NAG Library Function Document

nag_ode_bvp_coll_nlin_setup (d02tvc)

1 Purpose

nag_ode_bvp_coll_nlin_setup (d02tvc) is a setup function which must be called prior to the first call of the nonlinear two-point boundary value solver nag_ode_bvp_coll_nlin_solve (d02tlc).

2 Specification

```
#include <nag.h>
#include <nagd02.h>
void nag_ode_bvp_coll_nlin_setup (Integer neq, const Integer m[],
     Integer nlbc, Integer nrbc, Integer ncol, const double tols[],
     Integer mxmesh, Integer nmesh, const double mesh[],
     const Integer ipmesh[], double rcomm[], Integer lrcomm, Integer icomm[],
     Integer licomm, NagError *fail)
```

3 Description

nag_ode_bvp_coll_nlin_setup (d02tvc) and its associated functions (nag_ode_bvp_coll_nlin_solve (d02tlc), nag_ode_bvp_coll_nlin_contin (d02txc), nag_ode_bvp_coll_nlin_interp (d02tyc) and nag_ode_bvp_coll_nlin_diag (d02tzc)) solve the two-point boundary value problem for a nonlinear system of ordinary differential equations

$$\begin{aligned} y_1^{(m_1)}(x) &= f_1\left(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots, y_n^{(m_n-1)}\right) \\ y_2^{(m_2)}(x) &= f_2\left(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots, y_n^{(m_n-1)}\right) \\ &\vdots \\ y_n^{(m_n)}(x) &= f_n\left(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots, y_n^{(m_n-1)}\right) \end{aligned}$$

over an interval $[a,b]$ subject to p (>0) nonlinear boundary conditions at a and q (>0) nonlinear boundary conditions at b, where $p+q=\sum_{i=1}^{n} m_i$. Note that $y_i^{(m)}(x)$ is the mth derivative of the ith solution component. Hence $y_i^{(0)}(x)=y_i(x)$. The left boundary conditions at a are defined as

$$g_i(z(y(a)))=0, \quad i=1,2,\ldots,p,$$

and the right boundary conditions at b as

$$\bar{g}_j(z(y(b)))=0, \quad j=1,2,\ldots,q,$$

where $y=(y_1,y_2,\ldots,y_n)$ and

$$z(y(x))=\left(y_1(x), y_1^{(1)}(x), \ldots, y_1^{(m_1-1)}(x), y_2(x), \ldots, y_n^{(m_n-1)}(x)\right).$$

See Section 9 for information on how boundary value problems of a more general nature can be treated.

nag_ode_bvp_coll_nlin_setup (d02tvc) is used to specify an initial mesh, error requirements and other details. nag_ode_bvp_coll_nlin_solve (d02tlc) is then used to solve the boundary value problem.

The solution function nag_ode_bvp_coll_nlin_solve (d02tlc) proceeds as follows. A modified Newton method is applied to the equations

$$y_i^{(m_i)}(x)-f_i(x,z(y(x)))=0, \quad i=1,\ldots,n$$

and the boundary conditions. To solve these equations numerically the components y_i are approximated

by piecewise polynomials v_{ij} using a monomial basis on the jth mesh sub-interval. The coefficients of the polynomials v_{ij} form the unknowns to be computed. Collocation is applied at Gaussian points

$$v_{ij}^{(m_i)}(x_{jk}) - f_i(x_{jk}, z(v(x_{jk}))) = 0, \quad i = 1, 2, \ldots, n,$$

where x_{jk} is the kth collocation point in the jth mesh sub-interval. Continuity at the mesh points is imposed, that is

$$v_{ij}(x_{j+1}) - v_{i,j+1}(x_{j+1}) = 0, \quad i = 1, 2, \ldots, n,$$

where x_{j+1} is the right-hand end of the jth mesh sub-interval. The linearized collocation equations and boundary conditions, together with the continuity conditions, form a system of linear algebraic equations, an almost block diagonal system which is solved using special linear solvers. To start the modified Newton process, an approximation to the solution on the initial mesh must be supplied via the procedure argument **guess** of nag_ode_bvp_coll_nlin_solve (d02tlc).

The solver attempts to satisfy the conditions

$$\frac{\|y_i - v_i\|}{(1.0 + \|v_i\|)} \leq \mathbf{tols}[i-1], \quad i = 1, 2, \ldots, n, \tag{1}$$

where v_i is the approximate solution for the ith solution component and **tols** is supplied by you. The mesh is refined by trying to equidistribute the estimated error in the computed solution over all mesh sub-intervals, and an extrapolation-like test (doubling the number of mesh sub-intervals) is used to check for (1).

