NAG Toolbox

nag_ode_bvp_fd_lin_gen (d02gb)

1 Purpose

nag_ode_bvp_fd_lin_gen (d02gb) solves a general linear two-point boundary value problem for a system of ordinary differential equations, using a deferred correction technique.

2 Syntax

```
[c, d, gam, x, y, np, ifail] = nag_ode_bvp_fd_lin_gen(a, b, tol, fcnf, fcng, c, d, gam, x, np, 'n', n, 'mnp', mnp)
[c, d, gam, x, y, np, ifail] = d02gb(a, b, tol, fcnf, fcng, c, d, gam, x, np, 'n', n, 'mnp', mnp)
```

3 Description

nag_ode_bvp_fd_lin_gen (d02gb) solves a linear two-point boundary value problem for a system of n ordinary differential equations in the interval $[a, b]$. The system is written in the form

$$y' = F(x)y + g(x) \tag{1}$$

and the boundary conditions are written in the form

$$Cy(a) + Dy(b) = \gamma. \tag{2}$$

Here $F(x)$, C and D are n by n matrices, and $g(x)$ and γ are n-component vectors. The approximate solution to (1) and (2) is found using a finite difference method with deferred correction. The algorithm is a specialization of that used in function nag_ode_bvp_fd_nonlin_gen (d02ra) which solves a nonlinear version of (1) and (2). The nonlinear version of the algorithm is described fully in Pereyra (1979).

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

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* (eds B Childs, M Scott, J W Daniel, E Denman and P Nelson) **76** Springer–Verlag

5 Parameters

5.1 Compulsory Input Parameters

1: **a** – REAL (KIND=nag_wp)

a, the left-hand boundary point.

2: **b** – REAL (KIND=nag_wp)

b, the right-hand boundary point.

Constraint: **b** $>$ **a**.

3: **tol** – REAL (KIND=nag_wp)

A positive absolute error tolerance. If

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

is the final mesh, $z(x)$ is the approximate solution from nag_ode_bvp_fd_lin_gen (d02gb) and $y(x)$ is the true solution of equations (1) and (2) then, except in extreme cases, it is expected that

$$\|z - y\| \leq \mathbf{tol} \tag{3}$$

where

$$\|u\| = \max_{1 \leq i \leq \mathbf{n1}} \max_{1 \leq j \leq \mathbf{np}} \left|u_i(x_j)\right|.$$

Constraint: **tol** > 0.0.

4: **fcnf** – SUBROUTINE, supplied by the user.

fcnf must evaluate the matrix $F(x)$ in (1) at a general point x.

```
[f] = fcnf(x)
```

Input Parameters

1: **x** – REAL (KIND=nag_wp)

x, the value of the independent variable.

Output Parameters

1: **f**(:) – REAL (KIND=nag_wp) array

f$(n \times (i-1) + j)$ must contain the (i,j)th element of the matrix $F(x)$, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$. (See Section 10 for an example.)

5: **fcng** – SUBROUTINE, supplied by the user.

fcng must evaluate the vector $g(x)$ in (1) at a general point x.

```
[g] = fcng(x)
```

Input Parameters

1: **x** – REAL (KIND=nag_wp)

x, the value of the independent variable.

Output Parameters

1: **g**(:) – REAL (KIND=nag_wp) array

The ith element of the vector $g(x)$, for $i = 1, 2, \ldots, n$. (See Section 10 for an example.)

6: **c**(**n**, **n**) – REAL (KIND=nag_wp) array
7: **d**(**n**, **n**) – REAL (KIND=nag_wp) array
8: **gam**(**n**) – REAL (KIND=nag_wp) array

The arrays **c** and **d** must be set to the matrices C and D in (2)). **gam** must be set to the vector γ in (2).

9: **x**(**mnp**) – REAL (KIND=nag_wp) array

If **np** ≥ 4 (see **np**), the first **np** elements must define an initial mesh. Otherwise the elements of x need not be set.

Constraint:

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

10: **np** – INTEGER

Determines whether a default mesh 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 **x** as in (4).

