

NAG Library Function Document

nag_ode_bvp_fd_nonlin_gen (d02rac)

1 Purpose

nag_ode_bvp_fd_nonlin_gen (d02rac) solves the two-point boundary value problem with general boundary conditions for a system of ordinary differential equations, using a deferred correction technique and Newton iteration.

2 Specification

```
#include <nag.h>
#include <nagd02.h>

void nag_ode_bvp_fd_nonlin_gen (Integer neq, double *deleps,
    void (*fcn)(Integer neq, double x, double eps, const double y[], double f[],
        Nag_User *comm),
    Integer numbeg, Integer nummix,
    void (*g)(Integer neq, double eps, const double ya[], const double yb[],
        double bc[], Nag_User *comm),
    Nag_MeshSet init, Integer mnp, Integer *np, double x[], double y[],
    double tol, double abt[],
    void (*jacobj)(Integer neq, double x, double eps, const double y[],
        double f[], Nag_User *comm),
    void (*jacobjg)(Integer neq, double eps, const double ya[], const double yb[],
        double aj[], double bj[], Nag_User *comm),
    void (*jaceps)(Integer neq, double x, double eps, const double y[],
        double f[], Nag_User *comm),
    void (*jacep)(Integer neq, double eps, const double ya[], const double yb[],
        double bcep[], Nag_User *comm),
    Nag_User *comm, NagError *fail)
```

3 Description

nag_ode_bvp_fd_nonlin_gen (d02rac) solves a two-point boundary value problem for a system of **neq** ordinary differential equations in the interval $[a, b]$ with $b > a$. The system is written in the form

$$y'_i = f_i(x, y_1, y_2, \dots, y_{\text{neq}}), i = 1, 2, \dots, \text{neq} \quad (1)$$

and the derivatives f_i are evaluated by **fcn**. With the differential equations (1) must be given a system of **neq** (nonlinear) boundary conditions

$$g_i(y(a), y(b)) = 0, i = 1, 2, \dots, \text{neq}$$

where

$$y(x) = [y_1(x), y_2(x), \dots, y_{\text{neq}}(x)]^T. \quad (2)$$

The functions g_i are evaluated by **g**. The solution is computed using a finite-difference technique with deferred correction allied to a Newton iteration to solve the finite-difference equations. The technique used is described fully in Pereyra (1979).

You must supply an absolute error tolerance and may also supply an initial mesh for the finite-difference equations and an initial approximate solution (alternatively a default mesh and approximation are used). The approximate solution is corrected using a Newton iteration and deferred correction. Then, additional points are added to the mesh and the solution is recomputed with the aim of making the error everywhere

less than your tolerance and of approximately equidistributing the error 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.

The Newton iteration requires Jacobian matrices

$$\left(\frac{\partial f_i}{\partial y_j}\right), \left(\frac{\partial g_i}{\partial y_j(a)}\right) \text{ and } \left(\frac{\partial g_i}{\partial y_j(b)}\right).$$

These may be supplied through functions **jacobjf** for $\left(\frac{\partial f_i}{\partial y_j}\right)$ and **jacobjg** for the others. Alternatively the Jacobians may be calculated by numerical differentiation using the algorithm described in Curtis *et al.* (1974).

For problems of the type (1) and (2) for which it is difficult to determine an initial approximation from which the Newton iteration will converge, a continuation facility is provided. You must set up a family of problems

$$y' = f(x, y, \epsilon), g(y(a), y(b), \epsilon) = 0, \quad (3)$$

where $f = [f_1, f_2, \dots, f_{\text{neq}}]^T$ etc, and where ϵ is a continuation argument. The choice $\epsilon = 0$ must give a problem (3) which is easy to solve and $\epsilon = 1$ must define the problem whose solution is actually required. The function solves a sequence of problems with ϵ values

$$0 = \epsilon_1 < \epsilon_2 < \dots < \epsilon_p = 1. \quad (4)$$

The number p and the values ϵ_i are chosen by the function so that each problem can be solved using the solution of its predecessor as a starting approximation. Jacobians $\frac{\partial f}{\partial \epsilon}$ and $\frac{\partial g}{\partial \epsilon}$ are required and they may be supplied via functions **jacpeps** and **jacgep** respectively or may be computed by numerical differentiation.

