

# NAG Toolbox for MATLAB

## d02bh

### 1 Purpose

d02bh 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 Syntax

```
[x, y, tol, ifail] = d02bh(x, xend, y, tol, irelab, hmax, fcn, g, 'n', n)
```

### 3 Description

d02bh advances the solution of a system of ordinary differential equations

$$y'_i = f_i(x, y_1, y_2, \dots, y_n), \quad i = 1, 2, \dots, n,$$

from  $x = \mathbf{x}$  towards  $x = \mathbf{xend}$  using a Merson form of the Runge–Kutta method. The system is defined by user-supplied (sub)program **fcn**, which evaluates  $f_i$  in terms of  $x$  and  $y_1, y_2, \dots, y_n$  (see Section 5), and the values of  $y_1, y_2, \dots, y_n$  must be given at  $x = \mathbf{x}$ .

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 the parameter **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

#### 5.1 Compulsory Input Parameters

1: **x – double scalar**

Must be set to the initial value of the independent variable  $x$ .

2: **xend – double scalar**

The final value of the independent variable  $x$ .

If **xend** < **x** on entry, integration proceeds in a negative direction.

3: **y(n) – double array**

The initial values of the solution  $y_1, y_2, \dots, y_n$ .

4: **tol – double scalar**

Must be set to a **positive** tolerance for controlling the error in the integration and in the determination of the position where  $g(x, y) = 0.0$ .

d02bh 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 d02bh 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 d02bh 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.

5: **irelab – int32 scalar**

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

$$est \leq \mathbf{tol} \times \max\{1.0, |y_1|, |y_2|, \dots, |y_n|\};$$

**irelab** = 1

$$est \leq \mathbf{tol};$$

**irelab** = 2

$$est \leq \mathbf{tol} \times \max\{\epsilon, |y_1|, |y_2|, \dots, |y_n|\}, \text{ where } \epsilon \text{ is } \textit{machine precision}.$$

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 **irelab** should be given the value 1 on entry, whereas if the error requirement is in terms of the number of correct significant digits, then **irelab** should be given the value 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:*  $0 \leq \mathbf{irelab} \leq 2$ .

6: **hmax – double scalar**

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.

7: **fcn – string containing name of m-file**

**fcn** must evaluate the functions  $f_i$  (i.e., the derivatives  $y'_i$ ) for given values of its arguments  $x, y_1, \dots, y_n$ .

Its specification is:

$$[f] = \mathbf{fcn}(x, y)$$

**Input Parameters**

1: **x – double scalar**

The value of the argument  $x$ .

2: **y(n) – double array**  
 The value of the argument  $y_i$ , for  $i = 1, 2, \dots, n$ .

#### Output Parameters

1: **f(n) – double array**  
 The value of  $f_i$ , for  $i = 1, 2, \dots, n$ .

8: **g – string containing name of m-file**

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

Its specification is:

```
[result] = g(x, y)
```

#### Input Parameters

1: **x – double scalar**  
 The value of the independent variable  $x$ .

2: **y(n) – double array**  
 The value of  $y_i$ , for  $i = 1, 2, \dots, n$ .

#### Output Parameters

1: **result – double scalar**  
 The result of the function.

## 5.2 Optional Input Parameters

1: **n – int32 scalar**

*Default:* The dimension of the array **y**.

$n$ , the number of differential equations.

*Constraint:*  $n > 0$ .

## 5.3 Input Parameters Omitted from the MATLAB Interface

w

## 5.4 Output Parameters

1: **x – double scalar**

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: **y(n) – double array**

The computed values of the solution at the final point  $x = \mathbf{x}$ .

3: **tol – double scalar**

Normally unchanged. However if the range from  $x = \mathbf{x}$  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, d02bh should be called again with a positive value of **tol** whose magnitude is considerably smaller than that of the previous call.

4: **ifail** – **int32** scalar

0 unless the function detects an error (see Section 6).

## 6 Error Indicators and Warnings

Errors or warnings detected by the function:

**ifail** = 1

On entry, **tol** ≤ 0.0,  
or **n** ≤ 0,  
or **irelab** ≠ 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  $x = \mathbf{x}$ , 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  $\mathbf{y}(1), \mathbf{y}(2), \dots, \mathbf{y}(n)$  contain the computed values of the solution at the current point  $x = \mathbf{x}$ . No point at which  $g(x, y)$  changes sign has been located up to the point  $x = \mathbf{x}$ .

**ifail** = 3

**tol** is too small for d02bh to take an initial step (see Section 8).  $\mathbf{x}$  and  $\mathbf{y}(1), \mathbf{y}(2), \dots, \mathbf{y}(n)$  retain their initial values.

**ifail** = 4

At no point in the range  $\mathbf{x}$  to  $\mathbf{xend}$  did the function  $g(x, y)$  change sign. It is assumed that  $g(x, y) = 0.0$  has no solution.

**ifail** = 5 (c05az)

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

**ifail** = 6

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

**ifail** = 7

A serious error has occurred in an internal call to an interpolation function. 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 user-supplied real function **g** without introducing any unnecessary cancellation errors.

## 8 Further Comments

The time taken by d02bh depends on the complexity and mathematical properties of the system of differential equations defined by user-supplied (sub)program **fcn**, the complexity of user-supplied real function **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 d02bh 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  $|\mathbf{xend} - \mathbf{x}|$  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 d02bh 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 function, the system may be very stiff (see below) or so badly scaled that it cannot be solved to the required accuracy.

If d02bh 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 function will usually stop with **ifail** = 2, unless overflow occurs first. If overflow occurs using d02bh, d02pd 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 function will compute in very small steps in  $x$  (internally to d02bh) 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 d02ej which uses a Backward Differentiation Formula method. To determine whether a problem is stiff, d02pc 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 user-supplied (sub)program **fcn** is costly to evaluate, Merson’s method may not be appropriate and a computationally less expensive method may be d02cj which uses an Adams method.

For problems for which d02bh is not sufficiently general, you should consider d02pd. d02pd is a more general function with many facilities including a more general error control criterion. d02pd can be combined with the rootfinder c05az and the interpolation function d02px to solve equations involving  $y_1, y_2, \dots, y_n$  and their derivatives.

d02bh can also be used to solve an equation involving  $x, y_1, y_2, \dots, y_n$  and the derivatives of  $y_1, y_2, \dots, y_n$ . For example in Section 9, d02bh 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:

```
function result = g(x,y)
    f = d02bh_f(x,y);
    result = f(1);
```

This function 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 functions as discussed above.

## 9 Example

```
d02bh_fcn.m

function f = fcn(x,y)
    f = zeros(3,1);
```

```
f(1) = tan(y(3));  
f(2) = -0.032*tan(y(3))/y(2) - 0.02*y(2)/cos(y(3));  
f(3) = -0.032/y(2)^2;
```

```
d02bh_g.m
```

```
function result = g(x,y)  
    result = y(1);
```

```
x = 0;  
xend = 10;  
y = [0.5;  
     0.5;  
     0.6283185307179586];  
tol = 0.0001;  
irelab = int32(0);  
hmax = 0;  
[xOut, yOut, tolOut, ifail] = ...  
    d02bh(x, xend, y, tol, irelab, hmax, 'd02bh_fcn', 'd02bh_g')
```

```
xOut =  
    7.2884  
yOut =  
    0.0000  
    0.4749  
   -0.7601  
tolOut =  
    1.0000e-04  
ifail =  
         0
```