# NAG Toolbox for MATLAB

# d02hb

## 1 Purpose

d02hb solves the two-point boundary-value problem for a system of ordinary differential equations, using initial value techniques and Newton iteration; it generalizes (sub)program d02ha to include the case where parameters other than boundary-values are to be determined.

## 2 Syntax

```
[p, soln, w, ifail] = d02hb(p, pe, e, m1, fcn, bc, range, 'n1', n1, 'n',
n)
```

## 3 Description

d02hb solves the two-point boundary-value problem by determining the unknown parameters $p_1, p_2, \ldots, p_{n_1}$ of the problem. These parameters may be, but need not be, boundary-values; they may include eigenvalue parameters in the coefficients of the differential equations, length of the range of integration, etc. The notation and methods used are similar to those of d02ha and you are advised to study this first. (The parameters $p_1, p_2, \ldots, p_{n_1}$ correspond precisely to the unknown boundary conditions in d02ha.) It is assumed that we have a system of $n$ first-order ordinary differential equations of the form:

$$\frac{dy_i}{dx} = f_i(x, y_1, y_2, \ldots, y_n), \qquad i = 1, 2, \ldots, n,$$

and that the derivatives $f_i$ are evaluated by user-supplied (sub)program **fcn**. The system, including the boundary conditions given by user-supplied (sub)program **bc** and the range of integration given by user-supplied (sub)program **range**, involves the $n_1$ unknown parameters $p_1, p_2, \ldots, p_{n_1}$ which are to be determined, and for which initial estimates must be supplied. The number of unknown parameters $n_1$ must not exceed the number of equations $n$. If $n_1 < n$, we assume that $(n - n_1)$ equations of the system are not involved in the matching process. These are usually referred to as 'driving equations'; they are independent of the parameters and of the solutions of the other $n_1$ equations. In numbering the equations for the **fcn**, the driving equations must be put **first**.

The estimated values of the parameters are corrected by a form of Newton iteration. The Newton correction on each iteration is calculated using a Jacobian matrix whose $(i,j)$th element depends on the derivative of the $i$th component of the solution, $y_i$, with respect to the $j$th parameter, $p_j$. This matrix is calculated by a simple numerical differentiation technique which requires $n_1$ evaluations of the differential system.

If the parameter **ifail** is set appropriately, the function automatically prints messages to inform you of the flow of the calculation. These messages are discussed in detail in Section 8.

d02hb is a simplified version of d02sa which is described in detail in Gladwell 1979a.

## 4 References

Gladwell I 1979a The development of the boundary value codes in the ordinary differential equations chapter of the NAG Library *Codes for Boundary Value Problems in Ordinary Differential Equations. Lecture Notes in Computer Science* (ed B Childs, M Scott, J W Daniel, E Denman and P Nelson) **76** Springer–Verlag

## 5 Parameters

You are strongly recommended to read Sections 3 and 8 in conjunction with this section.

## 5.1 Compulsory Input Parameters

1: **p(n1) – double array**

An estimate for the $i$th parameter, $p_i$, for $i = 1, 2, \ldots, n_1$.

2: **pe(n1) – double array**

The elements of **pe** must be given small positive values. The element **pe**$(i)$ is used

(i) in the convergence test on the $i$th parameter in the Newton iteration, and

(ii) in perturbing the $i$th parameter when approximating the derivatives of the components of the solution with respect to this parameter for use in the Newton iteration.

The elements **pe**$(i)$ should not be chosen too small. They should usually be several orders of magnitude larger than ***machine precision***.

*Constraint*: **pe**$(i) > 0.0$, for $i = 1, 2, \ldots,$ **n1**.

3: **e(n) – double array**

The elements of **e** must be given positive values. The element **e**$(i)$ is used in the bound on the local error in the $i$th component of the solution $y_i$ during integration.

The elements **e**$(i)$ should not be chosen too small. They should usually be several orders of magnitude larger than ***machine precision***.

*Constraint*: **e**$(i) > 0.0$, for $i = 1, 2, \ldots,$ **n**.

4: **m1 – int32 scalar**

a value which controls exit values.

**m1** $= 1$

The final solution is not calculated.

**m1** $> 1$

The final values of the solution at interval (length of range)/(**m1** $- 1$) are calculated and stored sequentially in the array **soln** starting with the values of the solutions evaluated at the first end point (see user-supplied (sub)program **range**) stored in the first column of **soln**.

*Constraint*: **m1** $\geq 1$.

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

**fcn** must evaluate the function $f_i$ (i.e., the derivative $y_i'$), for $i = 1, 2, \ldots, n$.

Its specification is:

```
[f] = fcn(x, y, p)
```

**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 $i = 1, 2, \ldots, n$.

3: **p(n1) – double array**

The current estimate of the parameter $p_i$, for $i = 1, 2, \ldots, n_1$.

