,     xend : final x value.
!     alpha: attain y(m) = alpha, yinit: initial solution values.
      Read (nin,*) alpha
      Read (nin,*) xinit
      Read (nin,*) xend
      Read (nin,*) yinit(1:n)
      hmax = 0.0E0_nag_wp
      val = alpha
      Do i = 4, 5
        tol = 10.0E0_nag_wp**(-i)
        x = xinit
        y(1:n) = yinit(1:n)

!       ifail: behaviour on error exit
!              =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
        ifail = 0
        Call d02bgf(x,xend,n,y,tol,hmax,m,val,fcn,w,ifail)

        Write (nout,*)
        Write (nout,99999) 'Calculation with TOL =', tol
        Write (nout,99998) ' Y(M) changes sign at X = ', x
        If (tol<0.0E0_nag_wp) Then
          Write (nout,*) ' Over one-third steps controlled by HMAX'
        End If
      End Do

99999 Format (1X,A,E8.1)
99998 Format (1X,A,F7.4)
    End Program d02bgfe
```

10.2 Program Data

```
D02BGF Example Program Data
   1                                        : m
   0.0                                      : alpha
   0.0                                      : xinit
  10.0                                      : xend
   0.5  0.5  6.28318530717958647692E-1      : yinit
```

10.3 Program Results

```
 D02BGF Example Program Results

 Calculation with TOL = 0.1E-03
  Y(M) changes sign at X =  7.2884

 Calculation with TOL = 0.1E-04
  Y(M) changes sign at X =  7.2883
```