NAG Library Function Document

nag_ode_bvp_coll_nlin_solve (d02tlc)

1 Purpose

nag_ode_bvp_coll_nlin_solve (d02tlc) solves a general two-point boundary value problem for a nonlinear mixed order system of ordinary differential equations.

2 Specification

```
#include <nag.h>
#include <nagd02.h>
void nag_ode_bvp_coll_nlin_solve (
     void (*ffun)(double x, const double y[], Integer neq, const Integer m[],
          double f[], Nag_Comm *comm),
     void (*fjac)(double x, const double y[], Integer neq, const Integer m[],
          double dfdy[], Nag_Comm *comm),
     void (*gafun)(const double ya[], Integer neq, const Integer m[],
          Integer nlbc, double ga[], Nag_Comm *comm),
     void (*gbfun)(const double yb[], Integer neq, const Integer m[],
          Integer nrbc, double gb[], Nag_Comm *comm),
     void (*gajac)(const double ya[], Integer neq, const Integer m[],
          Integer nlbc, double dgady[], Nag_Comm *comm),
     void (*gbjac)(const double yb[], Integer neq, const Integer m[],
          Integer nrbc, double dgbdy[], Nag_Comm *comm),
     void (*guess)(double x, Integer neq, const Integer m[], double y[],
          double dym[], Nag_Comm *comm),
     double rcomm[], Integer icomm[], Nag_Comm *comm, NagError *fail)
```

3 Description

nag_ode_bvp_coll_nlin_solve (d02tlc) and its associated functions (nag_ode_bvp_coll_nlin_setup (d02tvc), 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 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 (d02tvc) 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 (d02tvc).) 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 and other details of the solution procedure, and nag_ode_bvp_coll_nlin_interp (d02tyc) 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 (d02tlc) is given in Section 3 in nag_ode_bvp_coll_nlin_setup (d02tvc).

nag_ode_bvp_coll_nlin_solve (d02tlc) 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 (d02txc) should be used in between calls to nag_ode_bvp_coll_nlin_solve (d02tlc) in this context.

See Section 9 in nag_ode_bvp_coll_nlin_setup (d02tvc) 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 Arguments

1: **ffun** – function, supplied by the user *External Function*

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

The specification of **ffun** is:

```
void ffun (double x, const double y[], Integer neq, const Integer m[],
     double f[], Nag_Comm *comm)
```

1: **x** – double *Input*

On entry: x, the independent variable.

2: **y**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **y** is **neq** $\times$ max(**m**[i]).

Where $\mathbf{Y}(i, j)$ appears in this document, it refers to the array element **y**[$j \times$ **neq** $+ i - 1$].

On entry: $\mathbf{Y}(i,j)$ contains $y_i^{(j)}(x)$, for $i = 1,2,\ldots,\mathbf{neq}$ and $j = 0,1,\ldots,\mathbf{m}[i-1]-1$.
Note: $y_i^{(0)}(x) = y_i(x)$.

3: **neq** – Integer *Input*

On entry: the number of differential equations.

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

On entry: $\mathbf{m}[i-1]$ contains m_i, the order of the ith differential equation, for $i = 1,2,\ldots,\mathbf{neq}$.

5: **f**[**neq**] – double *Output*

On exit: $\mathbf{f}[i-1]$ must contain f_i, for $i = 1,2,\ldots,\mathbf{neq}$.

6: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **ffun**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_ode_bvp_coll_nlin_solve (d02tlc) you may allocate memory and initialize these pointers with various quantities for use by **ffun** when called from nag_ode_bvp_coll_nlin_solve (d02tlc) (see Section 3.2.1.1 in the Essential Introduction).

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

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

The specification of **fjac** is:

```
void fjac (double x, const double y[], Integer neq, const Integer m[],
     double dfdy[], Nag_Comm *comm)
```

1: **x** – double *Input*

On entry: x, the independent variable.

2: **y**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **y** is **neq** $\times$ max(**m**[i]).

Where $\mathbf{Y}(i,j)$ appears in this document, it refers to the array element $\mathbf{y}[j \times \mathbf{neq} + i - 1]$.

On entry: $\mathbf{Y}(i,j)$ contains $y_i^{(j)}(x)$, for $i = 1,2,\ldots,\mathbf{neq}$ and $j = 0,1,\ldots,\mathbf{m}[i-1]-1$.
Note: $y_i^{(0)}(x) = y_i(x)$.

3: **neq** – Integer *Input*

On entry: the number of differential equations.