5.2 Optional Input Parameters

1: **n** – INTEGER

Default: the dimension of the array **gam** and the first dimension of the arrays **c**, **d** and the second dimension of the arrays **c**, **d**. (An error is raised if these dimensions are not equal.)

The number of equations; that is n is the order of system (1).

Constraint: **n** ≥ 2.

2: **mnp** – INTEGER

Default: the dimension of the array **x**.

The maximum permitted number of mesh points.

Constraint: **mnp** ≥ 32.

5.3 Output Parameters

1: **c**(**n**, **n**) – REAL (KIND=nag_wp) array
2: **d**(**n**, **n**) – REAL (KIND=nag_wp) array
3: **gam**(**n**) – REAL (KIND=nag_wp) array

The rows of **c** and **d** and the components of **gam** are reordered so that the boundary conditions are in the order:

(i) conditions on $y(a)$ only;

(ii) condition involving $y(a)$ and $y(b)$; and

(iii) conditions on $y(b)$ only.

The function will be slightly more efficient if the arrays **c**, **d** and **gam** are ordered in this way before entry, and in this event they will be unchanged on exit.

Note that the problems (1) and (2) must be of boundary value type, that is neither C nor D may be identically zero. Note also that the rank of the matrix $[C, D]$ must be n for the problem to be properly posed. Any violation of these conditions will lead to an error exit.

4: **x**(**mnp**) – REAL (KIND=nag_wp) 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 (4).

5: **y**(**n**, **mnp**) – REAL (KIND=nag_wp) array

The approximate solution $z(x)$ satisfying (3), on the final mesh, that is

$$\mathbf{y}(j,i) = z_j(x_i), \quad i = 1, 2, \ldots, \mathbf{np} \text{ and } j = 1, 2, \ldots, n$$

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

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

6: **np** – INTEGER

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

7: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 1$

One or more of the arguments **n**, **tol**, **np**, **mnp**, lw or liw is incorrectly set, **b** $\leq$ **a** or the condition (4) on **x** is not satisfied.

ifail $= 2$

There are three possible reasons for this error exit to be taken:

(i) one of the matrices C or D is identically zero (that is, the problem is of initial value and not boundary value type). In this case, $iw(1) = 0$ on exit;

(ii) a row of C and the corresponding row of D are identically zero (that is, the boundary conditions are rank deficient). In this case, on exit $iw(1)$ contains the index of the first such row encountered; and

(iii) more than n of the columns of the n by $2n$ matrix $[C, D]$ are identically zero (that is, the boundary conditions are rank deficient). In this case, on exit $iw(1)$ contains minus the number of non-identically zero columns.

ifail $= 3$ (*warning*)

The function has failed to find a solution to the specified accuracy. There are a variety of possible reasons including:

(i) the boundary conditions are rank deficient, which may be indicated by the message that the Jacobian is singular. However this is an unlikely explanation for the error exit as all rank deficient boundary conditions should lead instead to error exits with either **ifail** $= 2$ or 5; see also (iv);

(ii) not enough mesh points are permitted in order to attain the required accuracy. This is indicated by **np** $=$ **mnp** on return from a call to nag_ode_bvp_fd_lin_gen (d02gb). This difficulty may be aggravated by a poor initial choice of mesh points;

(iii) the accuracy requested cannot be attained on the computer being used; and

(iv) an unlikely combination of values of Fx has led to a singular Jacobian. The error should not persist if more mesh points are allowed.

ifail $= 4$

A serious error has occurred in a call to nag_ode_bvp_fd_lin_gen (d02gb). Check all array subscripts and function argument lists in calls to nag_ode_bvp_fd_lin_gen (d02gb). Seek expert help.

ifail $= 5$

There are two possible reasons for this error exit which occurs when checking the rank of the boundary conditions by reduction to a row echelon form:

(i) at least one row of the n by $2n$ matrix $[C, D]$ is a linear combination of the other rows and hence the boundary conditions are rank deficient. The index of the first such row encountered is given by $iw(1)$ on exit; and

(ii) as (i) but the rank deficiency implied by this error exit has only been determined up to a numerical tolerance. Minus the index of the first such row encountered is given by $iw(1)$ on exit.

