NAG Library Function Document

nag_ode_bvp_coll_nlin_contin (d02txc)

1 Purpose

nag_ode_bvp_coll_nlin_contin (d02txc) allows a solution to a nonlinear two-point boundary value problem computed by nag_ode_bvp_coll_nlin_solve (d02tlc) to be used as an initial approximation in the solution of a related nonlinear two-point boundary value problem in a continuation call to nag_ode_bvp_coll_nlin_solve (d02tlc).

2 Specification

```
#include <nag.h>
#include <nagd02.h>
void nag_ode_bvp_coll_nlin_contin (Integer mxmesh, Integer nmesh,
     const double mesh[], const Integer ipmesh[], double rcomm[],
     Integer icomm[], NagError *fail)
```

3 Description

nag_ode_bvp_coll_nlin_contin (d02txc) and its associated functions (nag_ode_bvp_coll_nlin_solve (d02tlc), nag_ode_bvp_coll_nlin_setup (d02tvc), 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

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

First, nag_ode_bvp_coll_nlin_setup (d02tvc) must be called to specify the initial mesh, error requirements and other details. Then, nag_ode_bvp_coll_nlin_solve (d02tlc) can be used to solve the boundary value problem. After successful computation, nag_ode_bvp_coll_nlin_diag (d02tzc) can be used to ascertain details about the final mesh. nag_ode_bvp_coll_nlin_interp (d02tyc) can be used to compute the approximate solution anywhere on the interval $[a, b]$ using interpolation.

If the boundary value problem being solved is one of a sequence of related problems, for example as part of some continuation process, then nag_ode_bvp_coll_nlin_contin (d02txc) should be used between calls to nag_ode_bvp_coll_nlin_solve (d02tlc). This avoids the overhead of a complete initialization when the setup function nag_ode_bvp_coll_nlin_setup (d02tvc) is used. nag_ode_bvp_coll_nlin_contin

(d02txc) allows the solution values computed in the previous call to nag_ode_bvp_coll_nlin_solve (d02tlc) to be used as an initial approximation for the solution in the next call to nag_ode_bvp_coll_nlin_solve (d02tlc).

You must specify the new initial mesh. The previous mesh can be obtained by a call to nag_ode_bvp_coll_nlin_diag (d02tzc). It may be used unchanged as the new mesh, in which case any fixed points in the previous mesh remain as fixed points in the new mesh. Fixed and other points may be added or subtracted from the mesh by manipulation of the contents of the array argument **ipmesh**. Initial values for the solution components on the new mesh are computed by interpolation on the values for the solution components on the previous mesh.

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

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

5 Arguments

1: **mxmesh** – Integer *Input*

On entry: the maximum number of points allowed in the mesh.

Constraint: this must be identical to the value supplied for the argument **mxmesh** in the prior call to nag_ode_bvp_coll_nlin_setup (d02tvc).

2: **nmesh** – Integer *Input*

On entry: the number of points to be used in the new initial mesh. It is strongly recommended that if this function is called that the suggested value (see below) for **nmesh** is used. In this case the arrays **mesh** and **ipmesh** returned by nag_ode_bvp_coll_nlin_diag (d02tzc) can be passed to this function without any modification.

Suggested value: $(n^* + 1)/2$, where n^* is the number of mesh points used in the previous mesh as returned in the argument **nmesh** of nag_ode_bvp_coll_nlin_diag (d02tzc).

Constraint: $6 \leq \mathbf{nmesh} \leq (\mathbf{mxmesh} + 1)/2$.

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

On entry: the **nmesh** points to be used in the new initial mesh as specified by **ipmesh**.

Suggested value: the argument **mesh** returned from a call to nag_ode_bvp_coll_nlin_diag (d02tzc).

Constraint: $\mathbf{mesh}[i_j - 1] < \mathbf{mesh}[i_{j+1} - 1]$, for $j = 1, 2, \ldots, \mathbf{nmesh} - 1$, the values of $i_1, i_2, \ldots, i_{\mathbf{nmesh}}$ are defined in **ipmesh**.

$\mathbf{mesh}[i_1 - 1]$ must contain the left boundary point, a, and $\mathbf{mesh}[i_{\mathbf{nmesh}} - 1]$ must contain the right boundary point, b, as specified in the previous call to nag_ode_bvp_coll_nlin_setup (d02tvc).