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

On entry: $\mathbf{m}[i-1]$ contains m_i, the order of the ith differential equation, for $i = 1,2,\ldots,\mathbf{neq}$.

5: **dfdy**[dim] – double *Input/Output*

Note: the dimension, *dim*, of the array **dfdy** is **neq** × **neq** × max(**m**[i]).

Where **DFDY**(i, j, k) appears in this document, it refers to the array element **dfdy**[$k \times$ **neq** × **neq** + $(j-1) \times$ **neq** + $i - 1$].

On entry: set to zero.

On exit: **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**[$j-1$] $- 1$. Only nonzero partial derivatives need be set.

6: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **fjac**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_ode_bvp_coll_nlin_solve (d02tlc) you may allocate memory and initialize these pointers with various quantities for use by **fjac** when called from nag_ode_bvp_coll_nlin_solve (d02tlc) (see Section 3.2.1.1 in the Essential Introduction).

3: **gafun** – function, supplied by the user *External Function*

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 $z(y(a))$.

The specification of **gafun** is:

```
void gafun (const double ya[], Integer neq, const Integer m[],
     Integer nlbc, double ga[], Nag_Comm *comm)
```

1: **ya**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **ya** is **neq** × max(**m**[i]).

Where **YA**(i, j) appears in this document, it refers to the array element **ya**[$j \times$ **neq** + $i - 1$].

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

2: **neq** – Integer *Input*

On entry: the number of differential equations.

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

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

4: **nlbc** – Integer *Input*

On entry: the number of boundary conditions at a.

5: **ga**[**nlbc**] – double *Output*

On exit: **ga**[$i-1$] must contain $g_i(z(y(a)))$, for $i = 1, 2, \ldots,$ **nlbc**.

6: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **gafun**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_ode_bvp_coll_nlin_solve (d02tlc) you may allocate memory and initialize these pointers with various quantities for use by **gafun** when called from nag_ode_bvp_coll_nlin_solve (d02tlc) (see Section 3.2.1.1 in the Essential Introduction).

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

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 $z(y(b))$.

The specification of **gbfun** is:

```
void gbfun (const double yb[], Integer neq, const Integer m[],
     Integer nrbc, double gb[], Nag_Comm *comm)
```

1: **yb**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **yb** is **neq** $\times$ max(**m**[i]).

Where **YB**(i,j) appears in this document, it refers to the array element **yb**[$j \times$ **neq** $+ i - 1$].

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

2: **neq** – Integer *Input*

On entry: the number of differential equations.

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

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

4: **nrbc** – Integer *Input*

On entry: the number of boundary conditions at b.

5: **gb**[**nrbc**] – double *Output*

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

6: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **gbfun**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_ode_bvp_coll_nlin_solve (d02tlc) you may allocate memory and initialize these pointers with various quantities for use by **gbfun** when called from nag_ode_bvp_coll_nlin_solve (d02tlc) (see Section 3.2.1.1 in the Essential Introduction).

5: **gajac** – function, supplied by the user *External Function*

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

The specification of **gajac** is:

```
void gajac (const double ya[], Integer neq, const Integer m[],
     Integer nlbc, double dgady[], Nag_Comm *comm)
```

1: **ya**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **ya** is **neq** $\times$ max(**m**[i]).

Where **YA**(i,j) appears in this document, it refers to the array element **ya**[$j \times$ **neq** $+ i - 1$].

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

2: **neq** – Integer *Input*

On entry: the number of differential equations.

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

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

4: **nlbc** – Integer *Input*

On entry: the number of boundary conditions at a.

5: **dgady**[dim] – double *Input/Output*

Note: the dimension, *dim*, of the array **dgady** is **nlbc** $\times$ **neq** $\times$ max(**m**[i]).

Where **DGADY**(i,j,k) appears in this document, it refers to the array element **dgady**[$k \times$ **nlbc** $\times$ **neq** $+ (j-1) \times$ **nlbc** $+ i - 1$].

On entry: set to zero.

On exit: **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**[$j-1$] $- 1$. Only nonzero partial derivatives need be set.

6: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **gajac**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_ode_bvp_coll_nlin_solve (d02tlc) you may allocate memory and initialize these pointers with various quantities for use by **gajac** when called from nag_ode_bvp_coll_nlin_solve (d02tlc) (see Section 3.2.1.1 in the Essential Introduction).

