NAG Library Routine Document

D02BHF

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

D02BHF integrates a system of first-order ordinary differential equations over an interval with suitable initial conditions, using a Runge–Kutta–Merson method, until a user-specified function of the solution is zero.

2 Specification

```
SUBROUTINE D02BHF (X, XEND, N, Y, TOL, IRELAB, HMAX, FCN, G, W, IFAIL)

INTEGER             N, IRELAB, IFAIL
REAL (KIND=nag_wp) X, XEND, Y(N), TOL, HMAX, G, W(N,7)
EXTERNAL            FCN, G
```

3 Description

D02BHF advances the solution of a system of ordinary differential equations

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

from $x =$ X towards $x =$ 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 $x =$ X.

As the integration proceeds, a check is made on the function $g(x, y)$ specified by you, to determine an interval where it changes sign. The position of this sign change is then determined accurately by interpolating for the solution and its derivative. It is assumed that $g(x, y)$ is a continuous function of the variables, so that a solution of $g(x, y) = 0$ can be determined by searching for a change in sign in $g(x, y)$.

The accuracy of the integration and, indirectly, of the determination of the position where $g(x, y) = 0$, is controlled by 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 Parameters

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 $g(x, y) = 0.0$ unless an error has occurred, when it contains the value of x at the error. In particular, if $g(x, y) \neq 0.0$ anywhere on the range X to 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 XEND $<$ X on entry, integration proceeds in a 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 the final point $x =$ 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 $g(x, y) = 0.0$.

D02BHF 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 $g(x, y) = 0.0$ 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 D02BHF 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 D02BHF 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 $x =$ X to the position where $g(x, y) = 0.0$ (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 TOL is returned with its sign changed. To check results returned with TOL < 0.0, D02BHF should be called again with a positive value of TOL whose magnitude is considerably smaller than that of the previous call.

6: IRELAB – INTEGER *Input*

On entry: determines the type of error control. At each step in the numerical solution an estimate of the local error, *est*, is made. For the current step to be accepted the following condition must be satisfied:

IRELAB $= 0$
$est \leq$ TOL $\times \max\{1.0, |y_1|, |y_2|, \ldots, |y_n|\}$;

IRELAB $= 1$
$est \leq$ TOL;

IRELAB $= 2$
$est \leq$ TOL $\times \max\{\epsilon, |y_1|, |y_2|, \ldots, |y_n|\}$, where ϵ is ***machine precision***.

If the appropriate condition is not satisfied, the step size is reduced and the solution recomputed on the current step.

If you wish to measure the error in the computed solution in terms of the number of correct decimal places, then set IRELAB $= 1$ on entry, whereas if the error requirement is in terms of the number of correct significant digits, then set IRELAB $= 2$. Where there is no preference in the choice of error test, IRELAB $= 0$ will result in a mixed error test. It should be borne in mind that the computed solution will be used in evaluating $g(x, y)$.

Constraint: IRELAB $= 0$, 1 or 2.

7: HMAX – REAL (KIND=nag_wp) *Input*

On entry: if HMAX $= 0.0$, no special action is taken.

If HMAX $\neq 0.0$, a check is made for a change in sign of $g(x, y)$ at steps 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 $g(x,y)$ 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 $g(x,y)$ is expected on the range X to XEND, then the choice HMAX $= 0.0$ is most appropriate.

8: 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 parameters of D02BHF below, n denotes the value of N in the call of D02BHF.

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 D02BHF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

G must evaluate the function $g(x,y)$ at a specified point.

The specification of G is:

```
FUNCTION G (X, Y)
REAL (KIND=nag_wp) G
REAL (KIND=nag_wp) X, Y(*)
```

In the description of the parameters of D02BHF below, n denotes the value of N in the call of D02BHF.

1: X – REAL (KIND=nag_wp) *Input*

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

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

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

G must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02BHF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

10: W(N,7) – 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 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, TOL ≤ 0.0,
or N ≤ 0,
or IRELAB $\neq 0$, 1 or 2.

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 8 for a discussion of this error exit). The components $\mathrm{Y}(1), \mathrm{Y}(2), \ldots, \mathrm{Y}(n)$ contain the computed values of the solution at the current point $x =$ X. No point at which $g(x, y)$ changes sign has been located up to the point $x =$ X.

IFAIL $= 3$

TOL is too small for D02BHF to take an initial step (see Section 8). 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 $g(x, y)$ change sign. It is assumed that $g(x, y) = 0.0$ 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.

7 Accuracy

The accuracy depends on TOL, on the mathematical properties of the differential system, on the position where $g(x, y) = 0.0$ 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.

The accuracy may also be restricted by the properties of $g(x, y)$. You should try to code G without introducing any unnecessary cancellation errors.

8 Further Comments

The time taken by D02BHF depends on the complexity and mathematical properties of the system of differential equations defined by FCN, the complexity of G, on the range, the position of the 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 D02BHF will return IFAIL $= 4$ because of inaccuracy of the computed values Y, leading to inaccuracy in the computed values of $g(x, y)$ used in the search for the solution of $g(x, y) = 0.0$. This difficulty can be overcome by reducing TOL sufficiently, and if necessary, by choosing HMAX sufficiently small. If possible, you should choose XEND well beyond the expected point where $g(x, y) = 0.0$; for example make |XEND − X| about 50% larger 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 at XEND, then inaccuracy is not a likely source of error.