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

On entry: specifies the points in **mesh** to be used as the new initial mesh. Let $\{i_j : j = 1, 2, \ldots, \mathbf{nmesh}\}$ be the set of array indices of **ipmesh** such that **ipmesh**$[i_j - 1] = 1$ or 2 and $1 = i_1 < i_2 < \cdots < i_{\mathbf{nmesh}}$. Then **mesh**$[i_j - 1]$ will be included in the new initial mesh.

If **ipmesh**$[i_j - 1] = 1$, **mesh**$[i_j - 1]$ will be a fixed point in the new initial mesh.

If **ipmesh**$[k - 1] = 3$ for any k, then **mesh**$[k - 1]$ will not be included in the new mesh.

Suggested value: the argument **ipmesh** returned in a call to nag_ode_bvp_coll_nlin_diag (d02tzc).

Constraints:

ipmesh$[k - 1] = 1$, 2 or 3, for $k = 1, 2, \ldots, i_{\mathbf{nmesh}}$;
ipmesh$[0] =$ **ipmesh**$[i_{\mathbf{nmesh}} - 1] = 1$.

5: **rcomm**$[dim]$ – double *Communication Array*

Note: the dimension, dim, of this array is dictated by the requirements of associated functions that must have been previously called. This array MUST be the same array passed as argument **rcomm** in the previous call to nag_ode_bvp_coll_nlin_solve (d02tlc).

On entry: this must be the same array as supplied to nag_ode_bvp_coll_nlin_solve (d02tlc) and **must** remain unchanged between calls.

On exit: contains information about the solution for use on subsequent calls to associated functions.

6: **icomm**$[dim]$ – Integer *Communication Array*

Note: the dimension, dim, of this array is dictated by the requirements of associated functions that must have been previously called. This array MUST be the same array passed as argument **icomm** in the previous call to nag_ode_bvp_coll_nlin_solve (d02tlc).

On entry: this must be the same array as supplied to nag_ode_bvp_coll_nlin_solve (d02tlc) and **must** remain unchanged between calls.

On exit: contains information about the solution for use on subsequent calls to associated functions.

7: **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_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

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

NE_CONVERGENCE_SOL

The solver function did not produce any results suitable for remeshing.

NE_INT

An element of **ipmesh** was set to -1 before **nmesh** elements containing 1 or 2 were detected.

ipmesh$[i] \neq -1$, 1, 2 or 3 for some i.

On entry, **ipmesh**$[0] = \langle value \rangle$.
Constraint: **ipmesh**$[0] = 1$.

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

You have set the element of **ipmesh** corresponding to the last element of **mesh** to be included in the new mesh as $\langle value \rangle$, which is not 1.

NE_INT_2

On entry, **nmesh** $= \langle value \rangle$ and **mxmesh** $= \langle value \rangle$.
Constraint: **nmesh** $\leq$ (**mxmesh** $+ 1)/2$.

NE_INT_CHANGED

On entry, **mxmesh** $= \langle value \rangle$ and **mxmesh** $= \langle value \rangle$ in nag_ode_bvp_coll_nlin_setup (d02tvc).
Constraint: **mxmesh** $=$ **mxmesh** in nag_ode_bvp_coll_nlin_setup (d02tvc).

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.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in How to Use the NAG Library and its Documentation for further information.

NE_MESH_ERROR

The first element of array **mesh** does not coincide with the left hand end of the range previously specified.
First element of **mesh**: $\langle value \rangle$; left hand of the range: $\langle value \rangle$.

The last point of the new mesh does not coincide with the right hand end of the range previously specified.
Last point of the new mesh: $\langle value \rangle$; right hand end of the range: $\langle value \rangle$.

NE_MISSING_CALL

The solver function does not appear to have been called.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

NE_NOT_STRICTLY_INCREASING

The entries in **mesh** are not strictly increasing.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_ode_bvp_coll_nlin_contin (d02txc) is not threaded in any implementation.

9 Further Comments

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

– cluster the 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.

In the absence of any other information about the expected behaviour of the solution, using the values suggested in Section 5 for **nmesh**, **ipmesh** and **mesh** is strongly recommended.

10 Example

This example illustrates the use of continuation, solution on an infinite range, and solution of a system of two differential equations of orders 3 and 2. See also nag_ode_bvp_coll_nlin_solve (d02tlc), nag_ode_bvp_coll_nlin_setup (d02tvc), nag_ode_bvp_coll_nlin_interp (d02tyc) and nag_ode_bvp_coll_nlin_diag (d02tzc), for the illustration of other facilities.

