NAG Toolbox for MATLAB

d02ga

1 Purpose

d02ga solves the two-point boundary-value problem with assigned boundary-values for a system of ordinary differential equations, using a deferred correction technique and a Newton iteration.

2 Syntax

```
[x, y, np, ifail] = d02ga(u, v, a, b, tol, fcn, x, np, 'n', n, 'mnp', mnp)
```

3 Description

d02ga solves a two-point boundary-value problem for a system of n differential equations in the interval $[a,b]$. The system is written in the form

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

and the derivatives are evaluated by user-supplied (sub)program **fcn**. Initially, n boundary-values of the variables y_i must be specified (assigned), some at a and some at b. You also supply estimates of the remaining n boundary-values and all the boundary-values are used in constructing an initial approximation to the solution. This approximate solution is corrected by a finite-difference technique with deferred correction allied with a Newton iteration to solve the finite-difference equations. The technique used is described fully in Pereyra 1979. The Newton iteration requires a Jacobian matrix $\dfrac{\partial f_i}{\partial y_j}$ and this is calculated by numerical differentiation using an algorithm described in Curtis *et al.* 1974.

You supply an absolute error tolerance and may also supply an initial mesh for the construction of the finite-difference equations (alternatively a default mesh is used). The algorithm constructs a solution on a mesh defined by adding points to the initial mesh. This solution is chosen so that the error is everywhere less than your tolerance and so that the error is approximately equidistributed on the final mesh. The solution is returned on this final mesh.

If the solution is required at a few specific points then these should be included in the initial mesh. If on the other hand the solution is required at several specific points then you should use the interpolation functions provided in Chapter E01 if these points do not themselves form a convenient mesh.

4 References

Curtis A R, Powell M J D and Reid J K 1974 On the estimation of sparse Jacobian matrices *J. Inst. Maths. Applics.* **13** 117–119

Pereyra V 1979 PASVA3: An adaptive finite-difference Fortran program for first order nonlinear, ordinary boundary problems *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

5.1 Compulsory Input Parameters

1: **u(n,2) – double array**

$\mathbf{u}(i,1)$ must be set to the known (assigned) or estimated values of y_i at a and $\mathbf{u}(i,2)$ must be set to the known or estimated values of y_i at b, for $i = 1,2,\ldots,n$.

2: **v(n,2) – double array**

$\mathbf{v}(i,j)$ must be set to 0.0 if $\mathbf{u}(i,j)$ is a known (assigned) value and to 1.0 if $\mathbf{u}(i,j)$ is an estimated value, for $i = 1,2,\ldots,n$; $j = 1,2$.

Constraint: precisely **n** of the $\mathbf{v}(i,j)$ must be set to 0.0, i.e., precisely **n** of the $\mathbf{u}(i,j)$ must be known values, and these must not be all at a or all at b.

3: **a – double scalar**

a, the left-hand boundary point.

4: **b – double scalar**

b, the right-hand boundary point.

Constraint: $\mathbf{b} > \mathbf{a}$.

5: **tol – double scalar**

A positive absolute error tolerance. If

$$a = x_1 < x_2 < \cdots < x_{\mathbf{np}} = b$$

is the final mesh, $z_j(x_i)$ is the jth component of the approximate solution at x_i, and $y_j(x)$ is the jth component of the true solution of equation (1) (see Section 3) and the boundary conditions, then, except in extreme cases, it is expected that

$$\left|z_j(x_i) - y_j(x_i)\right| \leq \mathbf{tol}, \qquad i = 1,2,\ldots,\mathbf{np}; j = 1,2,\ldots,n. \tag{2}$$

Constraint: $\mathbf{tol} > 0.0$.

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

fcn must evaluate the functions f_i (i.e., the derivatives y_i') at the general point x.

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

7: **x(mnp) – double array**

If $\mathbf{np} \geq 4$ (see **np**), the first **np** elements must define an initial mesh. Otherwise the elements of **x** need not be set.

Constraint:

$$\mathbf{a} = \mathbf{x}(1) < \mathbf{x}(2) < \cdots < \mathbf{x}(\mathbf{np}) = \mathbf{b}, \qquad \mathbf{np} \geq 4. \tag{3}$$

8: **np – int32 scalar**

Determines whether a default or user-supplied mesh is used.

np $= 0$

A default value of 4 for **np** and a corresponding equispaced mesh $\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(\mathbf{np})$ are used.

np ≥ 4

You must define an initial mesh using the array **x** as described.

Constraint: **np** $= 0$ or $4 \leq$ **np** $\leq$ **mnp**.

5.2 Optional Input Parameters

1: **n – int32 scalar**

Default: The dimension of the arrays **u**, **v**, **y**. (An error is raised if these dimensions are not equal.) the number of equations.

Constraint: **n** ≥ 2.

2: **mnp – int32 scalar**

Default: The dimension of the arrays **x**, **y**. (An error is raised if these dimensions are not equal.) the maximum permitted number of mesh points.

Constraint: **mnp** ≥ 32.

5.3 Input Parameters Omitted from the MATLAB Interface

w, lw, iw, liw

5.4 Output Parameters

1: **x(mnp) – double array**

$\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(\mathbf{np})$ define the final mesh (with the returned value of **np**) satisfying the relation (3).

