

## nag\_ode\_bvp\_fd\_nonlin\_fixedbc (d02gac)

### 1. Purpose

**nag\_ode\_bvp\_fd\_nonlin\_fixedbc (d02gac)** 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. Specification

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

void nag_ode_bvp_fd_nonlin_fixedbc(Integer neq,
    void (*fcn)(Integer neq, double x, double y[], double f[],
        Nag_User *comm),
    double a, double b, double u[], Integer v[],
    Integer mnp, Integer *np, double x[], double y[],
    double tol, Nag_User *comm, NagError *fail)
```

### 3. Description

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

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

and the derivatives are evaluated by a function **fcn** supplied by the user. Initially, **neq** boundary values of the variables  $y_i$  must be specified (assigned), some at  $a$  and some at  $b$ . The user also supplies estimates of the remaining **neq** 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  $\frac{\partial f_i}{\partial y_j}$  and this is calculated by numerical differentiation using an algorithm described in Curtis *et al* (1974).

The user supplies 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 the user's 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 the user should use the interpolation routines provided in Chapter E01 if these points do not themselves form a convenient mesh.

### 4. Parameters

#### neq

Input: the number of equations.

Constraint: **neq**  $\geq 2$ .

#### fcn

The function **fcn** must evaluate the functions  $f_i$  (i.e., the derivatives  $y'_i$ ) at the general point  $x$ .

The specification of **fcn** is:

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

**neq**  
Input: the number of differential equations.

**x**  
Input: the value of the argument  $x$ .

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

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

**comm**  
Input/Output: pointer to a structure of type Nag\_User with the following member:

**p** - Pointer  
Input/Output: 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.)

**a**Input: the left-hand boundary point,  $a$ .**b**Input: the right-hand boundary point,  $b$ .  
Constraint: **b** > **a**.**u[neq][2]**Input: **u**[ $i - 1$ ][0] must be set to the known (assigned) or estimated values of  $y_i$  at  $a$  and **u**[ $i - 1$ ][1] must be set to the known or estimated values of  $y_i$  at  $b$ , for  $i = 1, 2, \dots, \mathbf{neq}$ .**v[neq][2]**Input: **v**[ $i - 1$ ][ $j - 1$ ] must be set to 0 if **u**[ $i - 1$ ][ $j - 1$ ] is a known (assigned) value and to 1 if **u**[ $i - 1$ ][ $j - 1$ ] is an estimated value,  $i = 1, 2, \dots, \mathbf{neq}$ ;  $j = 1, 2$ .Constraint: precisely **neq** of the **v**[ $i - 1$ ][ $j - 1$ ] must be set to 0 i.e., precisely **neq** of **u**[ $i - 1$ ][0] and **u**[ $i - 1$ ][1] must be known values and these must not be all at  $a$  or  $b$ .**mnp**Input: the maximum permitted number of mesh points.  
Constraint: **mnp**  $\geq$  32.**np**Input: determines whether a default or user-supplied initial mesh is used. If **np** = 0, then **np** is set to a default value of 4 and a corresponding equispaced mesh **x**[0], **x**[1], ..., **x**[**np** - 1] is used. If **np**  $\geq$  4, then the user must define an initial mesh using the array **x** as described.Constraint: **np** = 0 or  $4 \leq \mathbf{np} \leq \mathbf{mnp}$ .

Output: the number of points in the final (returned) mesh.

**x[mnp]**Input: if **np**  $\geq$  4 (see **np** above), the first **np** elements must define an initial mesh. Otherwise the elements of **x** need not be set.

Constraint:

$$\mathbf{a} = \mathbf{x}[0] < \mathbf{x}[1] < \dots < \mathbf{x}[\mathbf{np} - 1] = \mathbf{b} \text{ for } \mathbf{np} \geq 4 \quad (2)$$

Output: **x**[0], **x**[1], ..., **x**[**np**-1] define the final mesh (with the returned value of **np**) satisfying the relation (2).**y[neq][mnp]**Output: the approximate solution  $z_j(x_i)$  satisfying (3), on the final mesh, that is

$$\mathbf{y}[j - 1][i - 1] = z_j(x_i), \quad i = 1, 2, \dots, \mathbf{np}; j = 1, 2, \dots, \mathbf{neq},$$

where **np** is the number of points in the final mesh.The remaining columns of **y** are not used.

**tol**

Input: 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  $y_j(x_i)$  is the  $j$ th component of the true solution of equation (1) (see Section 3) and the boundary conditions, then, except in extreme cases, it is expected that