Consider the problem of swirling flow over an infinite stationary disk with a magnetic field along the axis of rotation. See Ascher *et al.* (1988) and the references therein. After transforming from a cylindrical coordinate system (r, θ, z), in which the θ component of the corresponding velocity field behaves like r^{-n}, the governing equations are

$$\begin{aligned} f''' + \tfrac{1}{2}(3-n)ff'' + n(f')^2 + g^2 - sf' &= \gamma^2 \\ g'' + \tfrac{1}{2}(3-n)fg' + (n-1)gf' - s(g-1) &= 0 \end{aligned}$$

with boundary conditions

$$f(0) = f'(0) = g(0) = 0, \quad f'(\infty) = 0, \quad g(\infty) = \gamma,$$

where s is the magnetic field strength, and γ is the Rossby number.

Some solutions of interest are for $\gamma = 1$, small n and $s \rightarrow 0$. An added complication is the infinite range, which we approximate by $[0, L]$. We choose $n = 0.2$ and first solve for $L = 60.0, s = 0.24$ using the initial approximations $f(x) = -x^2e^{-x}$ and $g(x) = 1.0 - e^{-x}$, which satisfy the boundary conditions, on a uniform mesh of 21 points. Simple continuation on the parameters L and s using the values $L = 120.0, s = 0.144$ and then $L = 240.0, s = 0.0864$ is used to compute further solutions. We use the suggested values for **nmesh**, **ipmesh** and **mesh** in the call to nag_ode_bvp_coll_nlin_contin (d02txc) prior to a continuation call, that is only every second point of the preceding mesh is used.

The equations are first mapped onto $[0, 1]$ to yield

$$\begin{aligned} f''' &= L^3\left(\gamma^2 - g^2\right) + L^2 sf' - L\left(\tfrac{1}{2}(3-n)ff'' + n(f')^2\right) \\ g'' &= L^2 s(g-1) - L\left(\tfrac{1}{2}(3-n)fg' + (n-1)f'g\right). \end{aligned}$$

10.1 Program Text

```
/* nag_ode_bvp_coll_nlin_contin (d02txc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

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

```
typedef struct
{
  double el, en, s;
} 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)
{
  static double ruser[7] = { -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0 };
  Integer exit_status = 0;
  Integer neq, mmax, nlbc, nrbc, nleft, nright;
  Integer i, iermx, ijermx, j, licomm, lrcomm, mxmesh, ncol, ncont, nmesh;
  double xsplit = 30.0;
  double dx, el, el_init, en, ermx, s, s_init, xx;
  double *mesh = 0, *rcomm = 0;
  double *tol = 0, *y = 0;
  Integer *ipmesh = 0, *icomm = 0, *m = 0;
  func_data fd;
  Nag_Comm comm;
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_ode_bvp_coll_nlin_contin (d02txc) Example Program Results\n\n");

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

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "", &neq, &mmax);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "", &neq, &mmax);
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &nlbc, &nrbc);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &nlbc, &nrbc);
#endif
#ifdef _WIN32
```

```
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &nleft, &nright);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &nleft, &nright);
#endif
  /* Read method parameters */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &ncol, &nmesh,
          &mxmesh);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &ncol, &nmesh,
        &mxmesh);
#endif
  licomm = mxmesh * (11 * neq + 6);
  lrcomm = mxmesh * (109 * pow((double) neq, 2) + 78 * neq + 7);
  /* Allocate memory */
  if (!(tol = NAG_ALLOC(neq, double)) ||
      !(y = NAG_ALLOC(neq * mmax, double)) ||
      !(m = NAG_ALLOC(neq, Integer)) ||
      !(mesh = NAG_ALLOC(mxmesh, double)) ||
      !(rcomm = NAG_ALLOC(lrcomm, double)) ||
      !(ipmesh = NAG_ALLOC(mxmesh, Integer)) ||
      !(icomm = NAG_ALLOC(licomm, Integer))
        )
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (i = 0; i < neq; i++) {
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &m[i]);
#else
    scanf("%" NAG_IFMT "", &m[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
  for (i = 0; i < neq; i++) {
#ifdef _WIN32
    scanf_s("%lf", &tol[i]);
#else
    scanf("%lf", &tol[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
  /* Read problem (initial) parameters */
#ifdef _WIN32
  scanf_s("%lf%lf%lf%*[^\n] ", &en, &el_init, &s_init);
#else
  scanf("%lf%lf%lf%*[^\n] ", &en, &el_init, &s_init);
#endif
  /* Initialize data */
  el = el_init;
  s = s_init;

