NAG Toolbox

nag_ode_bvp_coll_nlin_solve (d02tl)

1 Purpose

nag_ode_bvp_coll_nlin_solve (d02tl) solves a general two-point boundary value problem for a nonlinear mixed order system of ordinary differential equations.

2 Syntax

```
[rcomm, icomm, user, ifail] = nag_ode_bvp_coll_nlin_solve(ffun, fjac, gafun, gbfun, gajac, gbjac, guess, rcomm, icomm, 'user', user)
[rcomm, icomm, user, ifail] = d02tl(ffun, fjac, gafun, gbfun, gajac, gbjac, guess, rcomm, icomm, 'user', user)
```

3 Description

nag_ode_bvp_coll_nlin_solve (d02tl) and its associated functions (nag_ode_bvp_coll_nlin_setup (d02tv), nag_ode_bvp_coll_nlin_contin (d02tx), nag_ode_bvp_coll_nlin_interp (d02ty) and nag_ode_bvp_coll_nlin_diag (d02tz)) solve the two-point boundary value problem for a nonlinear mixed order 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 (d02tv) must be called to specify the initial mesh, error requirements and other details. Note that the error requirements apply only to the solution components $y_1, y_2, \ldots, y_n$ and that no error control is applied to derivatives of solution components. (If error control is required on derivatives then the system must be reduced in order by introducing the derivatives whose error is to be controlled as new variables. See Section 9 in nag_ode_bvp_coll_nlin_setup (d02tv).) Then, nag_ode_bvp_coll_nlin_solve (d02tl) can be used to solve the boundary value problem. After successful computation, nag_ode_bvp_coll_nlin_diag (d02tz) can be used to ascertain details about the final mesh and other details of the solution procedure, and nag_ode_bvp_coll_nlin_interp (d02ty) can be used to compute the approximate solution anywhere on the interval $[a, b]$.

A description of the numerical technique used in nag_ode_bvp_coll_nlin_solve (d02tl) is given in Section 3 in nag_ode_bvp_coll_nlin_setup (d02tv).

nag_ode_bvp_coll_nlin_solve (d02tl) can also be used in the solution of a series of problems, for example in performing continuation, when the mesh used to compute the solution of one problem is to be used as the initial mesh for the solution of the next related problem. nag_ode_bvp_coll_nlin_contin (d02tx) should be used in between calls to nag_ode_bvp_coll_nlin_solve (d02tl) in this context.

See Section 9 in nag_ode_bvp_coll_nlin_setup (d02tv) for details of how to solve boundary value problems of a more general nature.

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 Parameters

5.1 Compulsory Input Parameters

1: **ffun** – SUBROUTINE, supplied by the user.

ffun must evaluate the functions f_i for given values $x, z(y(x))$.

```
[f, user] = ffun(x, y, neq, m, user)
```

Input Parameters

1: **x** – REAL (KIND=nag_wp)

x, the independent variable.

2: **y**(**neq**, $0:*$) – REAL (KIND=nag_wp) array

y(i,j) contains $y_i^{(j)}(x)$, for $i = 1, 2, \ldots,$ **neq** and $j = 0, 1, \ldots,$ **m**$(i) - 1$.
Note: $y_i^{(0)}(x) = y_i(x)$.

3: **neq** – INTEGER

The number of differential equations.

4: **m**(**neq**) – INTEGER array

m(i) contains m_i, the order of the ith differential equation, for $i = 1, 2, \ldots,$ **neq**.

5: **user** – INTEGER array

ffun is called from nag_ode_bvp_coll_nlin_solve (d02tl) with the object supplied to nag_ode_bvp_coll_nlin_solve (d02tl).

Output Parameters

1: **f**(**neq**) – REAL (KIND=nag_wp) array

f(i) must contain f_i, for $i = 1, 2, \ldots,$ **neq**.

2: **user** – INTEGER array

2: **fjac** – SUBROUTINE, supplied by the user.

fjac must evaluate the partial derivatives of f_i with respect to the elements of $z(y(x)) = \left(y_1(x), y_1^1(x), \ldots, y_1^{(m_1-1)}(x), y_2(x), \ldots, y_n^{(m_n-1)}(x)\right)$.

```
[dfdy, user] = fjac(x, y, neq, m, dfdy, user)
```

Input Parameters

1: **x** – REAL (KIND=nag_wp)

x, the independent variable.

2: **y**(**neq**, $0 : *$) – REAL (KIND=nag_wp) array

y(i, j) contains $y_i^{(j)}(x)$, for $i = 1, 2, \ldots,$ **neq** and $j = 0, 1, \ldots,$ **m**$(i) - 1$.
Note: $y_i^{(0)}(x) = y_i(x)$.

3: **neq** – INTEGER

The number of differential equations.

4: **m**(**neq**) – INTEGER array

m(i) contains m_i, the order of the ith differential equation, for $i = 1, 2, \ldots,$ **neq**.

5: **dfdy**(**neq**, **neq**, $0 : *$) – REAL (KIND=nag_wp) array

Set to zero.

6: **user** – INTEGER array

fjac is called from nag_ode_bvp_coll_nlin_solve (d02tl) with the object supplied to nag_ode_bvp_coll_nlin_solve (d02tl).

Output Parameters

1: **dfdy**(**neq**, **neq**, $0 : *$) – REAL (KIND=nag_wp) array

dfdy(i, j, k) must contain the partial derivative of f_i with respect to $y_j^{(k)}$, for $i = 1, 2, \ldots,$ **neq**, $j = 1, 2, \ldots,$ **neq** and $k = 0, 1, \ldots,$ **m**$(j) - 1$. Only nonzero partial derivatives need be set.