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

5 Arguments

1: **neq** – Integer *Input*

On entry: the number of differential equations, **neq**.

Constraint: **neq** > 0.

2: **deleps** – double * *Input/Output*

On entry: must be given a value which specifies whether continuation is required.

If **deleps** ≤ 0.0 or **deleps** ≥ 1.0

It is assumed that continuation is not required.

0.0 < **deleps** < 1.0

It is assumed that continuation is required unless **deleps** < $\sqrt{\text{machine precision}}$ when an error exit is taken.

deleps is used as the increment $\epsilon_2 - \epsilon_1$ (see (4)) and the choice **deleps** = 0.1 is recommended.

On exit: an overestimate of the increment $\epsilon_p - \epsilon_{p-1}$ (in fact the value of the increment which would have been tried if the restriction $\epsilon_p = 1$ had not been imposed). If continuation was not requested then **deleps** = 0.0.

If continuation is not requested then the arguments **jaceps** and **jacgep** may be replaced by the NAG defined two null functions macro, **NULL_2_FUN**.

- 3: **fcn** – function, supplied by the user *External Function*

fcn must evaluate the functions f_i (i.e., the derivatives y'_i) at a general point x for a given value of ϵ , the continuation argument (see Section 3).

The specification of **fcn** is:

```
void fcn (Integer neq, double x, double eps, const double y[], double f[],
         Nag_User *comm)
```

1: **neq** – Integer *Input*

On entry: the number of equations.

2: **x** – double *Input*

On entry: the value of the argument x .

3: **eps** – double *Input*

On entry: the value of the continuation argument, ϵ . This is 1 if continuation is not being used.

4: **y[neq]** – const double *Input*

On entry: $y[i - 1]$ contains the value of the argument y_i , for $i = 1, 2, \dots, \mathbf{neq}$.

5: **f[neq]** – double *Output*

On exit: $f[i - 1]$ must contain the values of f_i , for $i = 1, 2, \dots, \mathbf{neq}$.

6: **comm** – Nag_User *

Pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/exit: the pointer **comm** \rightarrow **p** should be cast to the required type, e.g.,
 struct user *s = (struct user *)comm \rightarrow p, to obtain the original object's
 address with appropriate type. (See the argument **comm** below.)

- 4: **numbeg** – Integer *Input*

On entry: the number of left-hand boundary conditions (that is the number involving $y(a)$ only).

Constraint: $0 \leq \mathbf{numbeg} < \mathbf{neq}$.

- 5: **nummix** – Integer *Input*

On entry: the number of coupled boundary conditions (that is the number involving both $y(a)$ and $y(b)$).

Constraint: $0 \leq \mathbf{nummix} \leq \mathbf{neq} - \mathbf{numbeg}$.

- 6: **g** – function, supplied by the user *External Function*

g must evaluate the boundary conditions in equation (3) and place them in the array **bc**.

The specification of **g** is:

```
void g (Integer neq, double eps, const double ya[], const double yb[],
       double bc[], Nag_User *comm)
```

- | | | |
|----|---|---------------|
| 1: | neq – Integer | <i>Input</i> |
| | <i>On entry:</i> the number of equations. | |
| 2: | eps – double | <i>Input</i> |
| | <i>On entry:</i> the value of the continuation argument, ϵ . This is 1 if continuation is not being used. | |
| 3: | ya[neq] – const double | <i>Input</i> |
| | <i>On entry:</i> ya [$i - 1$] contains the value $y_i(a)$, for $i = 1, 2, \dots, \mathbf{neq}$. | |
| 4: | yb[neq] – const double | <i>Input</i> |
| | <i>On entry:</i> yb [$i - 1$] contains the value $y_i(b)$, for $i = 1, 2, \dots, \mathbf{neq}$. | |
| 5: | bc[neq] – double | <i>Output</i> |
| | <i>On exit:</i> must contain the values $g_i(y(a), y(b), \epsilon)$, for $i = 1, 2, \dots, \mathbf{neq}$. These must be ordered as follows: | |
| | (i) first, the conditions involving only $y(a)$ (see numbeg description above); | |
| | (ii) next, the nummix coupled conditions involving both $y(a)$ and $y(b)$ (see nummix description above); and, | |
| | (iii) finally, the conditions involving only $y(b)$ (neq – numbeg – nummix). | |
| 6: | comm – Nag_User * | |
| | Pointer to a structure of type Nag_User with the following member: | |
| | p – Pointer | |
| | <i>On entry/exit:</i> the pointer comm → p should be cast to the required type, e.g.,
struct user *s = (struct user *)comm → p, to obtain the original object's address with appropriate type. (See the argument comm below.) | |