  /* Set data required for the user-supplied functions */
  fd.el = el;
  fd.en = en;
  fd.s = s;
  /* Associate the data structure with comm.p */
  comm.p = (Pointer) &fd;

  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;

  /* 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, tol, 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 = 1;
    goto END;
  }

  /* Initialize number of continuation steps in el and s */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n] ", &ncont);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &ncont);
#endif

  for (j = 0; j < ncont; j++) {
    printf("\n Tolerance = %8.1e", tol[0]);
    printf("  l = %8.3f  s =%7.4f\n", el, s);
    /* 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 = 2;
      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 = 3;
      goto END;
    }

    printf("\n Used a mesh of %4" NAG_IFMT "  points\n", nmesh);
    printf(" Maximum error = %10.2e", ermx);
    printf("  in interval %4" NAG_IFMT " ", iermx);
    printf(" for component %4" NAG_IFMT " \n", ijermx);
    /* Print solution components on mesh */
    printf("\n\n Solution on original interval:\n     x        f          g\n");
    /* Left side domain [0,xsplit], evaluate at nleft+1 uniform grid points. */
```

```
    dx = xsplit / (double) (nleft) / el;
    xx = 0.0;
    for (i = 0; i <= nleft; 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(xx, 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("%8.2f %10.4f %10.4f \n", xx * el, Y(1, 0), Y(2, 0));
      xx = xx + dx;
    }
    /* Right side domain (xsplit,L], evaluate at nright uniform grid points. */
    dx = (el - xsplit) / (double) (nright) / el;
    xx = xsplit / el;
    for (i = 0; i < nright; i++) {
      xx = MIN(1.0, xx + dx);

      /* 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(xx, 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 = 6;
        goto END;
      }

      printf("%8.2f %10.4f %10.4f \n", xx * el, Y(1, 0), Y(2, 0));
    }
    /* Select mesh for continuation and update continuation parameters. */
    if (j < ncont - 1) {
      el = 2.0 * el;
      s = 0.6 * s;
      fd.el = el;
      fd.s = s;
      nmesh = (nmesh + 1) / 2;

      /* nag_ode_bvp_coll_nlin_contin (d02txc).
       * Ordinary differential equations, general nonlinear boundary value
       * problem, continuation facility for
       * nag_ode_bvp_coll_nlin_solve (d02tlc).
       */
      nag_ode_bvp_coll_nlin_contin(mxmesh, nmesh, mesh, ipmesh, rcomm, icomm,
                                   &fail);
      if (fail.code != NE_NOERROR) {
        printf("Error from nag_ode_bvp_coll_nlin_contin (d02txc).\n%s\n",
               fail.message);
        exit_status = 7;
        goto END;
      }
    }
  }

END:
  NAG_FREE(mesh);
  NAG_FREE(m);
  NAG_FREE(tol);
  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 el, en, s, t1, y11, y20;
  double half = 0.5;
  double one = 1.0;
  double three = 3.0;

  if (comm->user[0] == -1.0) {
    printf("(User-supplied callback ffun, first invocation.)\n");
    comm->user[0] = 0.0;
  }
  el = fd->el;
  en = fd->en;
  s = fd->s;
  t1 = half * (three - en) * Y(1, 0);
  y11 = Y(1, 1);
  y20 = Y(2, 0);
  f[0] = (pow(el, 3)) * (one - pow(y20, 2)) + (pow(el, 2)) * s * y11 -
         el * (t1 * Y(1, 2) + en * pow(y11, 2));
  f[1] = (pow(el, 2)) * s * (y20 - one) - el *
         (t1 * Y(2, 1) + (en - one) * y11 * y20);
}

static void NAG_CALL fjac(double x, const double y[], Integer neq,
                          const Integer m[], double dfdy[], Nag_Comm *comm)
{
#define DFDY(I, J, K) dfdy[I-1 + (J-1)* neq + K * neq * neq]
  func_data *fd = (func_data *) comm->p;
  double el, en, s;
  double half = 0.5;
  double one = 1.0;
  double two = 2.0;
  double three = 3.0;

