NAG Library Function Document

nag_ode_bvp_fd_lin_gen (d02gbc)

1 Purpose

nag_ode_bvp_fd_lin_gen (d02gbc) solves a general linear two-point boundary value problem for a system of ordinary differential equations using a deferred correction technique.

2 Specification

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

void nag_ode_bvp_fd_lin_gen (Integer neq,

    void (*fcnf)(Integer neq, double x, double f[], Nag_User *comm),

    void (*fcng)(Integer neq, double x, double g[], Nag_User *comm),

    double a, double b, double c[], double d[], double gam[], Integer mnp,
    Integer *np, double x[], double y[], double tol, Nag_User *comm,
    NagError *fail)
```

3 Description

nag_ode_bvp_fd_lin_gen (d02gbc) solves the linear two-point boundary value problem for a system of **neq** 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 **neq** by **neq** matrices, and $g(x)$ and γ are **neq** component vectors. The approximate solution to (1) and (2) is found using a finite difference method with deferred correction. The algorithm is a specialisation of that used in the function nag_ode_bvp_fd_nonlin_gen (d02rac) which solves a nonlinear version of (1) and (2). The nonlinear version of the algorithm is described fully in Pereyra (1979).

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

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; that is **neq** is the order of system (1).

Constraint: **neq** ≥ 2.

2: **fcnf** – function, supplied by the user *External Function*

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

The specification of **fcnf** is:

```
void fcnf (Integer neq, double x, 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 independent variable x.

3: **f**[**neq** $\times$ **neq**] – double *Output*

On exit: the (i, j)th element of the matrix $F(x)$, for $i, j = 1, 2, \ldots,$ **neq** where F_{ij} is set by **f**[$(i - 1) \times$ **neq** $+ (j - 1)$]. (See Section 10 for an example.)

4: **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: **fcng** – function, supplied by the user *External Function*

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

The specification of **fcng** is:

```
void fcng (Integer neq, double x, double g[], Nag_User *comm)
```

1: **neq** – Integer *Input*

On entry: the number of differential equations.

2: **x** – double *Input*

On entry: the value of the independent variable x.

3: **g**[**neq**] – double *Output*

On exit: the ith element of the vector $g(x)$, for $i = 1, 2, \ldots,$ **neq**. (See Section 10 for an example.)

4: **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.)

If you do not wish to supply **fcng** the actual argument **fcng** must be the NAG defined null function pointer NULLFN.

4:    **a** – double   *Input*

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

5:    **b** – double   *Input*

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

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

6:    **c**[**neq** × **neq**] – double   *Input/Output*
7:    **d**[**neq** × **neq**] – double   *Input/Output*
8:    **gam**[**neq**] – double   *Input/Output*

On entry: 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).

On exit: the rows of **c** and **d** and the components of **gam** are re-ordered 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 boundary conditions 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 **neq** for the problem to be properly posed. Any violation of these conditions will lead to an error exit.

9:    **mnp** – Integer   *Input*

On entry: the maximum permitted number of mesh points.

Constraint: **mnp** ≥ 32.

10:   **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 ≤ **np** ≤ **mnp**.

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

11: **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:

$$\mathbf{a} = \mathbf{x}[0] < \mathbf{x}[1] < \cdots < \mathbf{x}[\mathbf{np} - 1] = \mathbf{b}, \tag{3}$$

for **np** ≥ 4.

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 (3).

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

On exit: the approximate solution $z_j(x_i)$ satisfying (4), 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.

13: **tol** – double *Input*

On entry: a positive absolute error tolerance.

If

$$a = x_1 < x_2 < \cdots < x_{\mathbf{np}} = b \tag{4}$$

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

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

Constraint: **tol** > 0.0.

14: **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 **fcnf** and **fcng**. 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.