- | | | |
|----|--|---------------------|
| 7: | init – Nag_MeshSet | <i>Input</i> |
| | <i>On entry:</i> indicates whether you wish to supply an initial mesh and approximate solution (init = Nag_UserInitMesh) or whether default values are to be used, (init = Nag_DefInitMesh). | |
| | <i>Constraint:</i> init = Nag_UserInitMesh or Nag_DefInitMesh. | |
| 8: | mnp – Integer | <i>Input</i> |
| | <i>On entry:</i> must be set to the maximum permitted number of points in the finite-difference mesh. | |
| | <i>Constraint:</i> mnp \geq 32. | |
| 9: | np – Integer * | <i>Input/Output</i> |
| | <i>On entry:</i> must be set to the number of points to be used in the initial mesh. | |
| | <i>Constraint:</i> $4 \leq \mathbf{np} \leq \mathbf{mnp}$. | |
| | <i>On exit:</i> the number of points in the final mesh. | |

10: **x[**mnp**]** – double *Input/Output*

On entry: you must set $\mathbf{x}[0] = a$ and $\mathbf{x}[\mathbf{np} - 1] = b$. If **init** = Nag_DefInitMesh on entry a default equispaced mesh will be used, otherwise you must specify a mesh by setting $\mathbf{x}[i - 1] = x_i$, for $i = 2, 3, \dots, \mathbf{np} - 1$.

Constraints:

if **init** = Nag_DefInitMesh, $\mathbf{x}[0] < \mathbf{x}[\mathbf{np} - 1]$;
 if **init** = Nag_UserInitMesh, $\mathbf{x}[0] < \mathbf{x}[1] < \dots < \mathbf{x}[\mathbf{np} - 1]$.

On exit: $\mathbf{x}[0], \mathbf{x}[1], \dots, \mathbf{x}[\mathbf{np} - 1]$ define the final mesh (with the returned value of **np**) and $\mathbf{x}[0] = a$ and $\mathbf{x}[\mathbf{np} - 1] = b$.

11: **y[**neq** × **mnp**]** – double *Input/Output*

On entry: if**init** = Nag_DefInitMesh, **y** need not be set.

If **init** = Nag_UserInitMesh, **y** must contain an initial approximation to the solution such that $\mathbf{y}[j - 1][i - 1]$ contains an approximation to $y_j(x_i)$, $i = 1, 2, \dots, \mathbf{np}$, $j = 1, 2, \dots, \mathbf{neq}$.

On exit: the approximate solution $z_j(x_i)$ satisfying (5) on the final mesh, that is $\mathbf{y}[j - 1][i - 1] = z_j(x_i)$, $i = 1, 2, \dots, \mathbf{np}$, $j = 1, 2, \dots, \mathbf{neq}$, where **np** is the number of points in the final mesh. If an error has occurred then **y** contains the latest approximation to the solution. The remaining columns of **y** are not used.

12: **tol** – double *Input*

On entry: a positive absolute error tolerance. If $a = x_1 < x_2 < \dots < x_{\mathbf{np}} = b$ is the final mesh, $z_j(x_i)$ is the j th component of the approximate solution at x_i , and $y_j(x)$ is the j th component of the true solution of (1) and (2), then, except in extreme circumstances, it is expected that

$$|z_j(x_i) - y_j(x_i)| \leq \mathbf{tol}, \quad i = 1, 2, \dots, \mathbf{np}; j = 1, 2, \dots, \mathbf{neq} \quad (5)$$

Constraint: **tol** > 0.0.

13: **abt[**neq**]** – double *Output*

On exit: **abt**[$i - 1$], for $i = 1, 2, \dots, \mathbf{neq}$, holds the largest estimated error (in magnitude) of the i th component of the solution over all mesh points.