$$|z_j(x_i) - y_j(x_i)| \leq \mathbf{tol}, \quad i = 1, 2, \dots, \mathbf{np}; j = 1, 2, \dots, \mathbf{neq} \quad (3)$$

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

**comm**

Input/Output: pointer to a structure of type Nag\_User with the following member:

**p** - Pointer

Input/Output: The pointer **p**, of type Pointer, allows the user to communicate information to and from the user-defined function **fcn()**. An object of the required type should be declared by the user, 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.

**fail**

The NAG error parameter, see the Essential Introduction to the NAG C Library.

**5. Error Indications and Warnings****NE\_INT\_ARG\_LT**

On entry, **neq** must not be less than 2: **neq** =  $\langle value \rangle$ .

On entry, **mnp** must not be less than 32: **mnp** =  $\langle value \rangle$ .

**NE\_REAL\_ARG\_LE**

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

**NE\_2\_REAL\_ARG\_LE**

On entry **b** =  $\langle value \rangle$  while **a** =  $\langle value \rangle$ . These parameters must satisfy **b** > **a**.

**NE\_INT\_RANGE\_CONS\_2**

On entry **np** =  $\langle value \rangle$  and **mnp** =  $\langle value \rangle$ . The parameter **np** must satisfy either  $4 \leq \mathbf{np} \leq \mathbf{mnp}$  or **np** = 0.

**NE\_LF\_B\_VAL**

The number of known left boundary values must be less than the number of equations: The number of known left boundary values =  $\langle value \rangle$ : The number of equations =  $\langle value \rangle$ .

**NE\_RT\_B\_VAL**

The number of known right boundary values must be less than the number of equations: The number of known right boundary values =  $\langle value \rangle$ : The number of equations =  $\langle value \rangle$ .

**NE\_LFRT\_B\_VAL**

The sum of known left and right boundary values must equal the number of equations: The number of known left boundary values =  $\langle value \rangle$ : The number of known right boundary values =  $\langle value \rangle$ : The number of equations =  $\langle value \rangle$ .

**NE\_LF\_B\_MESH**

On entry, the left boundary value **a**, has not been set to **x**[0]: **a** =  $\langle value \rangle$ , **x**[0] =  $\langle value \rangle$ .

**NE\_RT\_B\_MESH**

On entry, the right boundary value **b**, has not been set to **x** [**np**-1]: **b** =  $\langle value \rangle$ , **x**[**np**-1] =  $\langle value \rangle$ .

**NE\_NOT\_STRICTLY\_INCREASING**

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

**NE\_ALLOC\_FAIL**

Memory allocation failed.

**NE\_CONV\_MESH**

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

**NE\_CONV\_MESH\_INIT**

The Newton iteration failed to converge on the initial mesh. This may be due to the initial mesh having too few points or the initial approximate solution being too inaccurate. Try using nag\_ode\_bvp\_fd\_nonlin\_gen (d02rac).

**NE\_CONV\_ROUNDOFF**

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

**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 consult NAG for assistance.

**6. Further Comments**

The time taken by the function 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.

A common cause of convergence problems in the Newton iteration is the user specifying too few points in the initial mesh. Although the routine 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 the known and estimated boundary values are set to zero, the routine 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.

The user 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 nag\_ode\_bvp\_fd\_nonlin\_gen (d02rac) is recommended.

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

**6.1. Accuracy**

The solution returned by the routine will be accurate to the user's 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.

**6.2. 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. In: '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.

**7. See Also**

nag\_ode\_bvp\_fd\_lin\_gen (d02gbc)

nag\_ode\_bvp\_fd\_nonlin\_gen (d02rac)

## 8. Example

We solve the differential equation

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

with boundary conditions

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

for  $\beta = 0.0$  and  $\beta = 0.2$  to an accuracy specified by `tol = 1.0e-3`. We solve first the simpler problem with  $\beta = 0.0$  using an equispaced mesh of 26 points and then we solve the problem with  $\beta = 0.2$  using the final mesh from the first problem.

### 8.1. Program Text