  if (comm->user[1] == -1.0) {
    printf("(User-supplied callback fjac, first invocation.)\n");
    comm->user[1] = 0.0;
  }
  el = fd->el;
  en = fd->en;
  s = fd->s;
  DFDY(1, 2, 0) = -two * pow(el, 3) * Y(2, 0);
  DFDY(1, 1, 0) = -el * half * (three - en) * Y(1, 2);
  DFDY(1, 1, 1) = pow(el, 2) * s - el * two * en * Y(1, 1);
  DFDY(1, 1, 2) = -el * half * (three - en) * Y(1, 0);
  DFDY(2, 2, 0) = pow(el, 2) * s - el * (en - one) * Y(1, 1);
  DFDY(2, 2, 1) = -el * half * (three - en) * Y(1, 0);
  DFDY(2, 1, 0) = -el * half * (three - en) * Y(2, 1);
  DFDY(2, 1, 1) = -el * (en - one) * Y(2, 0);
}

static void NAG_CALL gafun(const double ya[], Integer neq, const Integer m[],
                           Integer nlbc, double ga[], Nag_Comm *comm)
{
#define YA(I, J) ya[J * neq + I-1]

  if (comm->user[2] == -1.0) {
    printf("(User-supplied callback gafun, first invocation.)\n");
    comm->user[2] = 0.0;
  }
  ga[0] = YA(1, 0);
  ga[1] = YA(1, 1);
  ga[2] = YA(2, 0);
}

static void NAG_CALL gbfun(const double yb[], Integer neq, const Integer m[],
```

```
                           Integer nrbc, double gb[], Nag_Comm *comm)
{
#define YB(I, J) yb[J * neq + I-1]

  if (comm->user[3] == -1.0) {
    printf("(User-supplied callback gbfun, first invocation.)\n");
    comm->user[3] = 0.0;
  }
  gb[0] = YB(1, 1);
  gb[1] = YB(2, 0) - 1.0;
}

static void NAG_CALL gajac(const double ya[], Integer neq, const Integer m[],
                           Integer nlbc, double dgady[], Nag_Comm *comm)
{
#define DGADY(I, J, K) dgady[I-1 + (J-1)* nlbc + K * nlbc * neq]
  double one = 1.0;

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

static void NAG_CALL gbjac(const double yb[], Integer neq, const Integer m[],
                           Integer nrbc, double dgbdy[], Nag_Comm *comm)
{
#define DGBDY(I, J, K) dgbdy[I-1 + (J-1)* nrbc + K * nrbc * neq]
  double one = 1.0;