6: **gbjac** – function, supplied by the user *External Function*

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

The specification of **gbjac** is:

```
void gbjac (const double yb[], Integer neq, const Integer m[],
     Integer nrbc, double dgbdy[], Nag_Comm *comm)
```

1: **yb**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **yb** is **neq** $\times$ max(**m**[i]).

Where **YB**(i,j) appears in this document, it refers to the array element **yb**[$j \times$ **neq** $+ i - 1$].

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

2: **neq** – Integer *Input*

On entry: the number of differential equations.

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

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

4: **nrbc** – Integer *Input*

On entry: the number of boundary conditions at b.

5: **dgbdy**[dim] – double *Input/Output*

Note: the dimension, *dim*, of the array **dgbdy** is **nrbc** $\times$ **neq** $\times$ max(**m**[i]).

Where **DGBDY**(i,j,k) appears in this document, it refers to the array element **dgbdy**[$(k-1) \times$ **nrbc** $\times$ **neq** $+ (j-1) \times$ **nrbc** $+ i - 1$].

On entry: set to zero.

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

6: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **gbjac**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_ode_bvp_coll_nlin_solve (d02tlc) you may allocate memory and initialize these pointers with various quantities for use by **gbjac** when called from nag_ode_bvp_coll_nlin_solve (d02tlc) (see Section 3.2.1.1 in the Essential Introduction).

7: **guess** – function, supplied by the user *External Function*

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

If nag_ode_bvp_coll_nlin_solve (d02tlc) is being used in conjunction with nag_ode_bvp_coll_nlin_contin (d02txc) as part of a continuation process, then **guess** is not

called by nag_ode_bvp_coll_nlin_solve (d02tlc) after the call to nag_ode_bvp_coll_nlin_contin (d02txc).

The specification of **guess** is:

```
void guess (double x, Integer neq, const Integer m[], double y[],
     double dym[], Nag_Comm *comm)
```

1: **x** – double *Input*

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

2: **neq** – Integer *Input*

On entry: the number of differential equations.

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

On entry: $\mathbf{m}[i-1]$ contains m_i, the order of the ith differential equation, for $i = 1, 2, \ldots,$ **neq**.

4: **y**[dim] – double *Output*

Note: the dimension, *dim*, of the array **y** is **neq** $\times$ max(**m**[i]).

Where $\mathbf{Y}(i, j)$ appears in this document, it refers to the array element $\mathbf{y}[j \times \mathbf{neq} + i - 1]$.

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

5: **dym**[**neq**] – double *Output*

On exit: $\mathbf{dym}[i-1]$ must contain $y_i^{(m_i)}(x)$, for $i = 1, 2, \ldots,$ **neq**.

6: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **guess**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_ode_bvp_coll_nlin_solve (d02tlc) you may allocate memory and initialize these pointers with various quantities for use by **guess** when called from nag_ode_bvp_coll_nlin_solve (d02tlc) (see Section 3.2.1.1 in the Essential Introduction).

8: **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_setup (d02tvc).

On entry: this must be the same array as supplied to nag_ode_bvp_coll_nlin_setup (d02tvc) and **must** remain unchanged between calls.

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

9: **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_setup (d02tvc).

On entry: this must be the same array as supplied to nag_ode_bvp_coll_nlin_setup (d02tvc) and **must** remain unchanged between calls.

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

10: **comm** – Nag_Comm *

The NAG communication argument (see Section 3.2.1.1 in the Essential Introduction).

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

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

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

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

NE_CONVERGENCE_SOL

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.

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 the Essential Introduction for further information.

NE_MISSING_CALL

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

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NE_SING_JAC

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.

NW_MAX_SUBINT

The expected number of sub-intervals required to continue the computation exceeds the maximum specified: ⟨*value*⟩.
Results have been generated which may be useful.
Try increasing this number or relaxing the error requirements.

NW_NOT_CONVERGED

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.

7 Accuracy

The accuracy of the solution is determined by the argument **tols** in the prior call to nag_ode_bvp_coll_nlin_setup (d02tvc) (see Sections 3 and 9 in nag_ode_bvp_coll_nlin_setup (d02tvc) 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 (d02tzc).

8 Parallelism and Performance

nag_ode_bvp_coll_nlin_solve (d02tlc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_ode_bvp_coll_nlin_solve (d02tlc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

If nag_ode_bvp_coll_nlin_solve (d02tlc) returns with **fail.code** = NE_CONVERGENCE_SOL, NE_SING_JAC, NW_MAX_SUBINT or NW_NOT_CONVERGED and the call to nag_ode_bvp_coll_nlin_solve (d02tlc) was a part of some continuation procedure for which successful calls to nag_ode_bvp_coll_nlin_solve (d02tlc) 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 (d02tlc) is (are) too large for the problem under consideration. More conservative adjustment(s) to the continuation parameter(s) might be appropriate.