If D02BHF 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 D02BHF 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 D02BHF, 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 compute in very small steps in x (internally to D02BHF) 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 D02EJF which uses a Backward Differentiation Formula method. 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 D02BHF is not sufficiently general, you should consider D02PFF. D02PFF is a more general routine with many facilities including a more general error control criterion. D02PFF can be combined with the rootfinder C05AZF and the interpolation routine D02PSF to solve equations involving $y_1, y_2, \ldots, y_n$ and their derivatives.

D02BHF can also be used to solve an equation involving x, $y_1, y_2, \ldots, y_n$ and the derivatives of $y_1, y_2, \ldots, y_n$. For example in Section 9, D02BHF is used to find a value of X > 0.0 where Y$(1) = 0.0$. It could instead be used to find a turning-point of y_1 by replacing the function $g(x, y)$ in the program by:

```
REAL (kind=nag_wp) FUNCTION G(X,Y)
REAL (kind=nag_wp) X,Y(3),F(3)
```

```
      CALL FCN(X,Y,F)
      G = F(1)
      RETURN
      END
```

This routine is only intended to locate the **first** zero of $g(x,y)$. If later zeros are required, you are strongly advised to construct your own more general root-finding routines as discussed above.

9 Example

This example finds the value $\mathrm{X} > 0.0$ at which $y = 0.0$, where y, v, ϕ are defined by

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

and where at $\mathrm{X} = 0.0$ we are given $y = 0.5$, $v = 0.5$ and $\phi = \pi/5$. We write $y = \mathrm{Y}(1)$, $v = \mathrm{Y}(2)$ and $\phi = \mathrm{Y}(3)$ and we set TOL $= 1.0\mathrm{E}{-4}$ and TOL $= 1.0\mathrm{E}{-5}$ in turn so that we can compare the solutions. We expect the solution $\mathrm{X} \simeq 7.3$ and so we set XEND $= 10.0$ to avoid determining the solution of $y = 0.0$ too near the end of the range of integration. The initial values and range are read from a data file.

9.1 Program Text

```
!   D02BHF Example Program Text
!   Mark 24 Release. NAG Copyright 2012.

    Module d02bhfe_mod

!     D02BHF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter                   :: n = 3, nin = 5, nout = 6
!   n: number of differential equations
    Contains
      Subroutine fcn(x,y,f)

!       .. Parameters ..
        Real (Kind=nag_wp), Parameter        :: alpha = -0.032E0_nag_wp
        Real (Kind=nag_wp), Parameter        :: beta = -0.02E0_nag_wp
!       .. 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) = alpha*tan(y(3))/y(2) + beta*y(2)/cos(y(3))
        f(3) = alpha/y(2)**2
        Return
      End Subroutine fcn

      Function g(x,y)

!       .. Function Return Value ..
        Real (Kind=nag_wp)                   :: g
!       .. Scalar Arguments ..
```

```
        Real (Kind=nag_wp), Intent (In)      :: x
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: y(*)
!       .. Executable Statements ..
        g = y(1)
        Return
      End Function g
    End Module d02bhfe_mod

    Program d02bhfe

!     D02BHF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02bhf, nag_wp
      Use d02bhfe_mod, Only: fcn, g, n, nin, nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                   :: hmax, tol, x, xend, xinit
      Integer                              :: i, ifail, irelab, j
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable      :: w(:,:), y(:), yinit(:)
!     .. Executable Statements ..
      Write (nout,*) 'D02BHF Example Program Results'
      Allocate (w(n,7),y(n),yinit(n))
!     Skip heading in data file
      Read (nin,*)
!     xinit: initial x value,         xend  : final x value.
!     yinit: initial solution values, irelab: type of error control.
      Read (nin,*) xinit
      Read (nin,*) xend
      Read (nin,*) yinit(1:n)
      Read (nin,*) irelab
      hmax = 0.0E0_nag_wp
      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 d02bhf(x,xend,n,y,tol,irelab,hmax,fcn,g,w,ifail)

        Write (nout,*)
        Write (nout,99999) 'Calculation with TOL =', tol
        Write (nout,99998) ' Root of Y(1) at', x
        Write (nout,99997) ' Solution is', (y(j),j=1,n)
        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)
99997 Format (1X,A,3F13.5)
    End Program d02bhfe
```

9.2 Program Data

```
D02BHF Example Program Data
   0.0                                       : xinit
  10.0                                       : xend
   0.5  0.5  6.28318530717958647692E-1       : yinit
   0                                         : irelab
```

9.3 Program Results

```
 D02BHF Example Program Results

 Calculation with TOL = 0.1E-03
  Root of Y(1) at 7.2884
  Solution is       0.00000      0.47485     -0.76010

 Calculation with TOL = 0.1E-04
  Root of Y(1) at 7.2883
  Solution is      -0.00000      0.47486     -0.76011
```