The functions are based on modified versions of the codes COLSYS and COLNEW (see Ascher *et al.* (1979) and Ascher and Bader (1987)). A comprehensive treatment of the numerical solution of boundary value problems can be found in Ascher *et al.* (1988) and Keller (1992).

4 References

Ascher U M and Bader G (1987) A new basis implementation for a mixed order boundary value ODE solver *SIAM J. Sci. Stat. Comput.* **8** 483–500

Ascher U M, Christiansen J and Russell R D (1979) A collocation solver for mixed order systems of boundary value problems *Math. Comput.* **33** 659–679

Ascher U M, Mattheij R M M and Russell R D (1988) *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* Prentice–Hall

Gill P E, Murray W and Wright M H (1981) *Practical Optimization* Academic Press

Keller H B (1992) *Numerical Methods for Two-point Boundary-value Problems* Dover, New York

Schwartz I B (1983) Estimating regions of existence of unstable periodic orbits using computer-based techniques *SIAM J. Sci. Statist. Comput.* **20(1)** 106–120

5 Arguments

1: **neq** – Integer *Input*

On entry: n, the number of ordinary differential equations to be solved.

Constraint: **neq** ≥ 1.

2: **m**[**neq**] – const Integer *Input*

On entry: **m**$[i-1]$ must contain m_i, the order of the ith differential equation, for $i = 1, 2, \ldots, n$.

Constraint: $1 \leq$ **m**$[i-1] \leq 4$, for $i = 1, 2, \ldots, n$.

3: **nlbc** – Integer *Input*

On entry: p, the number of left boundary conditions defined at the left-hand end, a (= **mesh**[0]).

Constraint: **nlbc** ≥ 1.

4: **nrbc** – Integer *Input*

On entry: q, the number of right boundary conditions defined at the right-hand end, b (= **mesh**[**nmesh** − 1]).

Constraints:

$$\textbf{nrbc} \geq 1;$$
$$\textbf{nlbc} + \textbf{nrbc} = \sum_{i=1}^{n} \textbf{m}[i-1].$$

5: **ncol** – Integer *Input*

On entry: the number of collocation points to be used in each mesh sub-interval.

Constraint: $m_{\max} \leq$ **ncol** ≤ 7, where $m_{\max} = \max(\textbf{m}[i-1])$.

6: **tols**[**neq**] – const double *Input*

On entry: **tols**$[i-1]$ must contain the error requirement for the ith solution component.

Constraint: $100 \times$ ***machine precision*** $<$ **tols**$[i-1] < 1.0$, for $i = 1, 2, \ldots, n$.

7: **mxmesh** – Integer *Input*

On entry: the maximum number of mesh points to be used during the solution process.

Constraint: **mxmesh** $\geq 2 \times$ **nmesh** $- 1$.

8: **nmesh** – Integer *Input*

On entry: the number of points to be used in the initial mesh of the solution process.

Constraint: **nmesh** ≥ 6.

9: **mesh**[**mxmesh**] – const double *Input*

On entry: the positions of the initial **nmesh** mesh points. The remaining elements of **mesh** need not be set. You should try to place the mesh points in areas where you expect the solution to vary most rapidly. In the absence of any other information the points should be equally distributed on $[a, b]$.

mesh[0] must contain the left boundary point, a, and **mesh**[**nmesh** − 1] must contain the right boundary point, b.

Constraint: **mesh**$[i-1] <$ **mesh**$[i]$, for $i = 1, 2, \ldots,$ **nmesh** $- 1$.

10: **ipmesh**[**mxmesh**] – const Integer *Input*

On entry: **ipmesh**$[i-1]$ specifies whether or not the initial mesh point defined in **mesh**$[i-1]$, for $i = 1, 2, \ldots,$ **nmesh**, should be a fixed point in all meshes computed during the solution process. The remaining elements of **ipmesh** need not be set.

ipmesh$[i-1] = 1$
Indicates that **mesh**$[i-1]$ should be a fixed point in all meshes.

ipmesh$[i-1] = 2$
Indicates that **mesh**$[i-1]$ is not a fixed point.

Constraints:

ipmesh$[0] = 1$ and **ipmesh**$[$**nmesh** $- 1] = 1$, (i.e., the left and right boundary points, a and b, must be fixed points, in all meshes);
ipmesh$[i-1] = 1$ or 2, for $i = 2, 3, \ldots,$ **nmesh** $- 1$.

11: **rcomm**[**lrcomm**] – double *Communication Array*

On exit: contains information for use by nag_ode_bvp_coll_nlin_solve (d02tlc). This **must** be the same array as will be supplied to nag_ode_bvp_coll_nlin_solve (d02tlc). The contents of this array **must** remain unchanged between calls.