10 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 (d02tvc), 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.

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 (d02txc) for **nmesh**, **ipmesh** and **mesh** in the call to nag_ode_bvp_coll_nlin_contin (d02txc) 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**:

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

10.1 Program Text

```
/* nag_ode_bvp_coll_nlin_solve (d02tlc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */

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

typedef struct {
  double  omega;
  double  sqrofr;
} 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 */
  double      one = 1.0;
  Integer     exit_status = 0, neq = 3, mmax = 3, nlbc = 3, nrbc = 3;
  double      dx, ermx, r, omega;
  Integer     i, iermx, ijermx, j, k, licomm, lrcomm, mxmesh, ncol, ncont,
```

```
            nmesh;
  /* Arrays */
  static double ruser[7] = {-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0};
  double      rdum[1];
  Integer     idum[2];
  double      *mesh = 0, *tol = 0, *rcomm = 0, *y = 0;
  Integer     *ipmesh = 0, *icomm = 0, *m = 0;
  func_data   fd;
  /* Nag Types */
  Nag_Comm    comm;
  NagError    fail;

  INIT_FAIL(fail);

  printf("nag_ode_bvp_coll_nlin_solve (d02tlc) 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 "%"NAG_IFMT "%*[^\n] ", &ncol, &nmesh, &mxmesh);
#else
  scanf("%"NAG_IFMT "%"NAG_IFMT "%"NAG_IFMT "%*[^\n] ", &ncol, &nmesh, &mxmesh);
#endif

  if (!(mesh   = NAG_ALLOC(mxmesh, double)) ||
      !(m      = NAG_ALLOC(neq, Integer)) ||
      !(tol    = 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 problem equation orders. */
  m[0] = 1;
  m[1] = 3;
  m[2] = 2;

#ifdef _WIN32
  scanf_s("%lf%*[^\n] ", &omega);
#else
  scanf("%lf%*[^\n] ", &omega);
#endif

#ifdef _WIN32
  for (i = 0; i < neq; i++) scanf_s("%lf", &tol[i]);
#else
  for (i = 0; i < neq; i++) scanf("%lf", &tol[i]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  dx = one/(double) (nmesh - 1);
  mesh[0] = 0.0;
  ipmesh[0] = 1;
  for (i = 1; i < nmesh - 1; i++) {
    mesh[i] = mesh[i - 1] + dx;
    ipmesh[i] = 2;
```

```
  }
  mesh[nmesh-1] = one;
  ipmesh[nmesh-1] = 1;

  /* Query to get size of rcomm and icomm (by setting lrcomm=0) */
  /* 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, rdum, 0, idum, 2, &fail);
  if (fail.code == NE_NOERROR) {
    lrcomm = idum[0];
    licomm = idum[1];
    printf("lrcomm = %"NAG_IFMT", licomm = %"NAG_IFMT"\n",lrcomm,licomm);
    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, 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;
  }

  /* Number of continuation steps (last r=100**ncont, sqrofr=10**ncont) */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT "%*[^\n] ", &ncont);
#else
  scanf("%"NAG_IFMT "%*[^\n] ", &ncont);
#endif
  /* Initialize problem continuation parameter. */
#ifdef _WIN32
  scanf_s("%lf%*[^\n] ", &r);
#else
  scanf("%lf%*[^\n] ", &r);
#endif

  /* Set data required for the user-supplied functions */
  fd.omega = omega;
  fd.sqrofr = sqrt(r);
  /* Associate the data structure with comm.p */
  comm.p = (Pointer) &fd;

  for (j = 0; j < ncont; j++) {
    printf("\n Tolerance = %8.1e  r = %10.3e\n", tol[0], r);
    /* Solve problem.*/

    /* 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;
    }

    /* Print mesh and error statistics.*/
    printf("\n Used a mesh of %4"NAG_IFMT "  points\n", nmesh);
    printf(" Maximum error = %10.2e in interval %4"NAG_IFMT" for component"
           " %4"NAG_IFMT"\n\n\n", ermx, iermx, ijermx);
    printf(" Mesh points:\n");
    for (i = 0; i < nmesh; i++) {
      printf("%4"NAG_IFMT"(%1"NAG_IFMT")", i+1, ipmesh[i]);
      printf("%12.4e%s", mesh[i], i%4==3?"\n":" ");
    }
    /* Print solution components on mesh.*/
    printf("\n\n      x        f        f'       g\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 = 4;
        goto END;
      }

      printf("%8.3f ", mesh[i]);
      for (k = 0; k < neq; k++) printf("%9.4f", y[k]);
      printf("\n");
    }
    if (j == ncont-1) {
      goto END;
    }
    /* Modify continuation parameter.*/
    r = 100.0 * r;
    fd.sqrofr = sqrt(r);

    /* Select mesh for continuation and call continuation primer routine.*/
    for (i = 1; i < nmesh - 1; i++) {
      ipmesh[i] = 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 = 5;
      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 y22 = y[1 + 2*neq];
  double y31 = y[2 + 1*neq];
  if (comm->user[0] == -1.0)
    {
      printf("(User-supplied callback ffun, first invocation.)\n");
      comm->user[0] = 0.0;
    }
  f[0] = y[1];
  f[1] = -(y[0]*y22  + y[2]*y31) * fd->sqrofr;
  f[2] =  (y[1]*y[2] - y[0]*y31) * fd->sqrofr;
}

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 y22 = y[1 + 2*neq];
  double y31 = y[2 + 1*neq];

  if (comm->user[1] == -1.0)
    {
      printf("(User-supplied callback fjac, first invocation.)\n");
      comm->user[1] = 0.0;
    }
  dfdy[0 + 1*neq + 0*neq*neq] = 1.0;
  dfdy[1 + 0*neq + 0*neq*neq] = -y22 * fd->sqrofr;
  dfdy[1 + 1*neq + 2*neq*neq] = -y[0]* fd->sqrofr;
  dfdy[1 + 2*neq + 0*neq*neq] = -y31 * fd->sqrofr;
  dfdy[1 + 2*neq + 1*neq*neq] = -y[2]* fd->sqrofr;
  dfdy[2 + 0*neq + 0*neq*neq] = -y31 * fd->sqrofr;
  dfdy[2 + 1*neq + 0*neq*neq] = -y[2]* fd->sqrofr;
  dfdy[2 + 2*neq + 0*neq*neq] =  y[1]* fd->sqrofr;
  dfdy[2 + 2*neq + 1*neq*neq] = -y[0]* fd->sqrofr;
}

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

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