15: **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, $\mathbf{b} = \langle value \rangle$ while $\mathbf{a} = \langle value \rangle$. These arguments must satisfy $\mathbf{b} > \mathbf{a}$.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BOUND_COND_COL

More than **neq** columns of the **neq** by $2 \times$ **neq** matrix $[C, D]$ are identically zero. i.e., the boundary conditions are rank deficient. The number of non-identically zero columns is $\langle value \rangle$.

NE_BOUND_COND_LC

At least one row of the **neq** by $2 \times$ **neq** matrix $[C, D]$ is a linear combination of the other rows, i. e., the boundary conditions are rank deficient. The index of the first such row is $\langle value \rangle$.

NE_BOUND_COND_MAT

One of the matrices C or D is identically zero, i.e., the problem is of initial value and not of the boundary type.

NE_BOUND_COND_NLC

At least one row of the **neq** by $2 \times$ **neq** matrix $[C, D]$ is a linear combination of the other rows determined up to a numerical tolerance, i.e., the boundary conditions are rank deficient. The index of first such row is $\langle value \rangle$. 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

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

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

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

NE_BOUND_COND_ROW

Row $\langle value \rangle$ of the array **c** and the corresponding row of array **d** are identically zero, i.e., the boundary conditions are rank deficient.

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** $= \langle value \rangle$.
Constraint: **mnp** ≥ 32.

On entry, **neq** $= \langle value \rangle$.
Constraint: **neq** ≥ 2.

NE_INT_RANGE_CONS_2

On entry, **np** $= \langle value \rangle$ and **mnp** $= \langle value \rangle$. The argument **np** must satisfy either $4 \leq$ **np** $\leq$ **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 $\mathbf{x}[0]$: $\mathbf{a} = \langle value \rangle$, $\mathbf{x}[0] = \langle value \rangle$.

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 $\mathbf{x}[\mathbf{np} - 1]$: $\mathbf{b} = \langle value \rangle$, $\mathbf{x}[\mathbf{np} - 1] = \langle value \rangle$.

7   Accuracy

The solution returned by the function will be accurate to your tolerance as defined by the relation (4) 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_lin_gen (d02gbc) 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 (and meshes) used and the number of deferred corrections.

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

10   Example

We solve the problem (written as a first order system)