14: **jacobj** – function, supplied by the user *External Function*

jacobj must evaluate the Jacobian $\left(\frac{\partial f_i}{\partial y_j}\right)$ for $i, j = 1, 2, \dots, \mathbf{neq}$, given x and y_j , for $j = 1, 2, \dots, \mathbf{neq}$.

The specification of **jacobj** is:

```
void jacobj (Integer neq, double x, double eps, const double y[], double f[],
            Nag_User *comm)
```

1: **neq** – Integer *Input*

On entry: the number of equations.

2: **x** – double *Input*

On entry: the value of the argument x .

3: **eps** – double *Input*

On entry: the value of the continuation argument, ϵ . This is 1 if continuation is not being used.

4:	y[neq] – const double <i>On entry:</i> y [<i>i</i> – 1] contains the value of the argument y_i , for $i = 1, 2, \dots, \mathbf{neq}$.	<i>Input</i>
5:	f[neq × neq] – double <i>On exit:</i> f [(<i>i</i> – 1) × neq + (<i>j</i> – 1)] must be set to the value of $\frac{\partial f_i}{\partial y_j}$, evaluated at the point (x, y), for $i, j = 1, 2, \dots, \mathbf{neq}$.	<i>Output</i>
6:	comm – Nag_User * Pointer to a structure of type Nag_User with the following member: p – Pointer <i>On entry/exit:</i> the pointer comm → p should be cast to the required type, e.g., <code>struct user *s = (struct user *)comm → p</code> , to obtain the original object's address with appropriate type. (See the argument comm below.)	

Note that if **jacobjf** is supplied then **jacobjg** (see below) must also be supplied. Note that if **jacobjf** is supplied and continuation is requested then **jacobjps** and **jacobjep** (see below) must also be supplied.

15: **jacobjg** – function, supplied by the user *External Function*

jacobjg must evaluate the Jacobians $\left(\frac{\partial g_i}{\partial y_j(a)}\right)$ and $\left(\frac{\partial g_i}{\partial y_j(b)}\right)$. The ordering of the rows of **aj** and **bj** must correspond to the ordering of the boundary conditions described in the specification of function **g** above.

The specification of jacobjg is:		
void jacobjg (Integer neq, double eps, const double ya[], const double yb[], double aj[], double bj[], Nag_User *comm)		
1:	neq – Integer <i>On entry:</i> the number of equations.	<i>Input</i>
2:	eps – double <i>On entry:</i> the value of the continuation argument, ϵ . This is 1 if continuation is not being used.	<i>Input</i>
3:	ya[neq] – const double <i>On entry:</i> ya [<i>i</i> – 1] contains the value $y_i(a)$, for $i = 1, 2, \dots, \mathbf{neq}$.	<i>Input</i>
4:	yb[neq] – const double <i>On entry:</i> yb [<i>i</i> – 1] contains the value $y_i(b)$, for $i = 1, 2, \dots, \mathbf{neq}$.	<i>Input</i>
5:	aj[neq × neq] – double <i>On exit:</i> aj [(<i>i</i> – 1) × neq + (<i>j</i> – 1)] must be set to the value $\frac{\partial g_i}{\partial y_j(a)}$, for $i, j = 1, 2, \dots, \mathbf{neq}$.	<i>Output</i>
6:	bj[neq × neq] – double <i>On exit:</i> bj [(<i>i</i> – 1) × neq + (<i>j</i> – 1)] must be set to the value $\frac{\partial g_i}{\partial y_j(b)}$, for $i, j = 1, 2, \dots, \mathbf{neq}$.	<i>Output</i>

7: **comm** – Nag_User *

Pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/exit: the pointer **comm** → **p** should be cast to the required type, e.g.,
 struct user *s = (struct user *)comm → p, to obtain the original object's
 address with appropriate type. (See the argument **comm** below.)

Note that if **jacobj** is supplied then **jacobjf** (see above) must also be supplied. If numerical differentiation is to be used to calculate the Jacobian then **jacobjf** and **jacobjg** may be replaced by the NAG-defined two null functions macro, **NULL_2_FUN**.

16: **jaceps** – function, supplied by the user *External Function*

jaceps must evaluate the derivative $\frac{\partial f_i}{\partial \epsilon}$ given x, y and ϵ if continuation is being used.