```
  gb[1] = yb[1];
  gb[2] = yb[2] + fd->omega;
}

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

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

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 xh = x-0.5;
  double xx1 = x*(x-1.0);

  if (comm->user[6] == -1.0)
    {
      printf("(User-supplied callback guess, first invocation.)\n");
      comm->user[6] = 0.0;
    }
  y[0+0*neq] = -xh * pow(xx1, 2);
  y[1+0*neq] = -xx1 * (5.0*xx1 + 1.0);
  y[1+1*neq] = -2.0*xh * (10.0*xx1 + 1.0);
  y[1+2*neq] = -12.0 * (5.0*xx1 +  x);
  y[2+0*neq] = -8.0 * fd->omega * pow(xh, 3);
  y[2+1*neq] = -24.0 * fd->omega * pow(xh, 2);

  dym[0] = y[1];
  dym[1] = -120.0 * xh;
  dym[2] = -56.0 * fd->omega * xh;
}
```

10.2 Program Data

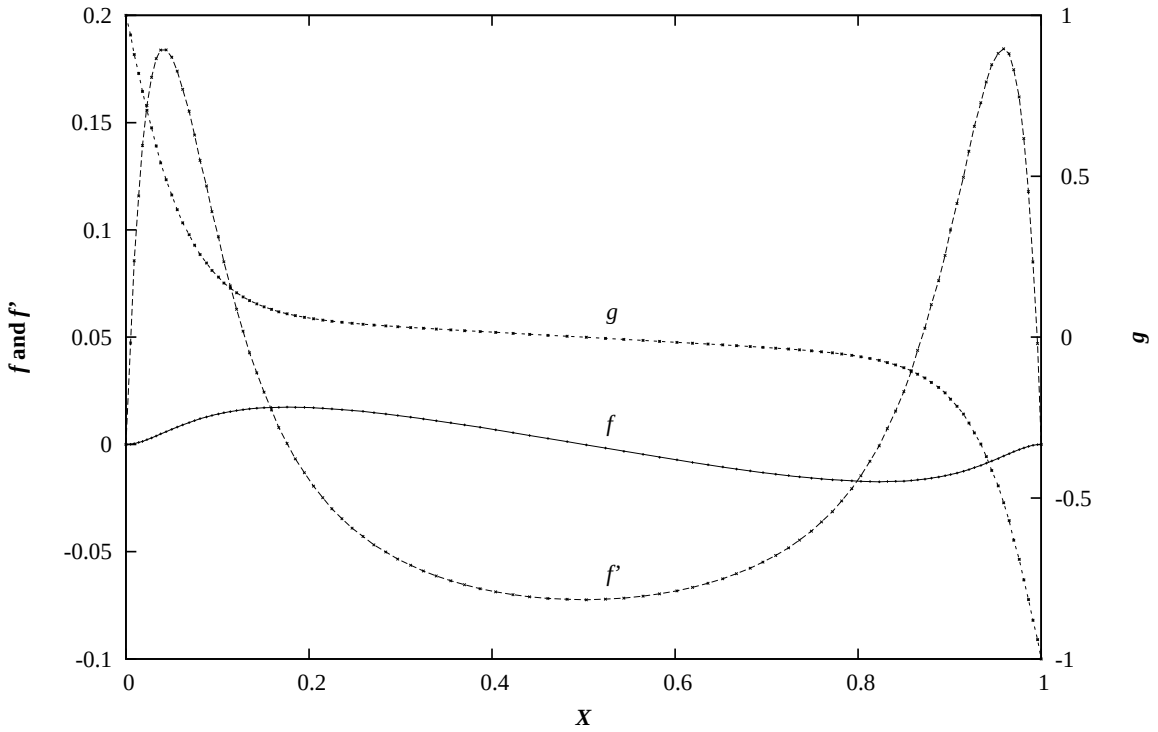
```
nag_ode_bvp_coll_nlin_solve (d02tlc) Example Program Data
   7      11      101        : ncol, nmesh, mxmesh
   1.0                       : omega
   1.0e-3  1.0e-3  1.0e-3    : tol(1:neq)
   2                         : ncont
   1.0e+6                    : r
```

10.3 Program Results

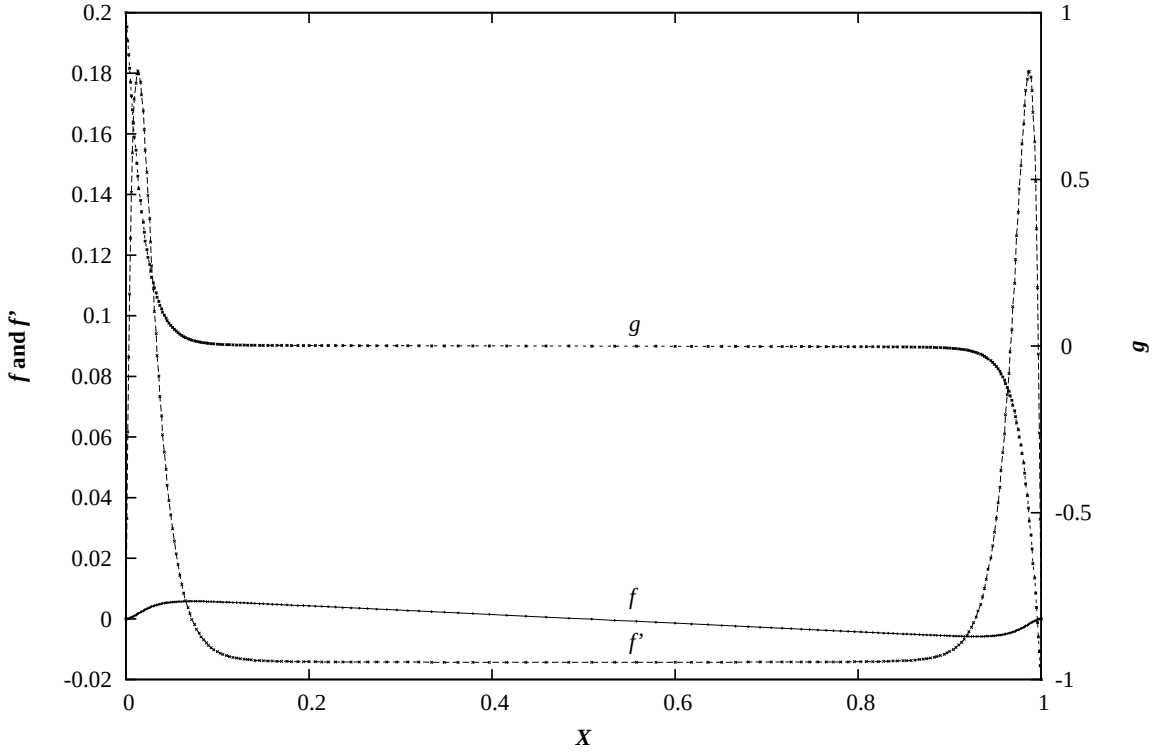
```
nag_ode_bvp_coll_nlin_solve (d02tlc) Example Program Results