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

with boundary conditions

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

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

10.1 Program Text

```
/* nag_ode_bvp_fd_lin_gen (d02gbc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 *
```

```
 */

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

#ifdef __cplusplus
extern "C"
{
#endif
  static void NAG_CALL fcnf(Integer neq, double x, double f[],
                            Nag_User *comm);
#ifdef __cplusplus
}
#endif

#define NEQ 2
#define MNP 70

#define C(I, J) c[(I) *tdc + J]
#define D(I, J) d[(I) *tdd + J]
#define Y(I, J) y[(I) *tdy + J]

int main(void)
{

  Integer exit_status = 0, i, j, mnp, neq, np, tdc, tdd, tdy;
  NagError fail;
  Nag_User comm;
  double a, b, *c = 0, *d = 0, eps, *gam = 0, tol, *x = 0, *y = 0;

  INIT_FAIL(fail);

  printf("nag_ode_bvp_fd_lin_gen (d02gbc) Example Program Results\n");

  /* For communication with function fcnf()
   * assign address of eps to comm.p.
   */
  comm.p = (Pointer) &eps;

  neq = NEQ;
  mnp = MNP;
  tol = 1.0e-3;
  np = 0;
  a = 0.0;
  b = 1.0;
  if (mnp >= 32 && neq >= 2) {
    if (!(c = NAG_ALLOC(NEQ * NEQ, double)) ||
        !(d = NAG_ALLOC(NEQ * NEQ, double)) ||
        !(gam = NAG_ALLOC(NEQ, double)) ||
        !(x = NAG_ALLOC(MNP, double)) || !(y = NAG_ALLOC(NEQ * MNP, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
    tdc = neq;
    tdd = neq;
    tdy = mnp;
  }
  else {
    exit_status = 1;
    return exit_status;
  }

  for (i = 0; i < neq; ++i) {
    gam[i] = 0.0;
    for (j = 0; j < neq; ++j) {
      C(i, j) = 0.0;
```

```
      D(i, j) = 0.0;
    }
  }
  C(0, 0) = 1.0;
  D(1, 0) = 1.0;
  gam[1] = 1.0;
  for (i = 1; i <= 2; ++i) {
    eps = pow(10.0, (double) -i);
    printf("\nProblem with epsilon = %7.4f\n", eps);
    /* nag_ode_bvp_fd_lin_gen (d02gbc).
     * Ordinary differential equations solver, for general
     * linear two-point boundary value problems, using a finite
     * difference technique with deferred correction
     */
    nag_ode_bvp_fd_lin_gen(neq, fcnf, NULLFN, a, b, c, d, gam,
                           mnp, &np, x, y, tol, &comm, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_ode_bvp_fd_lin_gen (d02gbc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }
    printf("\nApproximate solution on final mesh of %" NAG_IFMT " points\n",
           np);
    printf("    X(I)          Y(1,I)\n");
    for (j = 0; j < np; ++j)
      printf("%9.4f    %9.4f\n", x[j], Y(0, j));
  }
END:
  NAG_FREE(c);
  NAG_FREE(d);
  NAG_FREE(gam);
  NAG_FREE(x);
  NAG_FREE(y);
  return exit_status;
}

static void NAG_CALL fcnf(Integer neq, double x, double f[], Nag_User *comm)
{
#define F(I, J) f[(I) *neq+J]
  double *eps = (double *) comm->p;

  F(0, 0) = 0.0;
  F(0, 1) = 1.0;
  F(1, 0) = 0.0;
  F(1, 1) = -1.0 / *eps;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_ode_bvp_fd_lin_gen (d02gbc) Example Program Results

Problem with epsilon =  0.1000

Approximate solution on final mesh of 15 points
    X(I)          Y(1,I)
  0.0000      0.0000
  0.0278      0.2425
  0.0556      0.4263
  0.1111      0.6708
  0.1667      0.8112
  0.2222      0.8917
  0.2778      0.9379
  0.3333      0.9644
  0.4444      0.9883
  0.5556      0.9962
```

```
    0.6667      0.9988
    0.7500      0.9995
    0.8333      0.9998
    0.9167      0.9999
    1.0000      1.0000

Problem with epsilon =  0.0100

Approximate solution on final mesh of 49 points
    X(I)        Y(1,I)
    0.0000      0.0000
    0.0009      0.0884
    0.0019      0.1690
    0.0028      0.2425
    0.0037      0.3095
    0.0046      0.3706
    0.0056      0.4262
    0.0065      0.4770
    0.0074      0.5232
    0.0083      0.5654
    0.0093      0.6038
    0.0111      0.6708
    0.0130      0.7265
    0.0148      0.7727
    0.0167      0.8111
    0.0185      0.8431
    0.0204      0.8696
    0.0222      0.8916
    0.0241      0.9100
    0.0259      0.9252
    0.0278      0.9378
    0.0306      0.9529
    0.0333      0.9643
    0.0361      0.9730
    0.0389      0.9795
    0.0417      0.9845
    0.0444      0.9883
    0.0472      0.9911
    0.0500      0.9933
    0.0528      0.9949
    0.0556      0.9961
    0.0648      0.9985
    0.0741      0.9994
    0.0833      0.9998
    0.0926      0.9999
    0.1019      1.0000
    0.1111      1.0000
    0.1389      1.0000
    0.1667      1.0000
    0.2222      1.0000
    0.2778      1.0000
    0.3333      1.0000
    0.4444      1.0000
    0.5556      1.0000
    0.6667      1.0000
    0.7500      1.0000
    0.8333      1.0000
    0.9167      1.0000
    1.0000      1.0000
```