2: **user** – INTEGER array

3: **gafun** – SUBROUTINE, supplied by the user.

gafun must evaluate the boundary conditions at the left-hand end of the range, that is functions $g_i(z(y(a)))$ for given values of $z(y(a))$.

```
[ga, user] = gafun(ya, neq, m, nlbc, user)
```

Input Parameters

1: **ya**(**neq**, $0:*$) – REAL (KIND=nag_wp) array

ya(i,j) contains $y_i^{(j)}(a)$, for $i = 1,2,\ldots,$**neq** and $j = 0,1,\ldots,$**m**$(i)-1$.
Note: $y_i^{(0)}(a) = y_i(a)$.

2: **neq** – INTEGER

The number of differential equations.

3: **m**(**neq**) – INTEGER array

m(i) contains m_i, the order of the ith differential equation, for $i = 1,2,\ldots,$**neq**.

4: **nlbc** – INTEGER

The number of boundary conditions at a.

5: **user** – INTEGER array

gafun is called from nag_ode_bvp_coll_nlin_solve (d02tl) with the object supplied to nag_ode_bvp_coll_nlin_solve (d02tl).

Output Parameters

1: **ga**(**nlbc**) – REAL (KIND=nag_wp) array

ga(i) must contain $g_i(z(y(a)))$, for $i = 1,2,\ldots,$**nlbc**.

2: **user** – INTEGER array

4: **gbfun** – SUBROUTINE, supplied by the user.

gbfun must evaluate the boundary conditions at the right-hand end of the range, that is functions $\bar{g}_i(z(y(b)))$ for given values of $z(y(b))$.

```
[gb, user] = gbfun(yb, neq, m, nrbc, user)
```

Input Parameters

1: **yb**(**neq**, $0:*$) – REAL (KIND=nag_wp) array

yb(i,j) contains $y_i^{(j)}(b)$, for $i = 1,2,\ldots,$**neq** and $j = 0,1,\ldots,$**m**$(i)-1$.
Note: $y_i^{(0)}(b) = y_i(b)$.

2: **neq** – INTEGER

The number of differential equations.

3: **m**(**neq**) – INTEGER array

m(i) contains m_i, the order of the ith differential equation, for $i = 1,2,\ldots,$**neq**.

4: **nrbc** – INTEGER

The number of boundary conditions at b.

5: **user** – INTEGER array

gbfun is called from nag_ode_bvp_coll_nlin_solve (d02tl) with the object supplied to nag_ode_bvp_coll_nlin_solve (d02tl).

Output Parameters

1: **gb**(**nrbc**) – REAL (KIND=nag_wp) array

gb(i) must contain $\bar{g}_i(z(y(b)))$, for $i = 1, 2, \ldots,$ **nrbc**.

2: **user** – INTEGER array

5: **gajac** – SUBROUTINE, supplied by the user.

gajac must evaluate the partial derivatives of $g_i(z(y(a)))$ with respect to the elements of $z(y(a)) = \left(y_1(a), y_1^1(a), \ldots, y_1^{(m_1-1)}(a), y_2(a), \ldots, y_n^{(m_n-1)}(a)\right)$.

```
[dgady, user] = gajac(ya, neq, m, nlbc, dgady, user)
```

Input Parameters

1: **ya**(**neq**, $0 : *$) – REAL (KIND=nag_wp) array

ya(i, j) contains $y_i^{(j)}(a)$, for $i = 1, 2, \ldots,$ **neq** and $j = 0, 1, \ldots,$ **m**$(i) - 1$.
Note: $y_i^{(0)}(a) = y_i(a)$.

2: **neq** – INTEGER

The number of differential equations.

3: **m**(**neq**) – INTEGER array

m(i) contains m_i, the order of the ith differential equation, for $i = 1, 2, \ldots,$ **neq**.

4: **nlbc** – INTEGER

The number of boundary conditions at a.

5: **dgady**(**nlbc**, **neq**, $0 : *$) – REAL (KIND=nag_wp) array

Set to zero.

6: **user** – INTEGER array

gajac is called from nag_ode_bvp_coll_nlin_solve (d02tl) with the object supplied to nag_ode_bvp_coll_nlin_solve (d02tl).

Output Parameters

1: **dgady**(**nlbc**, **neq**, $0 : *$) – REAL (KIND=nag_wp) array

dgady(i, j, k) must contain the partial derivative of $g_i(z(y(a)))$ with respect to $y_j^{(k)}(a)$, for $i = 1, 2, \ldots,$ **nlbc**, $j = 1, 2, \ldots,$ **neq** and $k = 0, 1, \ldots,$ **m**$(j) - 1$. Only nonzero partial derivatives need be set.

2: **user** – INTEGER array

6: **gbjac** – SUBROUTINE, supplied by the user.

gbjac must evaluate the partial derivatives of $\bar{g}_i(z(y(b)))$ with respect to the elements of $z(y(b)) = \left(y_1(b), y_1^1(b), \ldots, y_1^{(m_1-1)}(b), y_2(b), \ldots, y_n^{(m_n-1)}(b)\right)$.

```
[dgbdy, user] = gbjac(yb, neq, m, nrbc, dgbdy, user)
```

Input Parameters

1: **yb**$(\mathbf{neq}, 0:*)$ – REAL (KIND=nag_wp) array

yb(i,j) contains $y_i^{(j)}(b)$, for $i = 1,2,\ldots,\mathbf{neq}$ and $j = 0,1,\ldots,\mathbf{m}(i)-1$.
Note: $y_i^{(0)}(b) = y_i(b)$.

2: **neq** – INTEGER

The number of differential equations.

3: **m**$(\mathbf{neq})$ – INTEGER array

m(i) contains m_i, the order of the ith differential equation, for $i = 1,2,\ldots,\mathbf{neq}$.