lrcomm = 3012, licomm = 127

 Tolerance =  1.0e-03  r =  1.000e+06
(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 =   2.77e-07 in interval   17 for component    3


 Mesh points:
  1(1)  0.0000e+00    2(3)  5.0000e-02    3(2)  1.0000e-01    4(3)  1.5000e-01
  5(2)  2.0000e-01    6(3)  2.5000e-01    7(2)  3.0000e-01    8(3)  3.5000e-01
  9(2)  4.0000e-01   10(3)  4.5000e-01   11(2)  5.0000e-01   12(3)  5.5000e-01
 13(2)  6.0000e-01   14(3)  6.5000e-01   15(2)  7.0000e-01   16(3)  7.5000e-01
 17(2)  8.0000e-01   18(3)  8.5000e-01   19(2)  9.0000e-01   20(3)  9.5000e-01
 21(1)  1.0000e+00

     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-03  r =  1.000e+08

 Used a mesh of   41  points
 Maximum error =   3.18e-06 in interval   17 for component    3


 Mesh points:
  1(1)  0.0000e+00    2(3)  7.2063e-03    3(2)  1.4413e-02    4(3)  2.1550e-02
  5(2)  2.8687e-02    6(3)  3.5634e-02    7(2)  4.2581e-02    8(3)  5.0435e-02
  9(2)  5.8290e-02   10(3)  6.7353e-02   11(2)  7.6416e-02   12(3)  8.8097e-02
 13(2)  9.9779e-02   14(3)  1.1602e-01   15(2)  1.3227e-01   16(3)  1.5346e-01
 17(2)  1.7465e-01   18(3)  2.0571e-01   19(2)  2.3677e-01   20(3)  2.8658e-01
 21(2)  3.3638e-01   22(3)  4.4256e-01   23(2)  5.4874e-01   24(3)  6.3511e-01
 25(2)  7.2147e-01   26(3)  7.6429e-01   27(2)  8.0711e-01   28(3)  8.3483e-01
 29(2)  8.6254e-01   30(3)  8.8245e-01   31(2)  9.0236e-01   32(3)  9.1689e-01
 33(2)  9.3142e-01   34(3)  9.4253e-01   35(2)  9.5364e-01   36(3)  9.6254e-01
 37(2)  9.7143e-01   38(3)  9.7877e-01   39(2)  9.8611e-01   40(3)  9.9305e-01
 41(1)  1.0000e+00