12: **lrcomm** – Integer *Input*

On entry: the dimension of the array **rcomm**. If **lrcomm** $= 0$, a communication array size query is requested. In this case there is an immediate return with communication array dimensions stored in **icomm**; **icomm**$[0]$ contains the required dimension of **rcomm**, while **icomm**$[1]$ contains the required dimension of **icomm**.

Constraint: **lrcomm** $= 0$, or **lrcomm** $\geq 51 +$ **neq** $+$ **mxmesh** $\times (2 + m^* + k_n) - k_n +$ **mxmesh**$/2$,
where $m^* = \sum_{i=1}^{n}$ **m**$[i-1]$ and $k_n =$ **ncol** $\times$ **neq**.

13: **icomm**[**licomm**] – Integer *Communication Array*

On exit: contains information for use by nag_ode_bvp_coll_nlin_solve (d02tlc). This **must** be the same array as will be supplied to nag_ode_bvp_coll_nlin_solve (d02tlc). The contents of this array **must** remain unchanged between calls. If **lrcomm** $= 0$, a communication array size query is requested. In this case, on immediate return, **icomm**$[0]$ will contain the required dimension for **rcomm** while **icomm**$[1]$ will contain the required dimension for **icomm**.

14: **licomm** – Integer *Input*

On entry: the dimension of the array **icomm**. If **lrcomm** $= 0$, a communication array size query is requested. In this case **icomm** need only be of dimension 2 in order to hold the required communication array dimensions for the given problem and algorithmic parameters.

Constraints:

if **lrcomm** $= 0$, **licomm** ≥ 2;
otherwise **licomm** $\geq 23 +$ **neq** $+$ **mxmesh**.

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

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

6 Error Indicators and Warnings

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **ipmesh**$[0]$ or **ipmesh**$[$**nmesh** $- 1]$ does not equal 1.

On entry, **ipmesh**$[i] \neq 1$ or 2 for some i.

On entry, **licomm** $=$ ⟨*value*⟩.
Constraint: **licomm** $\geq$ ⟨*value*⟩.

On entry, **lrcomm** $=$ ⟨*value*⟩.
Constraint: **lrcomm** $= 0$ or **lrcomm** $\geq$ ⟨*value*⟩.

On entry, $\mathbf{m}[\langle value \rangle] = \langle value \rangle$.
Constraint: $1 \leq \mathbf{m}[i-1] \leq 4$ for all i.

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

On entry, **nmesh** $= \langle value \rangle$.
Constraint: **nmesh** ≥ 6.

NE_INT_2

On entry, **mxmesh** $= \langle value \rangle$ and **nmesh** $= \langle value \rangle$.
Constraint: **mxmesh** $\geq 2 \times$ **nmesh** $- 1$.

On entry, **ncol** $= \langle value \rangle$ and $\max(\mathbf{m}[i]) = \langle value \rangle$.
Constraint: $\max(\mathbf{m}[i]) \leq$ **ncol** ≤ 7.

On entry, **nlbc** $= \langle value \rangle$ and **nrbc** $= \langle value \rangle$.
Constraint: **nlbc** ≥ 1 and **nrbc** ≥ 1.

NE_INT_3

On entry, **nlbc** $= \langle value \rangle$, **nrbc** $= \langle value \rangle$ and $\text{sum}(\mathbf{m}[i]) = \langle value \rangle$.
Constraint: **nlbc** $+$ **nrbc** $= \text{sum}(\mathbf{m}[i])$.

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_NOT_STRICTLY_INCREASING

On entry, the elements of **mesh** are not strictly increasing.

NE_REAL

On entry, **tols**$[\langle value \rangle] = \langle value \rangle$.
Constraint: **tols**$[i-1] > \langle value \rangle$ for all i.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

For problems where sharp changes of behaviour are expected over short intervals it may be advisable to:

– use a large value for **ncol**;

– cluster the initial mesh points where sharp changes in behaviour are expected;

– maintain fixed points in the mesh using the argument **ipmesh** to ensure that the remeshing process does not inadvertently remove mesh points from areas of known interest before they are detected automatically by the algorithm.