The specification of **jaceps** is:

```
void jaceps (Integer neq, double x, double eps, const double y[], double f[],
            Nag_User *comm)
```

1: **neq** – Integer *Input*

On entry: the number of equations.

2: **x** – double *Input*

On entry: the value of the argument x .

3: **eps** – double *Input*

On entry: the value of the continuation argument, ϵ .

4: **y[neq]** – const double *Input*

On entry: **y**[$i - 1$] contains the solution values y_i at the point x , for $i = 1, 2, \dots, \mathbf{neq}$.

5: **f[neq]** – double *Output*

On exit: **f**[$i - 1$] must contain $f(i)$, the value $\frac{\partial f_i}{\partial \epsilon}$ at the point (x, y) , given ϵ , for $i = 1, 2, \dots, \mathbf{neq}$.

6: **comm** – Nag_User *

Pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/exit: the pointer **comm** → **p** should be cast to the required type, e.g.,
 struct user *s = (struct user *)comm → p, to obtain the original object's
 address with appropriate type. (See the argument **comm** below.)

Note that if **jaceps** is defined then **jacgep** (see below) must also be defined.

17: **jacgep** – function, supplied by the user *External Function*

jacgep must evaluate the derivatives $\frac{\partial g_i}{\partial \epsilon}$ if continuation is being used.

The specification of **jacgep** is:

```
void jacgep (Integer neq, double eps, const double ya[], const double yb[],
            double bcep[], Nag_User *comm)
```

- | | | |
|----|--|---------------|
| 1: | neq – Integer | <i>Input</i> |
| | <i>On entry:</i> the number of equations. | |
| 2: | eps – double | <i>Input</i> |
| | <i>On entry:</i> the value of the continuation argument, ϵ . | |
| 3: | ya[neq] – const double | <i>Input</i> |
| | <i>On entry:</i> ya [$i - 1$] contains the value of $y_i(a)$, for $i = 1, 2, \dots, \mathbf{neq}$. | |
| 4: | yb[neq] – const double | <i>Input</i> |
| | <i>On entry:</i> yb [$i - 1$] contains the value of $y_i(b)$, for $i = 1, 2, \dots, \mathbf{neq}$. | |
| 5: | bcep[neq] – double | <i>Output</i> |
| | <i>On exit:</i> bcep [$i - 1$] must contain the value of $\frac{\partial g_i}{\partial \epsilon}$, for $i = 1, 2, \dots, \mathbf{neq}$. | |
| 6: | comm – Nag_User * | |
| | Pointer to a structure of type Nag_User with the following member: | |
| | p – Pointer | |
| | <i>On entry/exit:</i> the pointer comm \rightarrow p should be cast to the required type, e.g.,
struct user *s = (struct user *)comm \rightarrow p, to obtain the original object's
address with appropriate type. (See the argument comm below.) | |

Note that if **jacgep** is defined then **jaceps** (see above) must also be defined. If numerical differentiation is to be used to calculate the Jacobian and continuation is not required then **jacobj**, **jacobjg**, **jaceps** and **jacgep** may be replaced by the NAG-defined four null functions macro, **NULL_2_FUN**.

- 18: **comm** – Nag_User *

Pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/exit: the pointer **p**, of type Pointer, allows you to communicate information to and from **fcn**, **g**, **jacobj**, **jacobjg**, **jaceps**, and **jacgep**. An object of the required type should be declared, e.g., a structure, and its address assigned to the pointer **p** by means of a cast to Pointer in the calling program, e.g., **comm.p** = (Pointer)&s. The type pointer will be void * with a C compiler that defines void * and char * otherwise.

- 19: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_2_INT_ARG_ZERO

On entry, **numbeg** = 0 and **nummix** = 0. These arguments must not both be zero.

NE_2_REAL_ARG_LE

On entry, $\mathbf{x}[\mathbf{np} - 1] = \langle value \rangle$ while $\mathbf{x}[0] = \langle value \rangle$. These arguments must satisfy $\mathbf{x}[\mathbf{np} - 1] > \mathbf{x}[0]$

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument **init** had an illegal value.

NE_CONV_CONT

Convergence failure. There are a number of possible causes.

- (a) Faulty coding of the Jacobian calculation functions.
- (b) If Jacobians have not been supplied then inaccurate Jacobians have been calculated internally (not very likely).
- (c) A poor choice of initial mesh or initial starting conditions either by you or by default. Try using the continuation facility.