**Output Parameters**

1: **f**$(n)$ – **double array**

The value of $f_i$, for $i = 1, 2, \ldots, n$. The $f_i$ may depend upon the parameters $p_j$, for $j = 1, 2, \ldots, n_1$. If there are any driving equations (see Section 3) then these must be numbered first in the ordering of the components of **f** in **fcn**.

6: **bc – string containing name of m-file**

**bc** must place in **g1** and **g2** the boundary conditions at $a$ and $b$ respectively (see user-supplied (sub)program **range**).

Its specification is:

```
[g1, g2] = bc(p)
```

**Input Parameters**

1: **p**$(n1)$ – **double array**

An estimate of the parameter $p_i$, for $i = 1, 2, \ldots, n_1$.

**Output Parameters**

1: **g1**$(n)$ – **double array**

The value of $y_i(a)$, (where this may be a known value or a function of the parameters $p_j$, for $j = 1, 2, \ldots, n_1$; $i = 1, 2, \ldots, n$).

2: **g2**$(n)$ – **double array**

The value of $y_i(b)$, for $i = 1, 2, \ldots, n$, (where these may be known values or functions of the parameters $p_j$, for $j = 1, 2, \ldots, n_1$). If $n > n_1$, so that there are some driving equations, then the first $n - n_1$ values of **g2** need not be set since they are never used.

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

**range** must evaluate the boundary points $a$ and $b$, each of which may depend on the parameters $p_1, p_2, \ldots, p_{n_1}$. The integrations in the shooting method are always from $a$ to $b$.

Its specification is:

```
[a, b] = range(p)
```

**Input Parameters**

1: **p**$(n1)$ – **double array**

The current estimate of the $i$th parameter, $p_i$, for $i = 1, 2, \ldots, n_1$.

**Output Parameters**

1: **a – double scalar**

$a$, one of the boundary points.

2: **b – double scalar**

The second boundary point, $b$. Note that **b** > **a** forces the direction of integration to be that of increasing $x$. If **a** and **b** are interchanged the direction of integration is reversed.

### 5.2 Optional Input Parameters

1: **n1 – int32 scalar**

*Default*: The dimension of the arrays **p**, **pe**. (An error is raised if these dimensions are not equal.)

$n_1$, the number of parameters.

*Constraint*: $1 \leq \mathbf{n1} \leq \mathbf{n}$.

2: **n – int32 scalar**

*Default*: The dimension of the arrays **e**, **soln**, **w**. (An error is raised if these dimensions are not equal.)

$n$, the total number of differential equations.

*Constraint*: $\mathbf{n} \geq 2$.

### 5.3 Input Parameters Omitted from the MATLAB Interface

sdw

### 5.4 Output Parameters

1: **p(n1) – double array**

The corrected value for the $i$th parameter, unless an error has occurred, when it contains the last calculated value of the parameter.

2: **soln(n,m1) – double array**

The solution when $\mathbf{m1} > 1$ (see below).

3: **w(n,sdw) – double array**

Used mainly for workspace.

With $\mathbf{ifail} = 2$, 3, 4 or 5 (see Section 6), $\mathbf{w}(i,1)$, for $i = 1,2,\ldots,n$, contains the solution at the point $x$ when the error occurred. $\mathbf{w}(1,2)$ contains $x$.

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** $= 1$

One or more of the parameters **n**, **n1**, **m1**, **sdw**, **e** or **pe** is incorrectly set.

**ifail** $= 2$

The step length for the integration became too short whilst calculating the residual (see Section 8).

**ifail** $= 3$

No initial step length could be chosen for the integration whilst calculating the residual.

**Note**: **ifail** $= 2$ or 3 can occur due to choosing too small a value for **e** or due to choosing the wrong direction of integration. Try varying **e** and interchanging $a$ and $b$. These error exits can also occur for very poor initial choices of the parameters in the array **p** and, in extreme cases, because this function cannot be used to solve the problem posed.

**ifail** = 4

As for **ifail** = 2 but the error occurred when calculating the Jacobian.

**ifail** = 5

As for **ifail** = 3 but the error occurred when calculating the Jacobian.

**ifail** = 6

The calculated Jacobian has an insignificant column. This can occur because a parameter $p_i$ is incorrectly entered when posing the problem.

**Note**: **ifail** = 4, 5 or 6 usually indicate a badly scaled problem. You may vary the size of **pe**. Otherwise the use of the more general d02sa which affords more control over the calculations is advised.

**ifail** = 7

The linear algebra function used (f08kb) has failed. This error exit should not occur and can be avoided by changing the initial estimates $p_i$.

**ifail** = 8

The Newton iteration has failed to converge. This can indicate a poor initial choice of parameters $p_i$ or a very difficult problem. Consider varying the elements $\mathbf{pe}(i)$ if the residuals are small in the monitoring output. If the residuals are large, try varying the initial parameters $p_i$.

**ifail** = 9
**ifail** = 10
**ifail** = 11
**ifail** = 12
**ifail** = 13

Indicate that a serious error has occurred. Check all array subscripts and (sub)program parameter lists in the call to d02hb. Seek expert help.