In case (ii) there is some doubt as to the rank deficiency of the boundary conditions. However even if the boundary conditions are not rank deficient they are not posed in a suitable form for use with this function.

For example, if

$$C = \begin{pmatrix} 1 & 0 \\ 1 & \epsilon \end{pmatrix}, \quad D = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}, \quad \gamma = \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix}$$

and ϵ is small enough, this error exit is likely to be taken. A better form for the boundary conditions in this case would be

$$C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad D = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad \gamma = \begin{pmatrix} \gamma_1 \\ \epsilon^{-1}(\gamma_2 - \gamma_1) \end{pmatrix}.$$

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

The solution returned by the function will be accurate to your tolerance as defined by the relation (3) 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 nag_ode_bvp_fd_lin_gen (d02gb) depends on the difficulty of the problem, the number of mesh points (and meshes) used 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 nag_file_set_unit_error (x04aa) (for error messages) or nag_file_set_unit_advisory (x04ab) (for monitoring information) – see Section 10 for an example. Otherwise the default channel numbers will be used.

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

This example solves the problem (written as a first-order system)

$$\epsilon y'' + y' = 0$$

with boundary conditions

$$y(0) = 0, \quad y(1) = 1$$

for the cases $\epsilon = 10^{-1}$ and $\epsilon = 10^{-2}$ using the default initial mesh in the first case, and the final mesh of the first case as initial mesh for the second (more difficult) case. We give the solution and the error at each mesh point to illustrate the accuracy of the method given the accuracy request **tol** $= 1.0\text{e}{-3}$.

Note the call to nag_file_set_unit_advisory (x04ab) prior to the call to nag_ode_bvp_fd_lin_gen (d02gb).

9.1 Program Text

```
function d02gb_example

fprintf('d02gb example results\n\n');

% For communication with f.
global eps;

% Set up initial values before calling NAG routine for the first time.
mnp = nag_int(70);
a = 0;
b = 1;
tol = 0.001;
c = [1, 0; 0, 0];
d = [0, 0; 1, 0];
gam = [0; 1];
x = zeros(1,mnp);
np = nag_int(0);
n = nag_int(2);
xkeep = zeros(1,2);
npkeep = zeros(2);
ykeep = zeros(1,2);
espkeep = zeros(2);

for i = 1:2
    eps = 10^(-i);
    [c, d, gam, x, y, np, ifail] = ...
        d02gb(a, b, tol, @f, @g, c, d, gam, x, np, 'n', n, 'mnp', mnp);
    % save results.
    for j = 1:np
        xkeep(j,i) = x(j);
        ykeep(j,i) = y(1,j);
    end
    % Store the number of points and epsilon for use in plot.
    npkeep(i) = np;
    espkeep(i) = eps;
    fprintf(' for eps = %7.4f, number of points required, np = %d\n', eps, np);
end

% Plot results.
fig1 = figure;
plot(xkeep(1:npkeep(1),1), ykeep(1:npkeep(1),1), '-+', ...
     xkeep(1:npkeep(2),2), ykeep(1:npkeep(2),2), ':x');
set(gca, 'YLim', [0 1.1]);
set(gca, 'XLim', [-0.01 1.01]);
title('Solution to Simple Second-order Problem');
xlabel('x');
ylabel('Solution');
legend(['\epsilon = ', num2str(espkeep(1))], ...
       ['\epsilon = ', num2str(espkeep(2))], 'Location', 'East');
```

```
function f = f(x)
% For communication with main routine.
global eps;

% Evaluate matrix.
f=zeros(2,2);
f(1,2) = 1;
f(2,2) = -1/eps;

function g = g(x)
% Evaluate vector.
g=zeros(2,1);
```

9.2 Program Results

```
     d02gb example results

 for eps =  0.1000, number of points required, np = 15
 for eps =  0.0100, number of points required, np = 49
```