NE_CONV_CONT_DELEPS

deleps is required to be less than machine precision for continuation to proceed. It is likely that either the problem has no solution for some value near the current value of ϵ or that the problem is so difficult that even with continuation it is unlikely to be solved using this function. Using more mesh points may help.

NE_CONV_CONT_DEP

There is no dependence on epsilon when continuation is being used. This may be due to faulty coding of **jaceps** or **jacgep**, or in some circumstances, to a zero initial choice of approximate solution (such as is chosen when **init** = Nag_DefInItMesh).

NE_CONV_JACOBG

The Jacobian calculated by **jacobjg** (or the equivalent matrix calculated by numerical differentiation) is singular. This may be due to faulty coding of **jacobjg** or in some circumstances, to a zero initial choice of approximate solution (such as is chosen when **init** = Nag_DefInItMesh).

NE_CONV_MESH

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

NE_CONV_ROUNDOFF

Solution cannot be improved due to roundoff error. Too much accuracy might have been requested.

NE_INT_ARG_LT

On entry, **mnp** = $\langle value \rangle$.
Constraint: **mnp** \geq 32.

On entry, **neq** = $\langle value \rangle$.
Constraint: **neq** \geq 1.

NE_INT_RANGE_CONS

On entry, **np** = $\langle value \rangle$ and **mnp** = $\langle value \rangle$. The argument **np** must satisfy $4 \leq \mathbf{np} \leq \mathbf{mnp}$.

On entry, **numbeg** = $\langle value \rangle$ and **neq** = $\langle value \rangle$. The argument **numbeg** must satisfy $0 \leq \mathbf{numbeg} \leq \mathbf{neq}$.

On entry, **nummix** = $\langle value \rangle$ and **neq** – **numbeg** = $\langle value \rangle$. The argument **nummix** must satisfy $0 \leq \mathbf{nummix} \leq \mathbf{neq} - \mathbf{numbeg}$.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

NE_INVALID_FUN_JAC

Only one of **jacobjf** or **jacobjg** has been set to non-null possibly implying user-defined Jacobians. Both must be non-null.

NE_INVALID_FUN_JAC_CONT

deleps has been set to $\langle value \rangle$ implying continuation and both **jacobjf** and **jacobjg** have been set to non-null implying user-defined Jacobians. Hence the functions **jaceps** and **jacgep** must also be non-null.

NE_INVALID_FUN_JAC_NO_CONT

deleps has been set to $\langle value \rangle$ implying no continuation and both **jacobjf** and **jacobjg** have been set to non-null implying user-defined Jacobians. Hence the functions **jaceps** and **jacgep** must be NULL.

NE_INVALID_FUN_NO_JAC_CONT

deleps has been set to $\langle value \rangle$ implying continuation and both **jacobjf** and **jacobjg** have been set to NULL implying no user-defined Jacobians. Hence the functions **jaceps** and **jacgep** must also be NULL.

NE_NOT_STRICTLY_INCREASING

The sequence **x** is not strictly increasing: $\mathbf{x}[\langle value \rangle] = \langle value \rangle$, $\mathbf{x}[\langle value \rangle] = \langle value \rangle$.

NE_REAL_ARG_LE

On entry, **tol** must not be less than or equal to 0.0: **tol** = $\langle value \rangle$.

7 Accuracy

The solution returned by the function will be accurate to your tolerance as defined by the relation (5) except in extreme circumstances. The final error estimate over the whole mesh for each component is given in the array **abt**. 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

There are too many factors present to quantify the timing. The time taken by the function is negligible only on very simple problems.

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

9 Example

We solve the differential equation

$$y''' = -yy'' - 2\epsilon(1 - y^2)$$

with $\epsilon = 1$ and boundary conditions

$$y(0) = y'(0) = 0, y'(10) = 1$$

to an accuracy specified by $\text{tol} = 1.0\text{e-}4$. The continuation facility is used with the continuation argument ϵ introduced as in the differential equation above and with $\text{deleps} = 0.1$ initially. (The continuation facility is not needed for this problem and is used here for illustration.)