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

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

  if (comm->user[6] == -1.0) {
    printf("(User-supplied callback guess, first invocation.)\n");
    comm->user[6] = 0.0;
  }
  ex = x * fd->el;
  expmx = exp(-ex);
  Y(1, 0) = -pow(ex, 2) * expmx;
  Y(1, 1) = (-two * ex + pow(ex, 2)) * expmx;
  Y(1, 2) = (-two + 4.0 * ex - pow(ex, 2)) * expmx;
  Y(2, 0) = one - expmx;
  Y(2, 1) = expmx;
  dym[0] = (6.0 - 6.0 * ex + pow(ex, 2)) * expmx;
  dym[1] = -expmx;
}
```

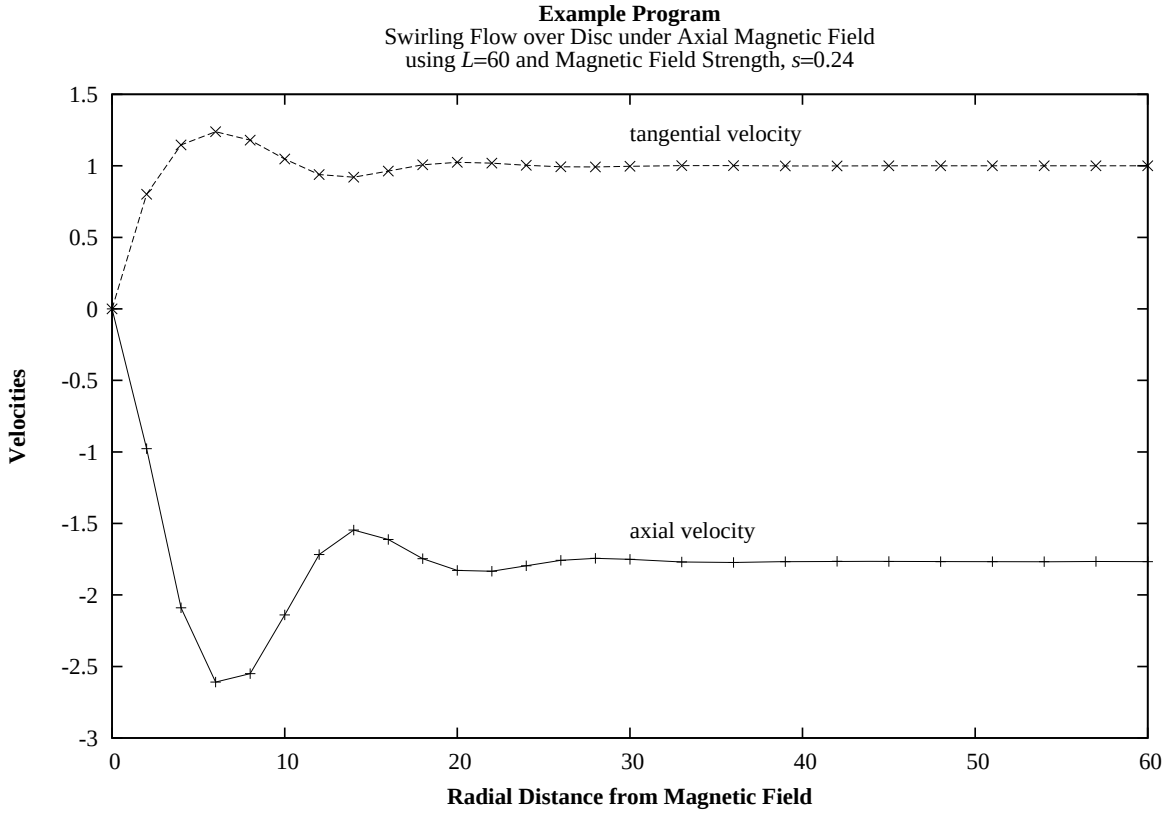
10.2 Program Data

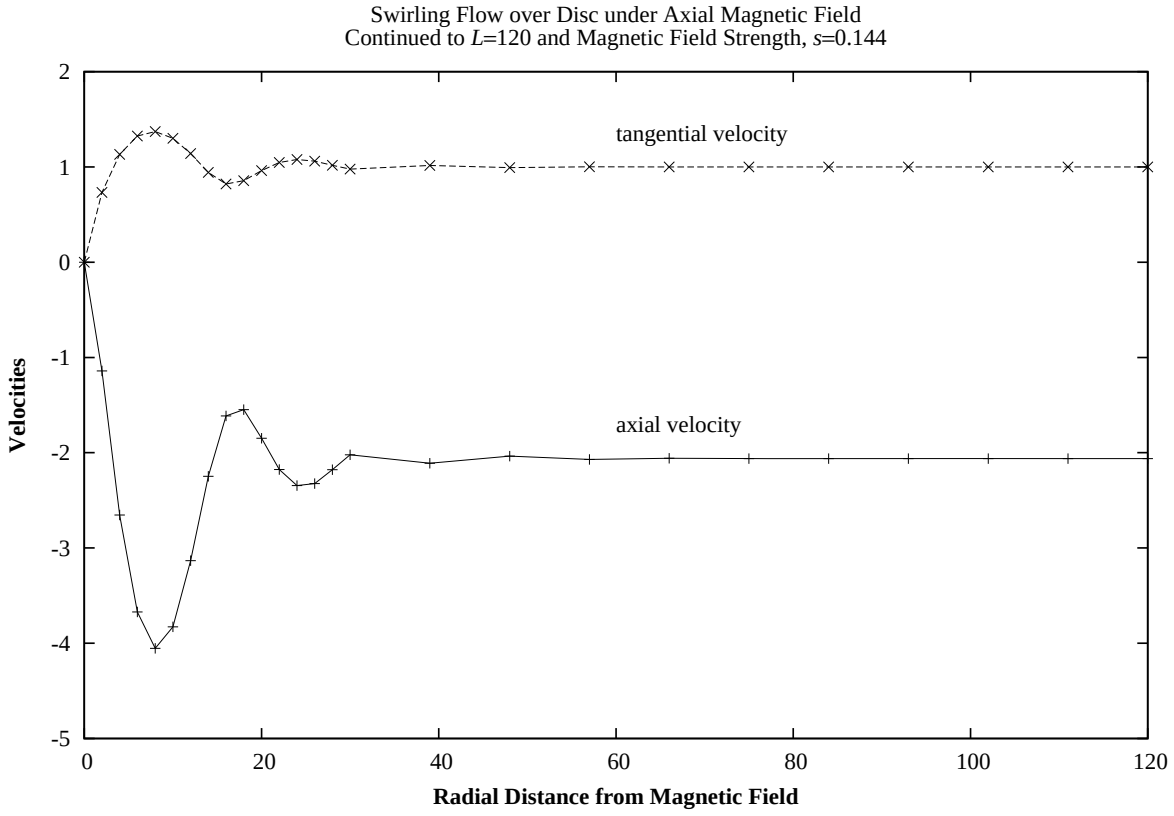
```
nag_ode_bvp_coll_nlin_contin (d02txc) Example Program Data
   2  3  3  2    : neq, mmax, nlbc, nrbc
   15 10         : nleft, nright
   6  21  250    : (method parameters) ncol, nmesh, mxmesh
   3  2          : m
   1.0e-5 1.0e-5 : tolerances
   0.2 60.0 0.24 : (problem parameters) en, el_init, s_init
   3             : ncont
```

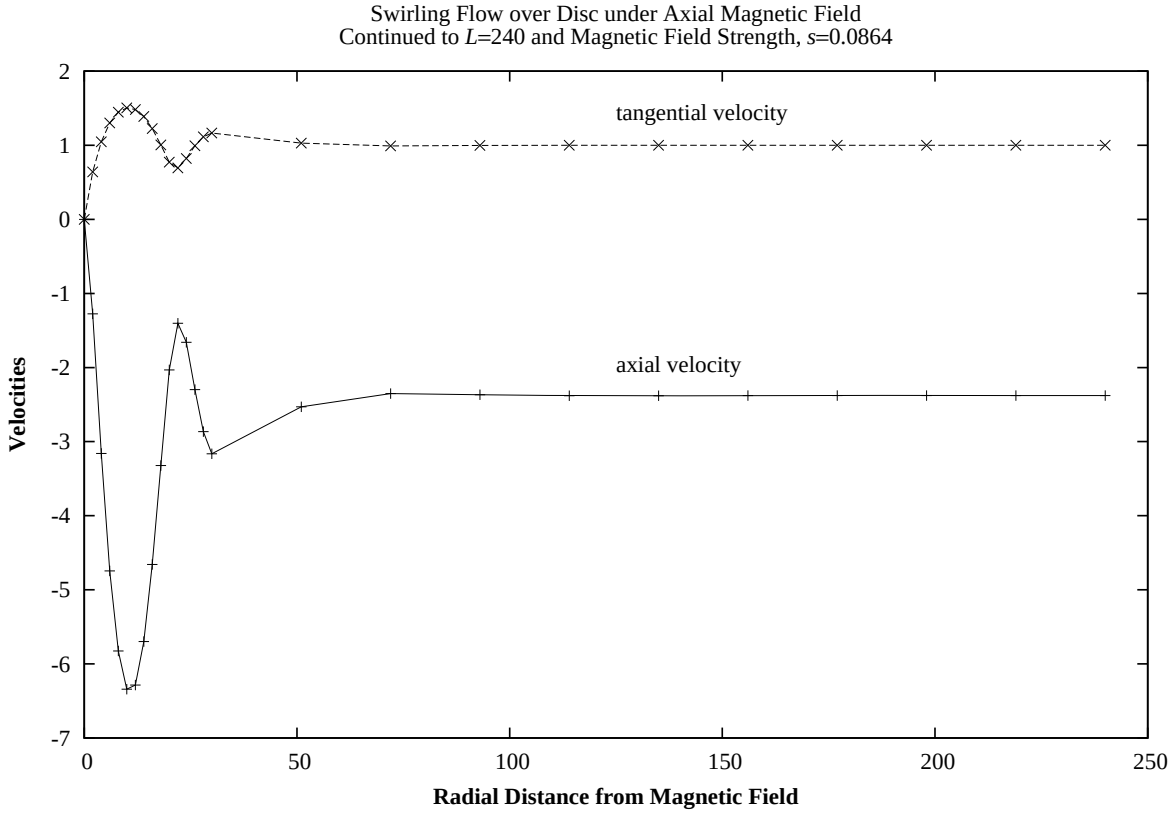
10.3 Program Results

