NAG Library Function Document

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, const double y[], double f[],
          Nag_User *comm),
     double a, double b, const double u[], const Integer v[], Integer mnp,
     Integer *np, double x[], double y[], double tol, Nag_User *comm,
     NagError *fail)
```

3 Description

nag_ode_bvp_fd_nonlin_fixedbc (d02gac) 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

$$\mathbf{y}_i' = f_i\left(x, y_1, y_2, \ldots, y_{\mathbf{neq}}\right), i = 1, 2, \ldots, \mathbf{neq} \tag{1}$$

and the derivatives are evaluated by **fcn**. Initially, **neq** boundary values of the variables y_i must be specified (assigned), some at a and some at b. You also need to supply 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 $\dfrac{\partial f_i}{\partial y_j}$ and this is calculated by numerical differentiation using an algorithm described in Curtis *et al.* (1974).

You need to 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* (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 equations.

Constraint: **neq** ≥ 2.

2: **fcn** – function, supplied by the user *External 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, const double y[], double f[],
     Nag_User *comm)
```

1: **neq** – Integer *Input*

On entry: the number of differential equations.

2: **x** – double *Input*

On entry: the value of the argument x.

3: **y**[**neq**] – const double *Input*

On entry: $y[i-1]$ holds the value of the argument y_i, for $i = 1, 2, \ldots,$ **neq**.

4: **f**[**neq**] – double *Output*

On exit: $f[i-1]$ must contain the values of f_i, for $i = 1, 2, \ldots,$ **neq**.

5: **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.)

3: **a** – double *Input*

On entry: the left-hand boundary point, a.

4: **b** – double *Input*

On entry: the right-hand boundary point, b.

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

5: **u**[**neq** $\times$ **2**] – const double *Input*

On entry: **u**$[(i-1)\times 2]$ must be set to the known (assigned) or estimated values of y_i at a and **u**$[(i-1)\times 2+1]$ must be set to the known or estimated values of y_i at b, for $i = 1, 2, \ldots,$ **neq**.

6: **v**[**neq** $\times$ **2**] – const Integer *Input*

On entry: **v**$[(i-1)\times 2+j-1]$ must be set to 0 if **u**$[(i-1)\times 2+j-1]$ is a known (assigned) value and to 1 if **u**$[(i-1)\times 2+j-1]$ is an estimated value, $i = 1, 2, \ldots,$ **neq** and $j = 1, 2$.

Constraint: precisely **neq** of the **v**$[(i-1)\times 2+j-1]$ must be set to 0, i.e., precisely **neq** of **u**$[(i-1)\times 2]$ and **u**$[(i-1)\times 2+1]$ must be known values and these must not be all at a or b.

7: **mnp** – Integer *Input*

On entry: the maximum permitted number of mesh points.

Constraint: **mnp** ≥ 32.

8: **np** – Integer * *Input/Output*

On entry: determines whether a default or user-supplied initial mesh is used.

np $= 0$
: **np** is set to a default value of 4 and a corresponding equispaced mesh $\mathbf{x}[0], \mathbf{x}[1], \ldots, \mathbf{x}[\mathbf{np}-1]$ is used.

np ≥ 4
: You must define an initial mesh using the array **x** as described.

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

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

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

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

Constraint:

$$\mathbf{a} = \mathbf{x}[0] < \mathbf{x}[1] < \cdots < \mathbf{x}[\mathbf{np}-1] = \mathbf{b} \text{ for } \mathbf{np} \geq 4 \tag{2}$$

On exit: $\mathbf{x}[0], \mathbf{x}[1], \ldots, \mathbf{x}[\mathbf{np}-1]$ define the final mesh (with the returned value of **np**) satisfying the relation (2).

10: **y**[**neq** $\times$ **mnp**] – double *Output*

On exit: the approximate solution $z_j(x_i)$ satisfying (3), on the final mesh, that is

$$\mathbf{y}[(j-1) \times \mathbf{mnp} + i - 1] = z_j(x_i), i = 1, 2, \ldots, \mathbf{np};\ j = 1, 2, \ldots, \mathbf{neq},$$

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

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

11: **tol** – double *Input*

On entry: 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 $y_j(x_i)$ 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

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

Constraint: **tol** > 0.0.

12: **comm** – Nag_User *

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

p – Pointer

On entry/exit: the pointer **comm**→**p**, of type Pointer, allows you to communicate information to and from **fcn**. An object of the required type should be declared, e.g., a structure, and its address assigned to the pointer **comm**→**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.

13: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_2_REAL_ARG_LE

On entry, **b** = ⟨*value*⟩ while **a** = ⟨*value*⟩. These arguments must satisfy **b** > **a**.