9.1 Nonseparated Boundary Conditions

A boundary value problem with nonseparated boundary conditions can be treated by transformation to an equivalent problem with separated conditions. As a simple example consider the system

$$y_1' = f_1(x, y_1, y_2)$$

$$y_2' = f_2(x, y_1, y_2)$$

on $[a, b]$ subject to the boundary conditions

$$\begin{aligned} g_1(y_1(a)) &= 0 \\ g_2(y_2(a), y_2(b)) &= 0. \end{aligned}$$

By adjoining the trivial ordinary differential equation

$$r' = 0,$$

which implies $r(a) = r(b)$, and letting $r(b) = y_2(b)$, say, we have a new system

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2) \\ y_2' &= f_2(x, y_1, y_2) \\ r' &= 0, \end{aligned}$$

subject to the separated boundary conditions

$$\begin{aligned} g_1(y_1(a)) &= 0 \\ g_2(y_2(a), r(a)) &= 0 \\ y_2(b) - r(b) &= 0. \end{aligned}$$

There is an obvious overhead in adjoining an extra differential equation: the system to be solved is increased in size.

9.2 Multipoint Boundary Value Problems

Multipoint boundary value problems, that is problems where conditions are specified at more than two points, can also be transformed to an equivalent problem with two boundary points. Each sub-interval defined by the multipoint conditions can be transformed onto the interval $[0, 1]$, say, leading to a larger set of differential equations. The boundary conditions of the transformed system consist of the original boundary conditions and the conditions imposed by the requirement that the solution components be continuous at the interior break-points. For example, consider the equation

$$y^{(3)} = f\left(t, y, y^{(1)}, y^{(2)}\right) \quad \text{on} \quad [a, c]$$

subject to the conditions

$$\begin{aligned} y(a) &= A \\ y(b) &= B \\ y^{(1)}(c) &= C \end{aligned}$$

where $a < b < c$. This can be transformed to the system

$$\left.\begin{aligned} y_1^{(3)} &= f\left(t, y_1, y_1^{(1)}, y_1^{(2)}\right) \\ y_2^{(3)} &= f\left(t, y_2, y_2^{(1)}, y_2^{(2)}\right) \end{aligned}\right\} \quad \text{on } [0, 1]$$

where

$$\begin{aligned} y_1 &\equiv y \quad \text{on} \quad [a, b] \\ y_2 &\equiv y \quad \text{on} \quad [b, c], \end{aligned}$$

subject to the boundary conditions

$$\begin{aligned} y_1(0) &= A \\ y_1(1) &= B \\ y_2^{(1)}(1) &= C \\ y_2(0) &= B \quad (\text{from } y_1(1) = y_2(0)) \\ y_1^{(1)}(1) &= y_2^{(1)}(0) \\ y_1^{(2)}(1) &= y_2^{(2)}(0). \end{aligned}$$

In this instance two of the resulting boundary conditions are nonseparated but they may next be treated as described above.

9.3 High Order Systems

Systems of ordinary differential equations containing derivatives of order greater than four can always be reduced to systems of order suitable for treatment by nag_ode_bvp_coll_nlin_setup (d02tvc) and its related functions. For example suppose we have the sixth-order equation

$$y^{(6)} = -y.$$

Writing the variables $y_1 = y$ and $y_2 = y^{(4)}$ we obtain the system

$$\begin{aligned} y_1^{(4)} &= y_2 \\ y_2^{(2)} &= -y_1 \end{aligned}$$

which has maximal order four, or writing the variables $y_1 = y$ and $y_2 = y^{(3)}$ we obtain the system

$$\begin{aligned} y_1^{(3)} &= y_2 \\ y_2^{(3)} &= -y_1 \end{aligned}$$

which has maximal order three. The best choice of reduction by choosing new variables will depend on the structure and physical meaning of the system. Note that you will control the error in each of the variables y_1 and y_2. Indeed, if you wish to control the error in certain derivatives of the solution of an equation of order greater than one, then you should make those derivatives new variables.

9.4 Fixed Points and Singularities

The solver function nag_ode_bvp_coll_nlin_solve (d02tlc) employs collocation at Gaussian points in each sub-interval of the mesh. Hence the coefficients of the differential equations are not evaluated at the mesh points. Thus, fixed points should be specified in the mesh where either the coefficients are singular, or the solution has less smoothness, or where the differential equations should not be evaluated. Singular coefficients at boundary points often arise when physical symmetry is used to reduce partial differential equations to ordinary differential equations. These do not pose a direct numerical problem for using this code but they can severely impact its convergence.

9.5 Numerical Jacobians

The solver function nag_ode_bvp_coll_nlin_solve (d02tlc) requires an external function **fjac** to evaluate the partial derivatives of f_i with respect to the elements of $z(y)$ $\left(= \left(y_1, y_1^1, \ldots, y_1^{(m_1-1)}, y_2, \ldots, y_n^{(m_n-1)}\right)\right)$. In cases where the partial derivatives are difficult to evaluate, numerical approximations can be used. However, this approach might have a negative impact on the convergence of the modified Newton method. You could consider the use of symbolic mathematic packages and/or automatic differentiation packages if available to you.