4: **nrbc** – INTEGER

The number of boundary conditions at b.

5: **dgbdy**$(\mathbf{nrbc}, \mathbf{neq}, 0:*)$ – REAL (KIND=nag_wp) array

Set to zero.

6: **user** – INTEGER array

gbjac is called from nag_ode_bvp_coll_nlin_solve (d02tl) with the object supplied to nag_ode_bvp_coll_nlin_solve (d02tl).

Output Parameters

1: **dgbdy**$(\mathbf{nrbc}, \mathbf{neq}, 0:*)$ – REAL (KIND=nag_wp) array

dgbdy$(i,j,k+1)$ must contain the partial derivative of $\bar{g}_i(z(y(b)))$ with respect to $y_j^{(k)}(b)$, for $i = 1,2,\ldots,\mathbf{nrbc}$, $j = 1,2,\ldots,\mathbf{neq}$ and $k = 0,1,\ldots,\mathbf{m}(j)-1$. Only nonzero partial derivatives need be set.

2: **user** – INTEGER array

7: **guess** – SUBROUTINE, supplied by the user.

guess must return initial approximations for the solution components $y_i^{(j)}$ and the derivatives $y_i^{(m_i)}$, for $i = 1,2,\ldots,\mathbf{neq}$ and $j = 0,1,\ldots,\mathbf{m}(i)-1$. Try to compute each derivative $y_i^{(m_i)}$ such that it corresponds to your approximations to $y_i^{(j)}$, for $j = 0,1,\ldots,\mathbf{m}(i)-1$. You should **not** call **ffun** to compute $y_i^{(m_i)}$.

If nag_ode_bvp_coll_nlin_solve (d02tl) is being used in conjunction with nag_ode_bvp_coll_nlin_contin (d02tx) as part of a continuation process, then **guess** is not called by nag_ode_bvp_coll_nlin_solve (d02tl) after the call to nag_ode_bvp_coll_nlin_contin (d02tx).

```
[y, dym, user] = guess(x, neq, m, user)
```

Input Parameters

1: **x** – REAL (KIND=nag_wp)

x, the independent variable; $x \in [a, b]$.

2: **neq** – INTEGER

The number of differential equations.

3: **m**(**neq**) – INTEGER array

m(i) contains m_i, the order of the ith differential equation, for $i = 1, 2, \ldots,$ **neq**.

4: **user** – INTEGER array

guess is called from nag_ode_bvp_coll_nlin_solve (d02tl) with the object supplied to nag_ode_bvp_coll_nlin_solve (d02tl).

Output Parameters

1: **y**(**neq**, $0 : *$) – REAL (KIND=nag_wp) array

y(i, j) must contain $y_i^{(j)}(x)$, for $i = 1, 2, \ldots,$ **neq** and $j = 0, 1, \ldots,$ **m**$(i) - 1$.
Note: $y_i^{(0)}(x) = y_i(x)$.

2: **dym**(**neq**) – REAL (KIND=nag_wp) array

dym(i) must contain $y_i^{(m_i)}(x)$, for $i = 1, 2, \ldots,$ **neq**.

3: **user** – INTEGER array

8: **rcomm**$(*)$ – REAL (KIND=nag_wp) array

Note: the dimension 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_setup (d02tv).

This must be the same array as supplied to nag_ode_bvp_coll_nlin_setup (d02tv) and **must** remain unchanged between calls.

9: **icomm**$(*)$ – INTEGER array

Note: the dimension 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_setup (d02tv).

This must be the same array as supplied to nag_ode_bvp_coll_nlin_setup (d02tv) and **must** remain unchanged between calls.

5.2 Optional Input Parameters

1: **user** – INTEGER array

user is not used by nag_ode_bvp_coll_nlin_solve (d02tl), but is passed to **ffun**, **fjac**, **gafun**, **gbfun**, **gajac**, **gbjac** and **guess**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

5.3 Output Parameters

1: **rcomm**$(*)$ – REAL (KIND=nag_wp) array

Note: the dimension 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_setup (d02tv).

Contains information about the solution for use on subsequent calls to associated functions.

2: **icomm**$(*)$ – INTEGER array

Note: the dimension 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_setup (d02tv).

Contains information about the solution for use on subsequent calls to associated functions.

3: **user** – INTEGER array

4: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Note: nag_ode_bvp_coll_nlin_solve (d02tl) may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the function:

ifail $= 1$

Either the setup function has not been called or the communication arrays have become corrupted. No solution will be computed.

ifail $= 2$

Numerical singularity has been detected in the Jacobian used in the Newton iteration.
No results have been generated. Check the coding of the functions for calculating the Jacobians of system and boundary conditions.

ifail $= 3$

All Newton iterations that have been attempted have failed to converge.
No results have been generated. Check the coding of the functions for calculating the Jacobians of system and boundary conditions.
Try to provide a better initial solution approximation.

ifail $= 4$

A Newton iteration has failed to converge. The computation has not succeeded but results have been returned for an intermediate mesh on which convergence was achieved.
These results should be treated with extreme caution.

ifail $= 5$

The expected number of sub-intervals required to continue the computation exceeds the maximum specified.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

The accuracy of the solution is determined by the argument **tols** in the prior call to nag_ode_bvp_coll_nlin_setup (d02tv) (see Section 3 and Section 9 in nag_ode_bvp_coll_nlin_setup (d02tv) for details and advice). Note that error control is applied only to solution components (variables) and not to any derivatives of the solution. An estimate of the maximum error in the computed solution is available by calling nag_ode_bvp_coll_nlin_diag (d02tz).