NE_ALLOC_FAIL

Dynamic 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_INT_ARG_LT

On entry, **mnp** = ⟨*value*⟩.
Constraint: **mnp** ≥ 32.

On entry, **neq** = ⟨*value*⟩.
Constraint: **neq** ≥ 2.

NE_INT_RANGE_CONS_2

On entry, **np** = ⟨*value*⟩ and **mnp** = ⟨*value*⟩. The argument **np** must satisfy either 4 ≤ **np** ≤ **mnp** or **np** = 0.

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_LF_B_MESH

On entry, the left boundary value **a**, has not been set to **x**[0]: **a** = ⟨*value*⟩, **x**[0] = ⟨*value*⟩.

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 = ⟨*value*⟩:
The number of equations = ⟨*value*⟩.

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 = ⟨*value*⟩:
The number of known right boundary values = ⟨*value*⟩:
The number of equations = ⟨*value*⟩.

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: $\mathbf{tol} = \langle value\rangle$.

NE_RT_B_MESH

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

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 $= \langle value\rangle$.

7 Accuracy

The solution returned by nag_ode_bvp_fd_nonlin_fixedbc (d02gac) 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 Parallelism and Performance

nag_ode_bvp_fd_nonlin_fixedbc (d02gac) is not threaded in any implementation.

9 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 that you are specifying 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 the known and estimated boundary values are set to zero, 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 nag_ode_bvp_fd_nonlin_gen (d02rac) 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.

10 Example

We solve the differential equation

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

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 $\mathbf{tol} = 1.0\text{e}{-3}$. 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.

10.1 Program Text

```
/* nag_ode_bvp_fd_nonlin_fixedbc (d02gac) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 *
 */

#include <nag.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, const double y[],
                           double f[], Nag_User *comm);
#ifdef __cplusplus
}
#endif

#define NEQ 3
#define MNP 40

#define U(I, J) u[(I) *tdu + J]
#define Y(I, J) y[(I) *tdy + J]
#define V(I, J) v[(I) *tdv + J]

int main(void)
{
  Integer exit_status = 0, i, j, k, mnp, neq, np, tdu, tdv, tdy, *v = 0;
  NagError fail;
  Nag_User comm;
  double a, b, beta, tol, *u = 0, *x = 0, *y = 0;

  INIT_FAIL(fail);

  printf("nag_ode_bvp_fd_nonlin_fixedbc (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;
  if (mnp >= 32 && neq >= 2) {
    if (!(u = NAG_ALLOC(neq * 2, double)) ||
        !(x = NAG_ALLOC(mnp, double)) ||
        !(y = NAG_ALLOC(neq * mnp, double)) ||
        !(v = NAG_ALLOC(neq * 2, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
    tdu = 2;
```

```
    tdy = mnp;
    tdv = 2;
  }
  else {
    exit_status = 1;
    return exit_status;
  }
  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) {
    printf("\nProblem with beta = %7.4f\n", beta);
    /* nag_ode_bvp_fd_nonlin_fixedbc (d02gac).
     * Ordinary differential equations solver, for simple
     * nonlinear two-point boundary value problems, using a
     * finite difference technique with deferred correction
     */
    nag_ode_bvp_fd_nonlin_fixedbc(neq, fcn, a, b, u, v, mnp,
                                  &np, x, y, tol, &comm, &fail);

    if (fail.code == NE_NOERROR || fail.code == NE_CONV_ROUNDOFF) {
      if (fail.code == NE_CONV_ROUNDOFF) {
        printf("Error from nag_ode_bvp_fd_nonlin_fixedbc (d02gac)"
               ".\n%s\n", fail.message);
        exit_status = 2;
      }
      printf("\nSolution on final mesh of %" NAG_IFMT " points\n", np);
      printf("     X          Y(1)        Y(2)        Y(3)\n");
      for (i = 0; i <= np - 1; ++i) {
        printf(" %9.4f ", x[i]);
        for (j = 0; j < neq; ++j)
          printf(" %9.4f  ", Y(j, i));
        printf("\n");
      }
      beta += 0.2;
    }
    else {
      printf("Error from nag_ode_bvp_fd_nonlin_fixedbc (d02gac).\n%s\n",
             fail.message);
      exit_status = 1;
    }
  }
END:
  NAG_FREE(u);
  NAG_FREE(x);
  NAG_FREE(y);
  NAG_FREE(v);
  return exit_status;
}

static void NAG_CALL fcn(Integer neq, double x, const double y[], double f[],
                         Nag_User *comm)
{
  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]);
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_ode_bvp_fd_nonlin_fixedbc (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
```