See Section 10 in nag_ode_bvp_coll_nlin_diag (d02tzc) for an example using numerical approximations to the Jacobian. There central differences are used and each f_i is assumed to depend on all the components of z. This requires two evaluations of the system of differential equations for each component of z. The perturbation used depends on the size of each component of z and a minimum quantity dependent on the ***machine precision***. The cost of this approach could be reduced by employing an alternative difference scheme and/or by only perturbing the components of z which appear in the

definitions of the f_i. A discussion on the choice of perturbation factors for use in finite difference approximations to partial derivatives can be found in Gill *et al.* (1981).

10 Example

The following example is used to illustrate the treatment of nonseparated boundary conditions. See also nag_ode_bvp_coll_nlin_solve (d02tlc), nag_ode_bvp_coll_nlin_contin (d02txc), nag_ode_bvp_coll_nlin_interp (d02tyc) and nag_ode_bvp_coll_nlin_diag (d02tzc), for the illustration of other facilities.

The following equations model of the spread of measles. See Schwartz (1983). Under certain assumptions the dynamics of the model can be expressed as

$$\begin{aligned} y_1' &= \mu - \beta(x) y_1 y_3 \\ y_2' &= \beta(x) y_1 y_3 - y_2/\lambda \\ y_3' &= y_2/\lambda - y_3/\eta \end{aligned}$$

subject to the periodic boundary conditions

$$y_i(0) = y_i(1), \quad i = 1, 2, 3.$$

Here y_1, y_2 and y_3 are respectively the proportions of susceptibles, infectives and latents to the whole population. λ ($= 0.0279$ years) is the latent period, η ($= 0.01$ years) is the infectious period and μ ($= 0.02$) is the population birth rate. $\beta(x) = \beta_0(1.0 + \cos 2\pi x)$ is the contact rate where $\beta_0 = 1575.0$.

The nonseparated boundary conditions are treated as described in Section 9 by adjoining the trivial differential equations

$$\begin{aligned} y_4' &= 0 \\ y_5' &= 0 \\ y_6' &= 0 \end{aligned}$$

that is y_4, y_5 and y_6 are constants. The boundary conditions of the augmented system can then be posed in the separated form

$$\begin{aligned} y_1(0) - y_4(0) &= 0 \\ y_2(0) - y_5(0) &= 0 \\ y_3(0) - y_6(0) &= 0 \\ y_1(1) - y_4(1) &= 0 \\ y_2(1) - y_5(1) &= 0 \\ y_3(1) - y_6(1) &= 0. \end{aligned}$$

This is a relatively easy problem and an (arbitrary) initial guess of 1 for each component suffices, even though two components of the solution are much smaller than 1.