2: **y(n,mnp) – double array**

The approximate solution $z_j(x_i)$ satisfying (2), on the final mesh, that is

$$\mathbf{y}(j,i) = z_j(x_i), \qquad i = 1, 2, \ldots, \mathbf{np}; j = 1, 2, \ldots, n,$$

where **np** is the number of points in the final mesh.

The remaining columns of **y** are not used.

3: **np – int32 scalar**

The number of points in the final (returned) mesh.

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**, **tol**, **np**, **mnp**, **lw** or **liw** has been incorrectly set, or $\mathbf{b} \leq \mathbf{a}$, or the condition (3) on **x** is not satisfied, or the number of known boundary-values (specified by **v**) is not **n**.

ifail $= 2$

The Newton iteration has failed to converge. This could be due to there being too few points in the initial mesh or to the initial approximate solution being too inaccurate. If this latter reason is suspected you should use (sub)program d02ra instead. If the warning 'Jacobian matrix is singular' is printed this could be due to specifying zero estimated boundary-values and these should be varied. This warning could also be printed in the unlikely event of the Jacobian matrix being calculated inaccurately. If you cannot make changes to prevent the warning then (sub)program d02ra should be used.

ifail $= 3$

The Newton iteration has reached round-off level. It could be, however, that the answer returned is satisfactory. This error might occur if too much accuracy is requested.

ifail $= 4$

A finer mesh is required for the accuracy requested; that is **mnp** is not large enough.

ifail $= 5$

A serious error has occurred in a call to d02ga. Check all array subscripts and (sub)program parameter lists in calls to d02ga. Seek expert help.

7 Accuracy

The solution returned by the function will be accurate to your tolerance as defined by the relation (2) except in extreme circumstances. If too many points are specified in the initial mesh, the solution may be more accurate than requested and the error may not be approximately equidistributed.

8 Further Comments

The time taken by d02ga depends on the difficulty of the problem, the number of mesh points used (and the number of different meshes used), the number of Newton iterations and the number of deferred corrections.

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.

A common cause of convergence problems in the Newton iteration is that you have specified too few points in the initial mesh. Although the function adds points to the mesh to improve accuracy it is unable to do so until the solution on the initial mesh has been calculated in the Newton iteration.

If you specify zero known **and** estimated boundary-values, the function constructs a zero initial approximation and in many cases the Jacobian is singular when evaluated for this approximation, leading to the breakdown of the Newton iteration.

You may be unable to provide a sufficiently good choice of initial mesh and estimated boundary-values, and hence the Newton iteration may never converge. In this case the continuation facility provided in d02ra is recommended.

In the case where you wish to solve a sequence of similar problems, the final mesh from solving one case is strongly recommended as the initial mesh for the next.

9 Example

d02ga_fcn.m

```
function f = fcn(x,y)
      f = zeros(3,1);
      f(1) = y(2);
      f(2) = y(3);
      f(3) = -y(1)*y(3);
```

```
u = [0, 10;
     0, 1;
     0, 0];
v = [0, 1;
     0, 0;
     1, 1];
a = 0;
b = 10;
tol = 0.001;
x = zeros(40, 1);
x(1:26) = [0;
     0.4;
     0.8;
     1.2;
     1.6;
     2;
     2.4;
     2.8;
     3.2;
     3.6;
     4;
     4.4;
     4.8;
     5.2;
     5.6;
     6;
     6.4;
     6.8;
     7.2;
     7.6;
     8;
     8.4;
     8.8;
     9.2;
     9.6;
     10];
np = int32(26);
[xOut, y, npOut, ifail] = d02ga(u, v, a, b, tol, 'd02ga_fcn', x, np)
```

```
xOut =
         0
    0.4000
    0.8000
    1.2000
    1.6000
    2.0000
    2.4000
    2.8000
    3.2000
    3.6000
    4.0000
    4.4000
```

```
    4.8000
    5.2000
    5.6000
    6.0000
    6.4000
    6.8000
    7.2000
    7.6000
    8.0000
    8.4000
    8.8000
    9.2000
    9.6000
   10.0000
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
y =
  Columns 1 through 7
         0    0.0375    0.1497    0.3336    0.5828    0.8864    1.2309
         0    0.1876    0.3719    0.5450    0.6963    0.8163    0.9009
    0.4695    0.4673    0.4511    0.4104    0.3424    0.2558    0.1678
  Columns 8 through 14
    1.6026    1.9900    2.3851    2.7834    3.1829    3.5828    3.9828
    0.9529    0.9805    0.9930    0.9978    0.9994    0.9999    1.0000
    0.0953    0.0464    0.0193    0.0069    0.0021    0.0006    0.0001
  Columns 15 through 21
    4.3828    4.7828    5.1828    5.5828    5.9828    6.3828    6.7828
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
    0.0000    0.0000    0.0000    0.0000   -0.0000    0.0000   -0.0000
  Columns 22 through 28
    7.1828    7.5828    7.9828    8.3828    8.7828         0         0
    1.0000    1.0000    1.0000    1.0000    1.0000         0         0
    0.0000   -0.0000    0.0000   -0.0000   -0.0000         0         0
  Columns 29 through 35
         0         0         0         0         0         0         0
         0         0         0         0         0         0         0
         0         0         0         0         0         0         0
  Columns 36 through 40
         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0
npOut =
          26
ifail =
           0
```