9.1 Program Text

```

/* nag_ode_bvp_fd_nonlin_gen (d02rac) Example Program.
 *
 * Copyright 1992 Numerical Algorithms Group.
 *
 * Mark 3, 1992.
 * Mark 7 revised, 2001.
 * Mark 8 revised, 2004.
 *
 */

#include <nag.h>
#include <nagx04.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagd02.h>

#ifdef __cplusplus
extern "C" {
#endif
static void NAG_CALL fcn(Integer neq, double x, double eps, double y[],
                        double f[], Nag_User *comm);
static void NAG_CALL g(Integer neq, double eps, double ya[], double yb[],
                      double bc[], Nag_User *comm);
static void NAG_CALL jaceps(Integer neq, double x, double eps, double y[],
                          double f[], Nag_User *comm);
static void NAG_CALL jacgep(Integer neq, double eps, double ya[], double yb[],
                          double bcep[], Nag_User *comm);
static void NAG_CALL jacobf(Integer neq, double x, double eps, double y[],
                          double f[], Nag_User *comm);
static void NAG_CALL jacobg(Integer neq, double eps, double ya[], double yb[],
                          double aj[], double bj[], Nag_User *comm);
#ifdef __cplusplus
}
#endif

#define NEQ 3
#define MNP 40

#define Y(I, J) y[(I) *tdy + J]
int main(int argc, char *argv[])
{
    FILE      *fpout;

    double    *abt = 0;
    double    deleps;
    double    tol;
    double    *x = 0, *y = 0;
    Integer    exit_status = 0;
    Integer    i, j;
    Integer    np;
    Integer    numbeg, nummix;
    Integer    neq, mnp, tdy;
    Nag_User  comm;
    NagError  fail;

    INIT_FAIL(fail);

    /* Check for command-line IO options */
    fpout = nag_example_file_io(argc, argv, "-results", NULL);

    fprintf(fpout,
           "nag_ode_bvp_fd_nonlin_gen (d02rac) Example Program Results\n");
    fprintf(fpout, "\nCalculation using analytic Jacobians\n\n");

```

```

neq = NEQ;
mnp = MNP;
if (neq >= 1)
{
    if (!(abt = NAG_ALLOC(neq, double)) ||
        !(x = NAG_ALLOC(mnp, double)) ||
        !(y = NAG_ALLOC(neq*mnp, double)))
    {
        fprintf(fpout, "Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    tdy = mnp;
}
else
{
    exit_status = 1;
    return exit_status;
}
tol = 1.0e-4;
np = 17;
numbeg = 2;
nummix = 0;
x[0] = 0.0;
x[np-1] = 10.0;
deleps = 0.1;

/* nag_ode_bvp_fd_nonlin_gen (d02rac).
 * Ordinary differential equations solver, for general
 * nonlinear two-point boundary value problems, using a
 * finite difference technique with deferred correction
 */
nag_ode_bvp_fd_nonlin_gen(neq, &deleps, fcn, numbeg, nummix, g,
                          Nag_DefInitMesh, mnp, &np, x, y, tol, abt, jacobf,
                          jacobg, jaceps, jacgep, &comm, &fail);
if (fail.code != NE_NOERROR)
{
    fprintf(fpout, "Error from nag_ode_bvp_fd_nonlin_gen (d02rac).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
}

fprintf(fpout, "Solution on final mesh of %ld points \n", np);
fprintf(fpout, "      X          Y(1)          Y(2)          Y(3)\n");

for (j = 0; j < np; ++j)
{
    fprintf(fpout, " %9.3f ", x[j]);
    for (i = 0; i < neq; ++i)
        fprintf(fpout, " %9.4f", Y(i, j));
    fprintf(fpout, "\n");
}

fprintf(fpout, "\n\nMaximum estimated error by components \n");

for (i = 1; i <= 3; ++i)
    fprintf(fpout, " %11.2e", abt[i-1]);
fprintf(fpout, " \n");

END:
if (fpout != stdout) fclose(fpout);
if (abt) NAG_FREE(abt);
if (x) NAG_FREE(x);
if (y) NAG_FREE(y);
return exit_status;
}
#undef Y

static void NAG_CALL fcn(Integer neq, double x, double eps, double y[],
                        double f[], Nag_User *comm)