10.1 Program Text

```
/* nag_ode_bvp_coll_nlin_setup (d02tvc) Example Program.
 *
 * Copyright 2013 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd02.h>
#include <nagx01.h>

typedef struct {
  double  beta0, eta, lambda, mu;
} func_data;

#ifdef __cplusplus
```

```
extern "C" {
#endif
static void NAG_CALL ffun(double x, const double y[], Integer neq,
                          const Integer m[], double f[], Nag_Comm *comm);
static void NAG_CALL fjac(double x, const double y[], Integer neq,
                          const Integer m[], double dfdy[], Nag_Comm *comm);
static void NAG_CALL gafun(const double ya[], Integer neq, const Integer m[],
                           Integer nlbc, double ga[], Nag_Comm *comm);
static void NAG_CALL gbfun(const double yb[], Integer neq, const Integer m[],
                           Integer nrbc, double gb[], Nag_Comm *comm);
static void NAG_CALL gajac(const double ya[], Integer neq, const Integer m[],
                           Integer nlbc, double dgady[], Nag_Comm *comm);
static void NAG_CALL gbjac(const double yb[], Integer neq, const Integer m[],
                           Integer nrbc, double dgbdy[], Nag_Comm *comm);
static void NAG_CALL guess(double x, Integer neq, const Integer m[], double y[],
                           double dym[], Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

int main(void)
{
  /* Scalars */
  Integer    exit_status = 0, neq = 6, mmax = 1, nlbc = 3, nrbc = 3;
  double     dx, ermx, beta0, eta, lambda, mu;
  Integer    i, iermx, ijermx, j, licomm, lrcomm, mxmesh, ncol, nmesh;
  /* Arrays */
  static double ruser[7] = {-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0};
  double     *mesh = 0, *rcomm = 0, *tols = 0, *y = 0;
  double     rdum[1];
  Integer    *ipmesh = 0, *icomm = 0, *m = 0;
  Integer    idum[2];
  func_data  fd;
  /* Nag Types */
  Nag_Comm   comm;
  NagError   fail;

  INIT_FAIL(fail);

  printf ("nag_ode_bvp_coll_nlin_setup (d02tvc) Example Program Results\n\n");

  /* For communication with user-supplied functions: */
  comm.user = ruser;

  /* Skip heading in data file*/
  scanf("%*[^\n] ");
  scanf("%"NAG_IFMT "%"NAG_IFMT "%"NAG_IFMT "%*[^\n] ", &ncol, &nmesh, &mxmesh);
  if (!(mesh   = NAG_ALLOC(mxmesh, double)) ||
      !(m      = NAG_ALLOC(neq, Integer)) ||
      !(tols   = NAG_ALLOC(neq, double)) ||
      !(y      = NAG_ALLOC(neq*mmax, double)) ||
      !(ipmesh = NAG_ALLOC(mxmesh, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Set orders of equations */
  for (i = 0; i < neq; i++) {
    m[i] = 1;
  }
  scanf("%lf%lf%lf%lf%*[^\n] ", &beta0, &eta, &lambda, &mu);
  for (i = 0; i < neq; i++) {
    scanf("%lf", &tols[i]);
  }
  scanf("%*[^\n] ");
  dx = 1.0/(double) (nmesh - 1);
  mesh[0] = 0.0;
  for (i = 1; i < nmesh - 1; i++) {
    mesh[i] = mesh[i - 1] + dx;
```

```
  }
  mesh[nmesh - 1] = 1.0;
  ipmesh[0] = 1;
  for (i = 1; i < nmesh - 1; i++) {
    ipmesh[i] = 2;
  }
  ipmesh[nmesh - 1] = 1;

  /* Set data required for the user-supplied functions */
  fd.beta0 = beta0;
  fd.eta = eta;
  fd.lambda = lambda;
  fd.mu = mu;
  /* Associate the data structure with comm.p */
  comm.p = (Pointer)

  /* Communication space query to get size of rcomm and icomm
   * by setting lrcomm=0 in call to
   * nag_ode_bvp_coll_nlin_setup (d02tvc):
   * Ordinary differential equations, general nonlinear boundary value problem,
   * setup for nag_ode_bvp_coll_nlin_solve (d02tlc).
   */
  nag_ode_bvp_coll_nlin_setup(neq, m, nlbc, nrbc, ncol, tols, mxmesh, nmesh,
                              mesh, ipmesh, rdum, 0, idum, 2, &fail);
  if (fail.code == NE_NOERROR) {
    lrcomm = idum[0];
    licomm = idum[1];

    if (!(rcomm = NAG_ALLOC(lrcomm, double)) ||
        !(icomm = NAG_ALLOC(licomm, Integer))) {
      printf("Allocation failure\n");
      exit_status = -2;
      goto END;
    }
    /* Initialize, again using nag_ode_bvp_coll_nlin_setup (d02tvc). */
    nag_ode_bvp_coll_nlin_setup(neq, m, nlbc, nrbc, ncol, tols, mxmesh, nmesh,
                                mesh, ipmesh, rcomm, lrcomm, icomm, licomm,
                                &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_ode_bvp_coll_nlin_setup (d02tvc).\n%s\n",
             fail.message);
      exit_status = 2;
      goto END;
    }
  }
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_ode_bvp_coll_nlin_setup (d02tvc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Solve*/

  /* nag_ode_bvp_coll_nlin_solve (d02tlc).
   * Ordinary differential equations, general nonlinear boundary value problem,
   * collocation technique.
   */
  nag_ode_bvp_coll_nlin_solve(ffun, fjac, gafun, gbfun, gajac, gbjac, guess,
                              rcomm, icomm, &comm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_ode_bvp_coll_nlin_solve (d02tlc).\n%s\n",
           fail.message);
    exit_status = 3;
    goto END;
  }

  /* Extract mesh.*/

  /* nag_ode_bvp_coll_nlin_diag (d02tzc).
   * Ordinary differential equations, general nonlinear boundary value
```

```