```

/* nag_ode_bvp_fd_nonlin_fixedbc(d02gac) Example Program
 *
 * Copyright 1994 Numerical Algorithms Group.
 *
 * Mark 3, 1994.
 *
 */

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

#ifdef NAG_PROTO
static void fcn(Integer neq, double x, double y[], double f[],
                Nag_User *comm);
#else
static void fcn();
#endif

#define NEQ 3
#define MNP 40

main()
{
    double a, b;
    Integer i, j, k;
    double u[NEQ][2], x[MNP], y[NEQ][MNP];
    Integer v[NEQ][2];
    Integer np;
    double tol;
    static NagError fail;
    Integer neq, mnp;
    double beta;
    Nag_User comm;

    Vprintf("d02gac Example Program Results\n");

    /* For communication with function fcn()
     * assign address of beta to comm.p.
     */
    comm.p = (Pointer)&beta;
    neq = NEQ;
    mnp = MNP;
    tol = 0.001;
    np = 26;
    a = 0.0;
    b = 10.0;
    beta = 0.0;
    for (i=0; i<neq; ++i)
        for (j=0; j<2; ++j)
            {
                u[i][j] = 0.0;
            }

```

```

        v[i][j] = 0;
    }

    v[2][0] = 1;
    v[0][1] = 1;
    v[2][1] = 1;
    u[1][1] = 1.0;
    u[0][1] = b;
    x[0] = a;
    for (i=2; i<=np-1; ++i)
        x[i-1] = ((double)(np-i)*a + (double)(i-1)*b)/
            (double)(np-1);
    x[np-1] = b;
    for (k=1; k<=2; ++k)
    {
        Vprintf("\nProblem with beta = %7.4f\n", beta);
        d02gac(neq, fcn, a, b, (double *)u, (Integer *)v, mnp,
            &np, (double *)x, (double *)y, tol, &comm, &fail);

        if (fail.code == NE_NOERROR || fail.code == NE_CONV_ROUNDOff)
        {
            Vprintf ("\nSolution on final mesh of %ld points\n", np);
            Vprintf ("      X          Y(1)          Y(2)          Y(3)\n");
            for (i=0; i<=np-1; ++i)
            {
                Vprintf (" %9.4f ", x[i]);
                for (j=0; j<neq; ++j)
                    Vprintf (" %9.4f ", y[j][i]);
                Vprintf("\n");
            }
            beta += 0.2;
        }
    }
    exit(EXIT_SUCCESS);
}

#ifdef NAG_PROTO
static void fcn(Integer neq, double x, double y[], double f[], Nag_User *comm )
#else
static void fcn(neq, x, y, f, comm)
Integer neq;
double x;
double y[], f[];
Nag_User *comm;
#endif
{
    double *beta = (double *)comm->p;

    f[0] = y[1];
    f[1] = y[2];
    f[2] = -y[0] * y[2] - *beta * (1.0-y[1]*y[1]);
}

```

## 8.2. Program Data

None.

## 8.3. Program Results

d02gac Example Program Results

Problem with beta = 0.0000

Solution on final mesh of 26 points

X	Y(1)	Y(2)	Y(3)
0.0000	0.0000	0.0000	0.4695
0.4000	0.0375	0.1876	0.4673
0.8000	0.1497	0.3719	0.4511
1.2000	0.3336	0.5450	0.4104
1.6000	0.5828	0.6963	0.3424

2.0000	0.8864	0.8163	0.2558
2.4000	1.2309	0.9009	0.1678
2.8000	1.6026	0.9529	0.0953
3.2000	1.9900	0.9805	0.0464
3.6000	2.3851	0.9930	0.0193
4.0000	2.7834	0.9978	0.0069
4.4000	3.1829	0.9994	0.0021
4.8000	3.5828	0.9999	0.0006
5.2000	3.9828	1.0000	0.0001
5.6000	4.3828	1.0000	0.0000
6.0000	4.7828	1.0000	0.0000
6.4000	5.1828	1.0000	0.0000
6.8000	5.5828	1.0000	0.0000
7.2000	5.9828	1.0000	0.0000
7.6000	6.3828	1.0000	0.0000
8.0000	6.7828	1.0000	0.0000
8.4000	7.1828	1.0000	0.0000
8.8000	7.5828	1.0000	0.0000
9.2000	7.9828	1.0000	0.0000
9.6000	8.3828	1.0000	0.0000
10.0000	8.7828	1.0000	0.0000

Problem with beta = 0.2000

Solution on final mesh of 26 points

X	Y(1)	Y(2)	Y(3)
0.0000	0.0000	0.0000	0.6865
0.4000	0.0528	0.2584	0.6040
0.8000	0.2020	0.4814	0.5091
1.2000	0.4324	0.6636	0.4001
1.6000	0.7268	0.8007	0.2860
2.0000	1.0670	0.8939	0.1821
2.4000	1.4368	0.9498	0.1017
2.8000	1.8233	0.9791	0.0492
3.2000	2.2180	0.9924	0.0206
3.6000	2.6162	0.9976	0.0074
4.0000	3.0157	0.9993	0.0023
4.4000	3.4156	0.9998	0.0006
4.8000	3.8155	1.0000	0.0001
5.2000	4.2155	1.0000	0.0000
5.6000	4.6155	1.0000	0.0000
6.0000	5.0155	1.0000	0.0000
6.4000	5.4155	1.0000	0.0000
6.8000	5.8155	1.0000	0.0000
7.2000	6.2155	1.0000	0.0000
7.6000	6.6155	1.0000	0.0000
8.0000	7.0155	1.0000	0.0000
8.4000	7.4155	1.0000	0.0000
8.8000	7.8155	1.0000	0.0000
9.2000	8.2155	1.0000	0.0000
9.6000	8.6155	1.0000	0.0000
10.0000	9.0155	1.0000	0.0000

---