## 7 Accuracy

If the process converges, the accuracy to which the unknown parameters are determined is usually close to that specified by you; and the solution, if requested, may be determined to a required accuracy by varying the parameter **e**.

## 8 Further Comments

The time taken by d02hb depends on the complexity of the system, and on the number of iterations required. In practice, integration of the differential equations is by far the most costly process involved.

Wherever they occur in the function, the error parameters contained in the arrays **e** and **pe** are used in 'mixed' form; that is $\mathbf{e}(i)$ always occurs in expressions of the form

$$\mathbf{e}(i) \times (1 + |y_i|)$$

and $\mathbf{pe}(i)$ always occurs in expressions of the form

$$\mathbf{pe}(i) \times (1 + |p_i|).$$

Though not ideal for every application, it is expected that this mixture of absolute and relative error testing will be adequate for most purposes.

You may determine a suitable direction of integration $a$ to $b$ and suitable values for $\mathbf{e}(i)$ by integrations with d02pc. The best direction of integration is usually the direction of decreasing solutions. You are strongly recommended to set **ifail** to obtain self-explanatory error messages, and also monitoring information about the course of the computation. You may select the channel numbers on which this output is to appear by calls of x04aa (for error messages) or x04ab (for monitoring information) – see Section 9 for an example. Otherwise the default channel numbers will be used. The monitoring

information produced at each iteration includes the current parameter values, the residuals and two norms: a basic norm and a current norm. At each iteration the aim is to find parameter values which make the current norm less than the basic norm. Both these norms should tend to zero as should the residuals. (They would all be zero if the exact parameters were used as input.) For more details, in particular about the other monitoring information printed, you are advised to consult the specification of d02sa and, especially, the description of the parameter (sub)program **monit** there.

The computing time for integrating the differential equations can sometimes depend critically on the quality of the initial estimates for the parameters $p_i$. If it seems that too much computing time is required and, in particular, if the values of the residuals printed by the monitoring function are much larger than the expected values of the solution at $b$ then the coding of the user-supplied (sub)programs **fcn**, **bc** and **range** should be checked for errors. If no errors can be found, an independent attempt should be made to improve the initial estimates for $p_i$.

The (sub)program can be used to solve a very wide range of problems, for example:

(a) eigenvalue problems, including problems where the eigenvalue occurs in the boundary conditions;

(b) problems where the differential equations depend on some parameters which are to be determined so as to satisfy certain boundary conditions (see Example 2 in Section 9);

(c) problems where one of the end points of the range of integration is to be determined as the point where a variable $y_i$ takes a particular value (see Example 2 in Section 9);

(d) singular problems and problems on infinite ranges of integration where the values of the solution at $a$ or $b$ or both are determined by a power series or an asymptotic expansion (or a more complicated expression) and where some of the coefficients in the expression are to be determined (see Example 1 in Section 9); and

(e) differential equations with certain terms defined by other independent (driving) differential equations.

# 9 Example

d02hb_bc.m

```
function [g1, g2] = bc(p)
  z=0.1;
  g1(1) = 0.1+p(1)*sqrt(z)*0.1+0.01*z;
  g1(2) = p(1)*0.05/sqrt(z) + 0.01;
  g2(1) = 1/6;
  g2(2) = p(2);
```

d02hb_fcn.m

```
function f = fcn(x, y, p)

  f = zeros(2,1);
  f(1) = y(2);
  f(2) = (y(1)^3-y(2))/2/x;
```

d02hb_range.m

```
function [a, b] = range(p)
  a = 0.1;
  b = 16.0;
```

```
p = [0.2;
     0];
pe = [1e-05;
     0.001];
e = [0.0001;
```

```
     0.0001];
m1 = int32(6);
[pOut, soln, w, ifail] = ...
    d02hb(p, pe, e, m1, 'd02hb_fcn', 'd02hb_bc', 'd02hb_range')
```

```
pOut =
    0.0463
    0.0035
soln =
    0.1025    0.1217    0.1338    0.1449    0.1557    0.1667
    0.0173    0.0042    0.0036    0.0034    0.0034    0.0035
w =
  Columns 1 through 7
    1.0000    0.0000    0.0000    0.0000    0.0463    0.0000    0.0000
    1.0000    0.0000    0.0000   -0.0000    0.0035   -0.0000    0.0000
  Columns 8 through 14
    0.0000    1.0012    0.9988    0.0360    0.9994    0.0608   -0.9982
    0.0000    0.5919    1.6896    0.9994   -0.0360    0.9982    0.0608
  Columns 15 through 21
    0.1667    0.1667    0.0035    0.1617    0.0034    0.1513    0.0034
    0.0035    0.0035    0.0000    0.0034    0.0000    0.0034    0.0000
  Columns 22 through 28
   -0.0000    0.0001    0.0000    0.1667   -0.0000    0.0001    0.0000
    0.0000    0.0001    0.0000    0.0035    0.0000    0.0001    0.0000
  Columns 29 through 31
         0    0.5926         0
         0    0.0395   -1.0000
ifail =
           0
```