NAG Toolbox for MATLAB

d02bg

1 Purpose

d02bg 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 Syntax

```
[x, y, tol, ifail] = d02bg(x, xend, y, tol, hmax, m, val, fcn, 'n', n)
```

3 Description

d02bg 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 $x = \mathbf{x}$ 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 $y_1, y_2, \ldots, y_n$ (see Section 5), and the values of $y_1, y_2, \ldots, y_n$ must be given at $x = \mathbf{x}$.

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 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 will proceed in the negative direction.

3: **y(n) – double array**

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

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 $y_m = \alpha$.

d02bg 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 d02bg 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 d02bg 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: **hmax – double scalar**

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.

6: **m – int32 scalar**

The index m of the component of the solution whose value is to be checked.

Constraint: $1 \leq$ **m** $\leq$ **n**.

7: **val – double scalar**

The value of α in the equation $y_m = \alpha$ to be solved for **x**.

8: **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 $x, y_1, \ldots, y_n$.

Its specification is:

```
[f] = 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 $i = 1, 2, \ldots, n$.

Output Parameters

1: **f**(n) **– double array**

The value of f_i, for $i = 1, 2, \ldots, n$.

5.2 Optional Input Parameters

1: **n – int32 scalar**

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

n, the number of differential equations.

Constraint: $\mathbf{n} > 0$.

5.3 Input Parameters Omitted from the MATLAB Interface

w

5.4 Output Parameters

1: **x – double scalar**

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 $x = \mathbf{x}$ to $x = \mathbf{xend}$, it will contain **xend** on exit.

2: **y(n) – double array**

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**.

3: **tol – double scalar**

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 $\mathbf{tol} < 0.0$, d02bg 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, $\mathbf{tol} \leq 0.0$,
or $\mathbf{n} \leq 0$,
or $\mathbf{m} \leq 0$,
or $\mathbf{m} > \mathbf{n}$.

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 $\mathbf{y}(1), \mathbf{y}(2), \ldots, \mathbf{y}(n)$ contain the computed values of the solution at the current point $x = \mathbf{x}$. No point at which $y_m - \alpha$ changes sign has been located up to the point $x = \mathbf{x}$.

ifail = 3

tol is too small for the function to take an initial step (see Section 8). **x** and $\mathbf{y}(1), \mathbf{y}(2), \ldots, \mathbf{y}(n)$ 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 $= 5$ (c05az)

A serious error has occurred 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 $y_m = \alpha$ 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 Further Comments

The time taken by d02bg depends on the complexity and mathematical properties of the system of differential equations defined by user-supplied (sub)program **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 d02bg 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 $|$**xend** $-$ **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 d02bg 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 d02bg 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 d02bg, function 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 use very small steps in x (internally to d02bg) 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 d02ej which uses a Backward Differentiation Formula. 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 d02bg is not sufficiently general, you should consider the functions d02pd and d02bh. Routine d02bh 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 d02bg if the component whose value is to be determined is very small in modulus on the integration range. d02bh can always be used in place of d02bg, but will usually be computationally more expensive for solving the same problem. d02pd is a more general function with many facilities including a more general error control criterion. d02pd can be combined with the root-finder c05az and the interpolation function d02px to solve equations involving $y_1, y_2, \ldots, y_n$ and their derivatives.

This function 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 functions as discussed above.

9 Example

```
d02bg_fcn.m
```

```
function [f] = d02bg_fcn(x,y)

      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;
```

```
x = 0;
xend = 10;
y = [0.5;
     0.5;
     0.6283185307179586];
tol = 0.0001;
hmax = 0;
m = int32(1);
val = 0;
[xOut, yOut, tolOut, ifail] = d02bg(x, xend, y, tol, hmax, m, val,
'd02bg_fcn')
```

```
xOut =
    7.2884
yOut =
   -0.6112
    0.5064
   -0.8390
tolOut =
   1.0000e-04
ifail =
           0
```