8 Further Comments

If nag_ode_bvp_coll_nlin_solve (d02tl) returns with **ifail** $= 2$, 3, 4 or 5 and the call to nag_ode_bvp_coll_nlin_solve (d02tl) was a part of some continuation procedure for which successful calls to nag_ode_bvp_coll_nlin_solve (d02tl) have already been made, then it is possible that the adjustment(s) to the continuation parameter(s) between calls to nag_ode_bvp_coll_nlin_solve (d02tl) is (are) too large for the problem under consideration. More conservative adjustment(s) to the continuation parameter(s) might be appropriate.

9 Example

The following example is used to illustrate the treatment of a high-order system, control of the error in a derivative of a component of the original system, and the use of continuation. See also nag_ode_bvp_coll_nlin_setup (d02tv), nag_ode_bvp_coll_nlin_contin (d02tx), nag_ode_bvp_coll_nlin_interp (d02ty) and nag_ode_bvp_coll_nlin_diag (d02tz), for the illustration of other facilities.

Consider the steady flow of an incompressible viscous fluid between two infinite coaxial rotating discs. See Ascher *et al.* (1979) and the references therein. The governing equations are

$$\begin{aligned} \frac{1}{\sqrt{R}}f''' + ff''' + gg' &= 0 \\ \frac{1}{\sqrt{R}}g'' + fg' - f'g &= 0 \end{aligned}$$

subject to the boundary conditions

$$f(0) = f'(0) = 0, \quad g(0) = \Omega_0, \quad f(1) = f'(1) = 0, \quad g(1) = \Omega_1,$$

where R is the Reynolds number and Ω_0, Ω_1 are the angular velocities of the disks.

We consider the case of counter-rotation and a symmetric solution, that is $\Omega_0 = 1, \Omega_1 = -1$. This problem is more difficult to solve, the larger the value of R. For illustration, we use simple continuation to compute the solution for three different values of R ($= 10^6, 10^8, 10^{10}$). However, this problem can be addressed directly for the largest value of R considered here. Instead of the values suggested in Section 5 in nag_ode_bvp_coll_nlin_contin (d02tx) for **nmesh**, **ipmesh** and **mesh** in the call to nag_ode_bvp_coll_nlin_contin (d02tx) prior to a continuation call, we use every point of the final mesh for the solution of the first value of R, that is we must modify the contents of **ipmesh**. For illustrative purposes we wish to control the computed error in f' and so recast the equations as

$$\begin{aligned} y_1' &= y_2 \\ y_2''' &= -\sqrt{R}\left(y_1 y_2'' + y_3 y_3'\right) \\ y_3'' &= \sqrt{R}\left(y_2 y_3 - y_1 y_3'\right) \end{aligned}$$

subject to the boundary conditions

$$y_1(0) = y_2(0) = 0, \quad y_3(0) = \Omega, \quad y_1(1) = y_2(1) = 0, \quad y_3(1) = -\Omega, \quad \Omega = 1.$$

For the symmetric boundary conditions considered, there exists an odd solution about $x = 0.5$. Hence,

to satisfy the boundary conditions, we use the following initial approximations to the solution in **guess**:

$$\begin{aligned} y_1(x) &= -x^2\left(x-\tfrac{1}{2}\right)(x-1)^2 \\ y_2(x) &= -x(x-1)\left(5x^2-5x+1\right) \\ y_3(x) &= -8\Omega\left(x-\tfrac{1}{2}\right)^3. \end{aligned}$$