   * problem, diagnostics for nag_ode_bvp_coll_nlin_solve (d02tlc).
   */
  nag_ode_bvp_coll_nlin_diag(mxmesh, &nmesh, mesh, ipmesh, &ermx, &iermx,
                             &ijermx, rcomm, icomm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_ode_bvp_coll_nlin_diag (d02tzc).\n%s\n",
           fail.message);
    exit_status = 4;
    goto END;
  }

  /* Print mesh statistics*/
  printf(" Used a mesh of %4ld  points\n", nmesh);
  printf(" Maximum error = %10.2e  in interval %4ld  for component ",
         ermx, iermx);
  printf("%4"NAG_IFMT " \n\n\n", ijermx);
  printf(" Mesh points:\n");
  for (i = 0; i < nmesh; i++) {
    printf("%4ld(%1ld)", i+1, ipmesh[i]);
    printf("%7.4f%s", mesh[i], (i+1)%4?" ":"\n");
  }
  printf("\n");
  /* Print solution on mesh.*/
  printf("\n Computed solution at mesh points\n");
  printf("    x       y1        y2         y3\n");
  for (i = 0; i < nmesh; i++) {

    /* nag_ode_bvp_coll_nlin_interp (d02tyc).
     * Ordinary differential equations, general nonlinear boundary value
     * problem, interpolation for nag_ode_bvp_coll_nlin_solve (d02tlc).
     */
    nag_ode_bvp_coll_nlin_interp(mesh[i], y, neq, mmax, rcomm, icomm,
                                 &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_ode_bvp_coll_nlin_interp (d02tyc).\n%s\n",
             fail.message);
      exit_status = 5;
      goto END;
    }

    printf("%6.3f ", mesh[i]);
    for (j = 0; j < 3; j++) {
      printf("%11.3e", y[j]);
    }
    printf("\n");
  }

 END:
  NAG_FREE(mesh);
  NAG_FREE(m);
  NAG_FREE(tols);
  NAG_FREE(rcomm);
  NAG_FREE(y);
  NAG_FREE(ipmesh);
  NAG_FREE(icomm);
  return exit_status;
}

static void NAG_CALL ffun(double x, const double y[], Integer neq,
                          const Integer m[], double f[], Nag_Comm *comm)
{
  func_data *fd = (func_data *)comm->p;
  double  beta;
  Integer i;
  double  one = 1.0;
  double  two = 2.0;
  double  zero = 0.0;

  if (comm->user[0] == -1.0)
    {
      printf("(User-supplied callback ffun, first invocation.)\n");
```

```
      comm->user[0] = 0.0;
    }
  /* nag_pi (x01aac). */
  beta = fd->beta0 * (one + cos(two * nag_pi * x));
  f[0] = fd->mu - beta * y[0] * y[2];
  f[1] = beta * y[0] * y[2] - y[1]/fd->lambda;
  f[2] = y[1]/fd->lambda - y[2]/fd->eta;
  for (i = 3; i < 6; i++) {
    f[i] = zero;
  }
}

static void NAG_CALL fjac(double x, const double y[], Integer neq,
                          const Integer m[], double dfdy[], Nag_Comm *comm)
{
  func_data *fd = (func_data *)comm->p;
  double beta;
  double  one = 1.0;
  double  two = 2.0;

  if (comm->user[1] == -1.0)
    {
      printf("(User-supplied callback fjac, first invocation.)\n");
      comm->user[1] = 0.0;
    }
  /* nag_pi (x01aac). */
  beta = fd->beta0 * (one + cos(two * nag_pi * x));
  dfdy[0+0*neq] = -beta * y[2];
  dfdy[0+2*neq] = -beta * y[0];
  dfdy[1+0*neq] =  beta * y[2];
  dfdy[1+1*neq] = -one/fd->lambda;
  dfdy[1+2*neq] =  beta * y[0];
  dfdy[2+1*neq] =  one/fd->lambda;
  dfdy[2+2*neq] = -one/fd->eta;
}

static void NAG_CALL gafun(const double ya[], Integer neq, const Integer m[],
                           Integer nlbc, double ga[], Nag_Comm *comm)
{
  if (comm->user[2] == -1.0)
    {
      printf("(User-supplied callback gafun, first invocation.)\n");
      comm->user[2] = 0.0;
    }
  ga[0] = ya[0] - ya[3];
  ga[1] = ya[1] - ya[4];
  ga[2] = ya[2] - ya[5];
}

static void NAG_CALL gbfun(const double yb[], Integer neq, const Integer m[],
                           Integer nrbc, double gb[], Nag_Comm *comm)
{
  if (comm->user[3] == -1.0)
    {
      printf("(User-supplied callback gbfun, first invocation.)\n");
      comm->user[3] = 0.0;
    }
  gb[0] = yb[0] - yb[3];
  gb[1] = yb[1] - yb[4];
  gb[2] = yb[2] - yb[5];
}

static void NAG_CALL gajac(const double ya[], Integer neq, const Integer m[],
                           Integer nlbc, double dgady[], Nag_Comm *comm)
{
  double  one = 1.0;

  if (comm->user[4] == -1.0)
    {
      printf("(User-supplied callback gajac, first invocation.)\n");
      comm->user[4] = 0.0;
```

```
    }
  dgady[0+0*nlbc] =  one;
  dgady[0+3*nlbc] = -one;
  dgady[1+1*nlbc] =  one;
  dgady[1+4*nlbc] = -one;
  dgady[2+2*nlbc] =  one;
  dgady[2+5*nlbc] = -one;
}

static void NAG_CALL gbjac(const double yb[], Integer neq, const Integer m[],
                           Integer nrbc, double dgbdy[], Nag_Comm *comm)
{
  double  one = 1.0;

  if (comm->user[5] == -1.0)
    {
      printf("(User-supplied callback gbjac, first invocation.)\n");
      comm->user[5] = 0.0;
    }
  dgbdy[0+0*nrbc] =  one;
  dgbdy[0+3*nrbc] = -one;
  dgbdy[1+1*nrbc] =  one;
  dgbdy[1+4*nrbc] = -one;
  dgbdy[2+2*nrbc] =  one;
  dgbdy[2+5*nrbc] = -one;
}

static void NAG_CALL guess(double x, Integer neq, const Integer m[], double y[],
                           double dym[], Nag_Comm *comm)
{
  Integer i;

  if (comm->user[6] == -1.0)
    {
      printf("(User-supplied callback guess, first invocation.)\n");
      comm->user[6] = 0.0;
    }
  for (i = 0; i < neq; i++) {
    y[i] = 1.0;
    dym[i] = 0.0;
  }
}
```