```

```

{
  f[0] = y[1];
  f[1] = y[2];
  f[2] = -y[0] * y[2] - (1.0 - y[1]*y[1])*2.0*eps;
}

static void NAG_CALL g(Integer neq, double eps, double ya[], double yb[],
                      double bc[], Nag_User *comm)
{
  bc[0] = ya[0];
  bc[1] = ya[1];
  bc[2] = yb[1] - 1.0;
}                                     /* g */

static void NAG_CALL jaceps(Integer neq, double x, double eps, double y[],
                           double f[], Nag_User *comm)
{
  f[0] = 0.0;
  f[1] = 0.0;
  f[2] = (1.0 - y[1]*y[1]) * -2.0;
}

static void NAG_CALL jacgep(Integer neq, double eps, double ya[], double
                           yb[], double bcep[], Nag_User *comm)
{
  Integer i;

  for (i = 0; i < neq; ++i)
    bcep[i] = 0.0;
}

static void NAG_CALL jacobf(Integer neq, double x, double eps, double y[],
                           double f[], Nag_User *comm)
{
  Integer i, j;

#define Y(I)    y[(I) -1]
#define F(I, J) f[((I) -1)*neq+(J) -1]

  for (i = 1; i <= neq; ++i)
    {
      for (j = 1; j <= neq; ++j)
        F(i, j) = 0.0;
    }
  F(1, 2) = 1.0;
  F(2, 3) = 1.0;
  F(3, 1) = -Y(3);
  F(3, 2) = Y(2) * 4.0 * eps;
  F(3, 3) = -Y(1);
}

static void NAG_CALL jacobg(Integer neq, double eps, double ya[],
                           double yb[], double aj[], double bj[],
                           Nag_User *comm)
{
  Integer i, j;
#define YA(I)    ya[(I) -1]
#define YB(I)    yb[(I) -1]
#define AJ(I, J) aj[((I) -1)*neq+(J) -1]
#define BJ(I, J) bj[((I) -1)*neq+(J) -1]

  for (i = 1; i <= neq; ++i)
    for (j = 1; j <= neq; ++j)
      {
        AJ(i, j) = 0.0;
        BJ(i, j) = 0.0;
      }
}

```

```

    }
    AJ(1, 1) = 1.0;
    AJ(2, 2) = 1.0;
    BJ(3, 2) = 1.0;
}

```

9.2 Program Data

None.

9.3 Program Results

nag_ode_bvp_fd_nonlin_gen (d02rac) Example Program Results

Calculation using analytic Jacobians

Solution on final mesh of 33 points

X	Y(1)	Y(2)	Y(3)
0.000	0.0000	0.0000	1.6872
0.062	0.0032	0.1016	1.5626
0.125	0.0125	0.1954	1.4398
0.188	0.0275	0.2816	1.3203
0.250	0.0476	0.3605	1.2054
0.375	0.1015	0.4976	0.9924
0.500	0.1709	0.6097	0.8048
0.625	0.2530	0.6999	0.6438
0.703	0.3095	0.7467	0.5563
0.781	0.3695	0.7871	0.4784
0.938	0.4978	0.8513	0.3490
1.094	0.6346	0.8977	0.2502
1.250	0.7776	0.9308	0.1763
1.458	0.9748	0.9598	0.1077
1.667	1.1768	0.9773	0.0639
1.875	1.3815	0.9876	0.0367
2.031	1.5362	0.9922	0.0238
2.188	1.6915	0.9952	0.0151
2.500	2.0031	0.9983	0.0058
2.656	2.1591	0.9990	0.0035
2.812	2.3153	0.9994	0.0021
3.125	2.6277	0.9998	0.0007
3.750	3.2526	1.0000	0.0001
4.375	3.8776	1.0000	0.0000
5.000	4.5026	1.0000	0.0000
5.625	5.1276	1.0000	-0.0000
6.250	5.7526	1.0000	0.0000
6.875	6.3776	1.0000	-0.0000
7.500	7.0026	1.0000	0.0000
8.125	7.6276	1.0000	-0.0000
8.750	8.2526	1.0000	0.0000
9.375	8.8776	1.0000	-0.0000
10.000	9.5026	1.0000	0.0000

Maximum estimated error by components

6.92e-05	1.81e-05	6.42e-05
----------	----------	----------