9.1 Program Text

```
     function d02tl_example

fprintf('d02tl example results\n\n');

global omega sqrofr

% Initialize variables and arrays.
neq  = nag_int(3);
nlbc = neq;
nrbc = neq;
ncol = nag_int(7);
mmax = nag_int(3);
m    = nag_int([1; 3; 2]);
tols = [1; 1; 1]/10^4;

% Set values for problem-specific physical parameters.
omega = 1;
r     = 10^6;

% Set up the mesh.
nmesh  = nag_int(11);
mxmesh = nag_int(51);
ipmesh = zeros(mxmesh, 1, nag_int_name);
mesh   = zeros(mxmesh, 1);

mesh(1:nmesh)     = [0:0.1:1];
ipmesh(1)         = 1;
ipmesh(2:nmesh-1) = 2;
ipmesh(nmesh)     = 1;

% d02tv is a setup routine to be called prior to d02tl.
[rcomm, icomm, ifail] = d02tv(...
                            m, nlbc, nrbc, ncol, tols, nmesh, mesh, ipmesh);

% Set number of continuation steps.
ncont = 3;

% We run through the calculation three times with different parameter sets.
for jcont = 1:ncont
  sqrofr = sqrt(r);
  fprintf('\n Tolerance = %8.1e  R = %10.3e\n\n', tols(1), r);

  % Call d02tk to solve BVP for this set of parameters.
  [rcomm, icomm, ifail] = ...
    d02tl(...
          @ffun, @fjac, @gafun, @gbfun, @gajac, @gbjac, @guess, rcomm, icomm);

  % Call d02tz to extract mesh from solution.
  [nmesh, mesh, ipmesh, ermx, iermx, ijermx, ifail] = ...
    d02tz( ...
           mxmesh, rcomm, icomm);

  % Output mesh results.
  fprintf(' Mesh size = %d\n', nmesh);
  fprintf(' Maximum error = %10.2e in interval %d for component %d\n\n', ...
          ermx, iermx, ijermx);
  fprintf('Mesh points:\n');
  for imesh = 1:nmesh
    fprintf( '%4d(%d) %6.4f', imesh, ipmesh(imesh), mesh(imesh));
    if mod(imesh, 4) == 0
      fprintf('\n');
```

```
    end
  end

  % Output solution, and store it for plotting.
  xarray = zeros(nmesh, 1);
  yarray = zeros(nmesh, 3);
  fprintf('\n\n    x        f         f''        g\n');
  for imesh = 1:nmesh
    % Call d02ty to perform interpolation on the solution.
    [y, rcomm, ifail] = d02ty(...
                              mesh(imesh), neq, mmax, rcomm, icomm);
    fprintf(' %6.3f  %8.4f  %8.4f  %8.4f\n', mesh(imesh), y(:,1));
    xarray(imesh) = mesh(imesh);
    yarray(imesh, 1:3) = y(1:3,1);
  end

  % Plot results for this parameter set.
  if jcont==1
    fig1 = figure;
  else
    if jcont==2
      fig2 = figure;
    else
      fig3 = figure;
    end
  end
  display_plot(xarray, yarray, r);

  % Select mesh for next calculation.
  if jcont < ncont
    r = 100*r;
    ipmesh(2:nmesh-1) = 2;

    % d02tx allows the current solution to be used for continuation
    [rcomm, icomm, ifail] = d02tx(...
                                  nmesh, mesh, ipmesh, rcomm, icomm);
  end
end

function [f, user] = ffun(x, y, neq, m, user)
  % Evaluate derivative functions (rhs of system of ODEs).

  global omega sqrofr; % For communication with main routine.
  f = zeros(neq, 1);
  f(1,1) =   y(2,1);
  f(2,1) = -(y(1,1)*y(2,3) + y(3,1)*y(3,2))*sqrofr;
  f(3,1) =  (y(2,1)*y(3,1) - y(1,1)*y(3,2))*sqrofr;

function [dfdy, user] = fjac(x, y, neq, m, dfdy, user)
  % Evaluate Jacobians (partial derivatives of f).

  global omega sqrofr
  dfdy = zeros(neq, neq, 3);
  dfdy(1,2,1) =  1.0;
  dfdy(2,1,1) = -y(2,3)*sqrofr;
  dfdy(2,2,3) = -y(1,1)*sqrofr;
  dfdy(2,3,1) = -y(3,2)*sqrofr;
  dfdy(2,3,2) = -y(3,1)*sqrofr;
  dfdy(3,1,1) = -y(3,2)*sqrofr;
  dfdy(3,2,1) =  y(3,1)*sqrofr;
  dfdy(3,3,1) =  y(2,1)*sqrofr;
  dfdy(3,3,2) = -y(1,1)*sqrofr;

function [ga, user] = gafun(ya, neq, m, nlbc, user)
  % Evaluate boundary conditions at left-hand end of range.

  global omega sqrofr
  ga = zeros(nlbc, 1);
  ga(1) = ya(1);
  ga(2) = ya(2);
  ga(3) = ya(3) - omega;
```