10.2 Program Data

```
nag_ode_bvp_coll_nlin_setup (d02tvc) Example Program Data
 5      11      100             : ncol, nmesh, mxmesh
 1575.0  0.01     0.0279   0.02 : beta0, eta, lambda, mu
 1.0e-5  1.0e-5   1.0e-5
 1.0e-5  1.0e-5   1.0e-5        : tols(1:neq=6)
```

10.3 Program Results

```
nag_ode_bvp_coll_nlin_setup (d02tvc) Example Program Results

(User-supplied callback guess, first invocation.)
(User-supplied callback gafun, first invocation.)
(User-supplied callback gajac, first invocation.)
(User-supplied callback gbfun, first invocation.)
(User-supplied callback gbjac, first invocation.)
(User-supplied callback ffun, first invocation.)
(User-supplied callback fjac, first invocation.)
 Used a mesh of   21  points
 Maximum error =   1.42e-08  in interval    5  for component    1


 Mesh points:
   1(1) 0.0000    2(3) 0.0500    3(2) 0.1000    4(3) 0.1500
   5(2) 0.2000    6(3) 0.2500    7(2) 0.3000    8(3) 0.3500
   9(2) 0.4000   10(3) 0.4500   11(2) 0.5000   12(3) 0.5500
```

```
 13(2) 0.6000   14(3) 0.6500   15(2) 0.7000   16(3) 0.7500
 17(2) 0.8000   18(3) 0.8500   19(2) 0.9000   20(3) 0.9500
 21(1) 1.0000

Computed solution at mesh points
   x       y1         y2         y3
0.000   7.523e-02  1.801e-05  4.981e-06
0.050   7.610e-02  7.889e-05  2.188e-05
0.100   7.655e-02  3.148e-04  8.917e-05
0.150   7.576e-02  1.009e-03  2.981e-04
0.200   7.264e-02  2.254e-03  7.135e-04
0.250   6.780e-02  3.114e-03  1.081e-03
0.300   6.407e-02  2.556e-03  9.842e-04
0.350   6.290e-02  1.285e-03  5.500e-04
0.400   6.335e-02  4.137e-04  1.969e-04
0.450   6.428e-02  9.118e-05  4.777e-05
0.500   6.527e-02  1.588e-05  8.806e-06
0.550   6.627e-02  2.773e-06  1.511e-06
0.600   6.727e-02  6.284e-07  3.130e-07
0.650   6.827e-02  2.186e-07  9.642e-08
0.700   6.927e-02  1.243e-07  4.870e-08
0.750   7.027e-02  1.156e-07  4.086e-08
0.800   7.127e-02  1.697e-07  5.511e-08
0.850   7.227e-02  3.704e-07  1.126e-07
0.900   7.327e-02  1.112e-06  3.220e-07
0.950   7.426e-02  4.197e-06  1.178e-06
1.000   7.523e-02  1.801e-05  4.981e-06
```