```
nag_ode_bvp_coll_nlin_contin (d02txc) Example Program Results


 Tolerance =  1.0e-05  l =   60.000  s = 0.2400
(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 =   2.66e-08  in interval    7  for component    1


 Solution on original interval:
     x        f          g
    0.00     0.0000     0.0000
    2.00    -0.9769     0.8011
    4.00    -2.0900     1.1459
    6.00    -2.6093     1.2389
    8.00    -2.5498     1.1794
   10.00    -2.1397     1.0478
   12.00    -1.7176     0.9395
   14.00    -1.5465     0.9206
   16.00    -1.6127     0.9630
   18.00    -1.7466     1.0068
   20.00    -1.8286     1.0244
   22.00    -1.8338     1.0185
   24.00    -1.7956     1.0041
   26.00    -1.7582     0.9940
   28.00    -1.7445     0.9926
   30.00    -1.7515     0.9965
   33.00    -1.7695     1.0019
   36.00    -1.7730     1.0018
   39.00    -1.7673     0.9998
   42.00    -1.7645     0.9993
   45.00    -1.7659     0.9999
   48.00    -1.7672     1.0002
   51.00    -1.7671     1.0001
   54.00    -1.7666     0.9999
   57.00    -1.7665     0.9999
   60.00    -1.7666     1.0000

 Tolerance =  1.0e-05  l =  120.000  s = 0.1440

 Used a mesh of   21  points
 Maximum error =   6.88e-06  in interval    7  for component    2


 Solution on original interval:
     x        f          g
    0.00     0.0000     0.0000
    2.00    -1.1406     0.7317
    4.00    -2.6531     1.1315
    6.00    -3.6721     1.3250
```

```
   8.00    -4.0539     1.3707
  10.00    -3.8285     1.3003
  12.00    -3.1339     1.1407
  14.00    -2.2469     0.9424
  16.00    -1.6146     0.8201
  18.00    -1.5472     0.8549
  20.00    -1.8483     0.9623
  22.00    -2.1761     1.0471
  24.00    -2.3451     1.0778
  26.00    -2.3236     1.0600
  28.00    -2.1784     1.0165
  30.00    -2.0214     0.9775
  39.00    -2.1109     1.0155
  48.00    -2.0362     0.9931
  57.00    -2.0709     1.0023
  66.00    -2.0588     0.9995
  75.00    -2.0616     1.0000
  84.00    -2.0615     1.0001
  93.00    -2.0611     0.9999
 102.00    -2.0614     1.0000
 111.00    -2.0613     1.0000
 120.00    -2.0613     1.0000

Tolerance =  1.0e-05  l =  240.000  s = 0.0864

Used a mesh of   81  points
Maximum error =   3.30e-07  in interval   19  for component    2


Solution on original interval:
    x         f          g
   0.00     0.0000     0.0000
   2.00    -1.2756     0.6404
   4.00    -3.1604     1.0463
   6.00    -4.7459     1.3011
   8.00    -5.8265     1.4467
  10.00    -6.3412     1.5036
  12.00    -6.2862     1.4824
  14.00    -5.6976     1.3886
  16.00    -4.6568     1.2263
  18.00    -3.3226     1.0042
  20.00    -2.0328     0.7718
  22.00    -1.4035     0.6943
  24.00    -1.6603     0.8218
  26.00    -2.2975     0.9928
  28.00    -2.8661     1.1139
  30.00    -3.1641     1.1641
  51.00    -2.5307     1.0279
  72.00    -2.3520     0.9919
  93.00    -2.3674     0.9975
 114.00    -2.3799     1.0003
 135.00    -2.3800     1.0002
 156.00    -2.3792     1.0000
 177.00    -2.3791     1.0000
 198.00    -2.3792     1.0000
 219.00    -2.3792     1.0000
 240.00    -2.3792     1.0000
```


Example Program
Swirling Flow over Disc under Axial Magnetic Field
using L=60 and Magnetic Field Strength, s=0.24
tangential velocity
axial velocity
Velocities
Radial Distance from Magnetic Field
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
0
10
20
30
40
50
60


Swirling Flow over Disc under Axial Magnetic Field
Continued to L=120 and Magnetic Field Strength, s=0.144
tangential velocity
axial velocity
Velocities
Radial Distance from Magnetic Field
2
1
0
-1
-2
-3
-4
-5
0
20
40
60
80
100
120


Swirling Flow over Disc under Axial Magnetic Field
Continued to L=240 and Magnetic Field Strength, s=0.0864
tangential velocity
axial velocity
Velocities
Radial Distance from Magnetic Field
2
1
0
-1
-2
-3
-4
-5
-6
-7
0
50
100
150
200
250