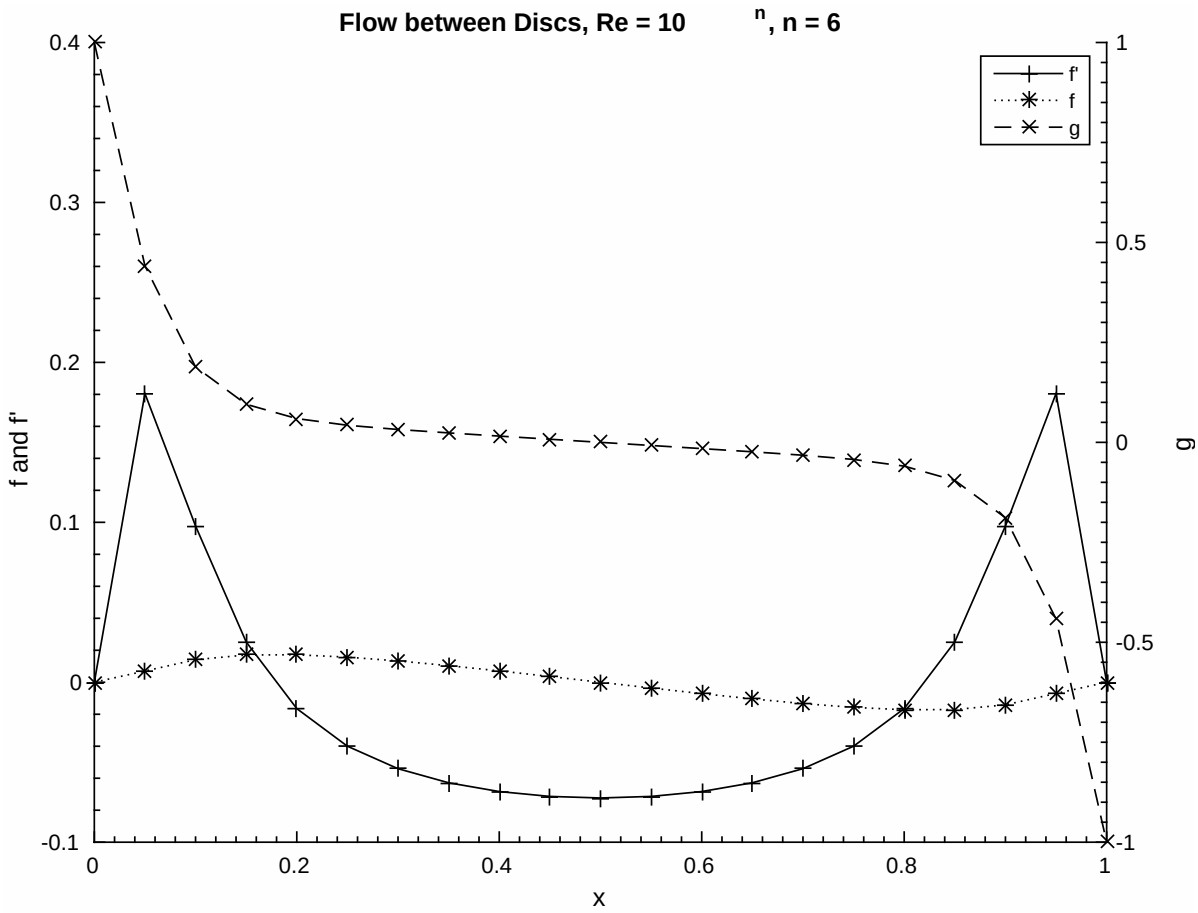
```
function [dgady, user] = gajac(ya, neq, m, nlbc, dgady, user)
  % Evaluate Jacobians (partial derivatives of ga).

  dgady = zeros(nlbc, neq, 3);
  dgady(1,1,1) = 1;
  dgady(2,2,1) = 1;
  dgady(3,3,1) = 1;

function [gb, user] = gbfun(yb, neq, m, nrbc, user)
  % Evaluate boundary conditions at right-hand end of range.

  global omega sqrofr
  gb = zeros(nrbc, 1);
  gb(1) = yb(1);
  gb(2) = yb(2);
  gb(3) = yb(3) + omega;

function [dgbdy, user] = gbjac(yb, neq, m, nrbc, dgbdy, user)
  % Evaluate Jacobians (partial derivatives of gb).

  dgbdy = zeros(nrbc, neq, 3);
  dgbdy(1,1,1) = 1;
  dgbdy(2,2,1) = 1;
  dgbdy(3,3,1) = 1;

function [y, dym, user] = guess(x, neq, m, user)
  % Evaluate initial approximations to solution components and derivatives.

  global omega sqrofr
  y = zeros(neq, 3);
  dym = zeros(neq, 1);
  y(1,1) = -x^2*(x - 0.5)*(x - 1)^2;
  y(2,1) = -x*(x - 1)*(5*x^2 - 5*x + 1);
  y(3,1) = -8*omega*(x - 0.5)^3;
  y(2,2) = -(20*x^3 - 30*x^2 + 12*x - 1);
  y(2,3) = -(60*x^2 - 60*x + 12*x);
  y(3,2) = -24*omega*(x - 0.5)^2;

  dym(1) = y(2,1);
  dym(2) = -(120*x - 60);
  dym(3) = -56*omega*(x - 0.5);

function display_plot(x, y, r)
  % Plot two of the curves, then add the other one.
  [haxes, hline1, hline2] = plotyy(x, y(:,2), x, y(:,3));
  % We want the third curve to be plotted on the left-hand y-axis.
  hold(haxes(1), 'on');
  hline3 = plot(x, y(:,1));
  % Set the axis limits and the tick specifications to beautify the plot.
  set(haxes(1), 'YLim', [-0.1 0.4]);
  set(haxes(2), 'YLim', [-1 1]);
  set(haxes(1), 'XMinorTick', 'on', 'YMinorTick', 'on');
  set(haxes(2), 'YMinorTick', 'on');
  set(haxes(1), 'YTick', [-0.1:0.1:0.4]);
  set(haxes(2), 'YTick', [-1:0.5:1]);
  for iaxis = 1:2
    % These properties must be the same for both sets of axes.
    set(haxes(iaxis), 'XLim', [0 1]);
    set(haxes(iaxis), 'XTick', [0:0.2:1]);
  end
  set(gca, 'box', 'off');
  % Add title.
  ord = log10(r);
  title(['Flow between Discs, Re = 10^n, n = ', num2str(ord)]);
  % Label the axes.
  xlabel('x');
  ylabel(haxes(1), 'f and f''');
  ylabel(haxes(2), 'g');
  % Add a legend.
  legend('f''','f','g','Location','Best')
```