     x        f        f'       g
  0.000    0.0000   0.0000   1.0000
```

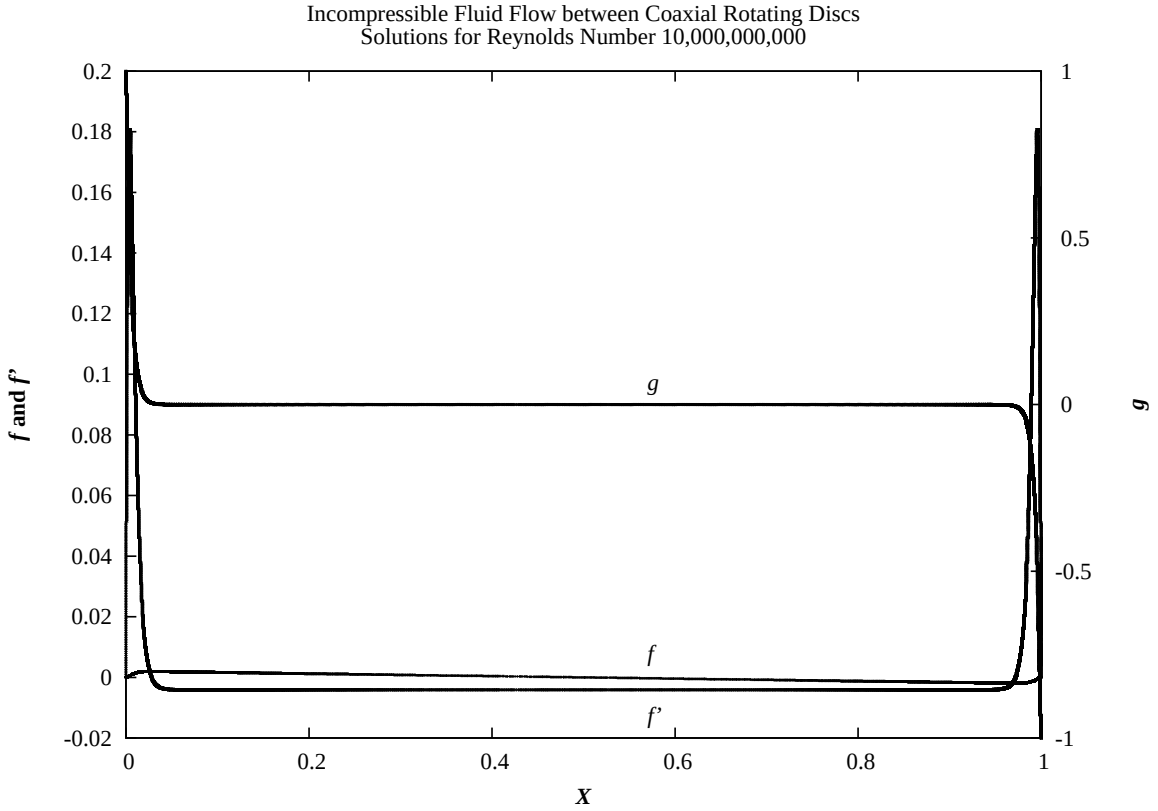
```
 0.007    0.0007   0.1560   0.7032
 0.014    0.0019   0.1815   0.4711
 0.022    0.0032   0.1561   0.3107
 0.029    0.0041   0.1185   0.2044
 0.036    0.0048   0.0847   0.1374
 0.043    0.0053   0.0580   0.0943
 0.050    0.0057   0.0361   0.0641
 0.058    0.0059   0.0214   0.0459
 0.067    0.0061   0.0104   0.0338
 0.076    0.0061   0.0036   0.0271
 0.088    0.0061  -0.0015   0.0225
 0.100    0.0061  -0.0045   0.0203
 0.116    0.0060  -0.0068   0.0188
 0.132    0.0059  -0.0082   0.0181
 0.153    0.0057  -0.0096   0.0175
 0.175    0.0055  -0.0108   0.0169
 0.206    0.0051  -0.0123   0.0159
 0.237    0.0047  -0.0137   0.0148
 0.287    0.0040  -0.0158   0.0126
 0.336    0.0031  -0.0175   0.0101
 0.443    0.0011  -0.0197   0.0037
 0.549   -0.0010  -0.0198  -0.0032
 0.635   -0.0026  -0.0183  -0.0085
 0.721   -0.0041  -0.0155  -0.0130
 0.764   -0.0047  -0.0137  -0.0148
 0.807   -0.0053  -0.0117  -0.0163
 0.835   -0.0056  -0.0103  -0.0171
 0.863   -0.0058  -0.0086  -0.0179
 0.882   -0.0060  -0.0070  -0.0187
 0.902   -0.0061  -0.0040  -0.0206
 0.917   -0.0061   0.0003  -0.0241
 0.931   -0.0061   0.0093  -0.0326
 0.943   -0.0059   0.0226  -0.0474
 0.954   -0.0055   0.0465  -0.0779
 0.963   -0.0050   0.0769  -0.1242
 0.971   -0.0041   0.1191  -0.2058
 0.979   -0.0031   0.1577  -0.3165
 0.986   -0.0018   0.1821  -0.4854
 0.993   -0.0006   0.1533  -0.7130
 1.000   -0.0000   0.0000  -1.0000
```

Example Program
Incompressible Fluid Flow between Coaxial Rotating Discs
Solutions for Reynolds Number 1,000,000



Incompressible Fluid Flow between Coaxial Rotating Discs
Solutions for Reynolds Number 100,000,000




Incompressible Fluid Flow between Coaxial Rotating Discs
Solutions for Reynolds Number 10,000,000,000
f and f'
g
X
g
f
f'
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.02
1
0.5
0
-0.5
-1
0
0.2
0.4
0.6
0.8
1