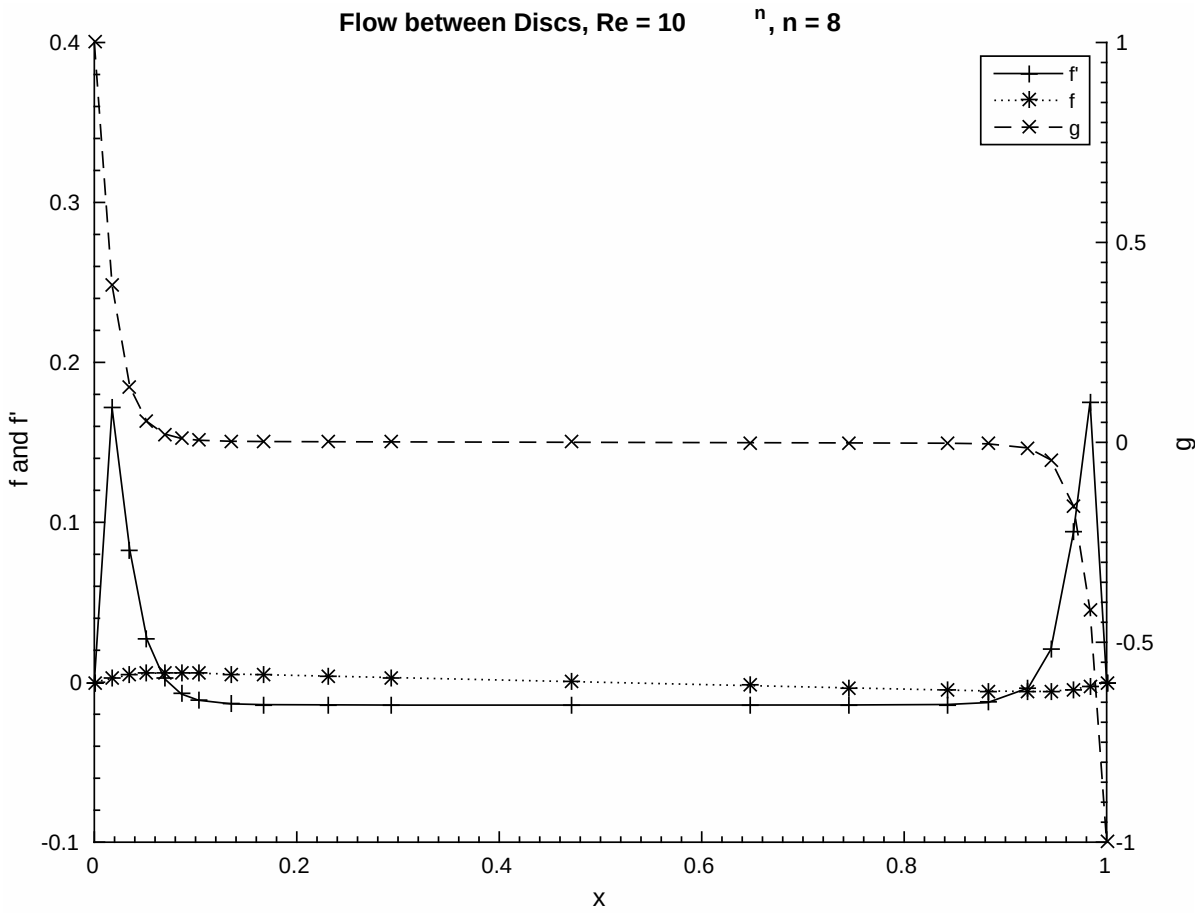
```
  % Set some features of the three lines.
  set(hline1, 'Linewidth', 0.25, 'Marker', '+', 'LineStyle', '-', ...
     'Color', 'Magenta');
  set(hline2, 'Linewidth', 0.25, 'Marker', 'x', 'LineStyle', '--');
  set(hline3, 'Linewidth', 0.25, 'Marker', '*', 'LineStyle', ':');
```

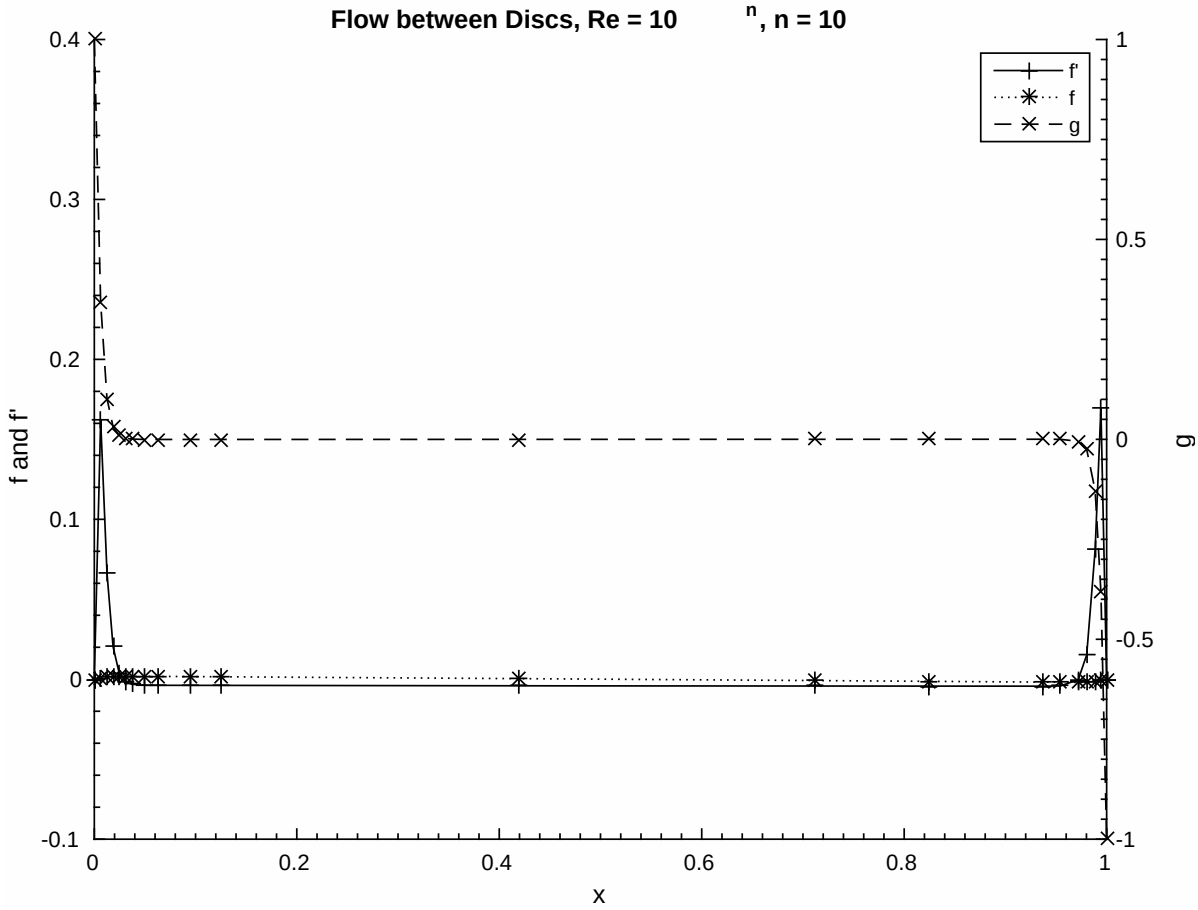
9.2 Program Results

```
     d02tl example results

 Tolerance =  1.0e-04  R =  1.000e+06

 Mesh size = 21
 Maximum error =   6.16e-10 in interval 20 for component 3

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

    x        f         f'        g
  0.000    0.0000    0.0000    1.0000
  0.050    0.0070    0.1805    0.4416
  0.100    0.0141    0.0977    0.1886
  0.150    0.0171    0.0252    0.0952
  0.200    0.0172   -0.0165    0.0595
  0.250    0.0157   -0.0400    0.0427
  0.300    0.0133   -0.0540    0.0322
  0.350    0.0104   -0.0628    0.0236
  0.400    0.0071   -0.0683    0.0156
  0.450    0.0036   -0.0714    0.0078
  0.500    0.0000   -0.0724    0.0000
  0.550   -0.0036   -0.0714   -0.0078
  0.600   -0.0071   -0.0683   -0.0156
  0.650   -0.0104   -0.0628   -0.0236
  0.700   -0.0133   -0.0540   -0.0322
  0.750   -0.0157   -0.0400   -0.0427
  0.800   -0.0172   -0.0165   -0.0595
  0.850   -0.0171    0.0252   -0.0952
  0.900   -0.0141    0.0977   -0.1886
  0.950   -0.0070    0.1805   -0.4416
  1.000   -0.0000   -0.0000   -1.0000

 Tolerance =  1.0e-04  R =  1.000e+08

 Mesh size = 21
 Maximum error =   4.49e-09 in interval 6 for component 3

Mesh points:
   1(1) 0.0000   2(3) 0.0176   3(2) 0.0351   4(3) 0.0520
   5(2) 0.0689   6(3) 0.0859   7(2) 0.1030   8(3) 0.1351
   9(2) 0.1672  10(3) 0.2306  11(2) 0.2939  12(3) 0.4713
  13(2) 0.6486  14(3) 0.7455  15(2) 0.8423  16(3) 0.8824
  17(2) 0.9225  18(3) 0.9449  19(2) 0.9673  20(3) 0.9836
  21(1) 1.0000

    x        f         f'        g
  0.000    0.0000    0.0000    1.0000
  0.018    0.0025    0.1713    0.3923
  0.035    0.0047    0.0824    0.1381
  0.052    0.0056    0.0267    0.0521
  0.069    0.0058    0.0025    0.0213
  0.086    0.0057   -0.0073    0.0097
  0.103    0.0056   -0.0113    0.0053
  0.135    0.0052   -0.0135    0.0027
  0.167    0.0047   -0.0140    0.0020
  0.231    0.0038   -0.0142    0.0015
```

```
  0.294    0.0029   -0.0142    0.0012
  0.471    0.0004   -0.0143    0.0002
  0.649   -0.0021   -0.0143   -0.0008
  0.745   -0.0035   -0.0142   -0.0014
  0.842   -0.0049   -0.0139   -0.0022
  0.882   -0.0054   -0.0127   -0.0036
  0.922   -0.0058   -0.0036   -0.0141
  0.945   -0.0057    0.0205   -0.0439
  0.967   -0.0045    0.0937   -0.1592
  0.984   -0.0023    0.1753   -0.4208
  1.000   -0.0000    0.0000   -1.0000

 Tolerance =  1.0e-04  R =  1.000e+10

 Mesh size = 21
 Maximum error =   3.13e-06 in interval 7 for component 3

Mesh points:
   1(1) 0.0000   2(3) 0.0063   3(2) 0.0125   4(3) 0.0185
   5(2) 0.0245   6(3) 0.0308   7(2) 0.0370   8(3) 0.0500
   9(2) 0.0629  10(3) 0.0942  11(2) 0.1256  12(3) 0.4190
  13(2) 0.7125  14(3) 0.8246  15(2) 0.9368  16(3) 0.9544
  17(2) 0.9719  18(3) 0.9803  19(2) 0.9886  20(3) 0.9943
  21(1) 1.0000

    x        f         f'        g
  0.000    0.0000    0.0000    1.0000
  0.006    0.0009    0.1623    0.3422
  0.013    0.0016    0.0665    0.1021
  0.019    0.0018    0.0204    0.0318
  0.025    0.0019    0.0041    0.0099
  0.031    0.0019   -0.0014    0.0028
  0.037    0.0019   -0.0031    0.0007
  0.050    0.0019   -0.0038   -0.0002
  0.063    0.0018   -0.0038   -0.0003
  0.094    0.0017   -0.0039   -0.0003
  0.126    0.0016   -0.0039   -0.0002
  0.419    0.0004   -0.0041   -0.0001
  0.712   -0.0008   -0.0042    0.0001
  0.825   -0.0013   -0.0043    0.0002
  0.937   -0.0018   -0.0043    0.0003
  0.954   -0.0019   -0.0042    0.0001
  0.972   -0.0019   -0.0003   -0.0049
  0.980   -0.0019    0.0152   -0.0252
  0.989   -0.0015    0.0809   -0.1279
  0.994   -0.0008    0.1699   -0.3814
  1.000    0.0000    0.0000   -1.0000
```


Flow between Discs, Re = 10 n, n = 6
f'
f
g
f and f'
g
x
0.4
0.3
0.2
0.1
0
-0.1
1
0.5
0
-0.5
-1
0
0.2
0.4
0.6
0.8
1


Flow between Discs, Re = 10 n, n = 8
f'
f
g
f and f'
g
x
0.4
0.3
0.2
0.1
0
-0.1
1
0.5
0
-0.5
-1
0
0.2
0.4
0.6
0.8
1


Flow between Discs, Re = 10 n, n = 10
f'
f
g
f and f'
g
x
0.4
0.3
0.2
0.1
0
-0.1
1
0.5
0
-0.5
-1
0
0.2
0.4
0.6
0.8
1