NAG Library Routine Document

D02TLF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

D02TLF solves a general two-point boundary value problem for a nonlinear mixed order system of ordinary differential equations.

2 Specification

```
SUBROUTINE D02TLF (FFUN, FJAC, GAFUN, GBFUN, GAJAC, GBJAC, GUESS, RCOMM,    &
                   ICOMM, IUSER, RUSER, IFAIL)

INTEGER            ICOMM(*), IUSER(*), IFAIL
REAL (KIND=nag_wp) RCOMM(*), RUSER(*)
EXTERNAL           FFUN, FJAC, GAFUN, GBFUN, GAJAC, GBJAC, GUESS
```

3 Description

D02TLF and its associated routines (D02TVF, D02TXF, D02TYF and D02TZF) solve the two-point boundary value problem for a nonlinear mixed order system of ordinary differential equations

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

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

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

First, D02TVF 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 D02TVF.) Then, D02TLF can be used to solve the boundary value problem. After successful computation, D02TZF can be used to ascertain details about the final mesh and other details of the solution procedure, and D02TYF can be used to compute the approximate solution anywhere on the interval $[a, b]$.

A description of the numerical technique used in D02TLF is given in Section 3 in D02TVF.

D02TLF 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. D02TXF should be used in between calls to D02TLF in this context.

See Section 9 in D02TVF for details of how to solve boundary value problems of a more general nature.

The routines 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

1: FFUN – SUBROUTINE, supplied by the user. *External Procedure*

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

The specification of FFUN is:

```
SUBROUTINE FFUN (X, Y, NEQ, M, F, IUSER, RUSER)
INTEGER             NEQ, M(NEQ), IUSER(*)
REAL (KIND=nag_wp) X, Y(NEQ,0:*), F(NEQ), RUSER(*)
```

1: X – REAL (KIND=nag_wp) *Input*

On entry: x, the independent variable.

2: Y(NEQ, $0:*$) – REAL (KIND=nag_wp) array *Input*

On entry: 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 *Input*

On entry: the number of differential equations.

4: M(NEQ) – INTEGER array *Input*

On entry: M(i) contains m_i, the order of the ith differential equation, for $i = 1,2,\ldots,$NEQ.

5: F(NEQ) – REAL (KIND=nag_wp) array *Output*

On exit: F(i) must contain f_i, for $i = 1,2,\ldots,$NEQ.

6: IUSER($*$) – INTEGER array *User Workspace*
7: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

FFUN is called with the parameters IUSER and RUSER as supplied to D02TLF. You are free to use the arrays IUSER and RUSER to supply information to FFUN as an alternative to using COMMON global variables.

FFUN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02TLF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: FJAC – SUBROUTINE, supplied by the user. *External Procedure*

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

The specification of FJAC is:

```
SUBROUTINE FJAC (X, Y, NEQ, M, DFDY, IUSER, RUSER)
INTEGER              NEQ, M(NEQ), IUSER(*)
REAL (KIND=nag_wp) X, Y(NEQ,0:*), DFDY(NEQ,NEQ,0:*), RUSER(*)
```

1: X – REAL (KIND=nag_wp) *Input*

On entry: x, the independent variable.

2: Y(NEQ, $0:*$) – REAL (KIND=nag_wp) array *Input*

On entry: Y(i,j) contains $y_i^{(j)}(x)$, for $i = 1,2,\dots,$NEQ and $j = 0,1,\dots,$M$(i) - 1$.
Note: $y_i^{(0)}(x) = y_i(x)$.

3: NEQ – INTEGER *Input*

On entry: the number of differential equations.

4: M(NEQ) – INTEGER array *Input*

On entry: M(i) contains m_i, the order of the ith differential equation, for $i = 1,2,\dots,$NEQ.

5: DFDY(NEQ, NEQ, $0:*$) – REAL (KIND=nag_wp) array *Input/Output*

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 $i = 1,2,\dots,$NEQ, $j = 1,2,\dots,$NEQ and $k = 0,1,\dots,$M$(j) - 1$. Only nonzero partial derivatives need be set.

6: IUSER($*$) – INTEGER array *User Workspace*
7: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

FJAC is called with the parameters IUSER and RUSER as supplied to D02TLF. You are free to use the arrays IUSER and RUSER to supply information to FJAC as an alternative to using COMMON global variables.

FJAC must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02TLF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

3: GAFUN – SUBROUTINE, supplied by the user. *External Procedure*

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:

```
SUBROUTINE GAFUN (YA, NEQ, M, NLBC, GA, IUSER, RUSER)
```

```
INTEGER              NEQ, M(NEQ), NLBC, IUSER(*)
REAL (KIND=nag_wp)  YA(NEQ,0:*), GA(NLBC), RUSER(*)
```

1: YA(NEQ, $0:*$) – REAL (KIND=nag_wp) array *Input*

On entry: 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 *Input*

On entry: the number of differential equations.

3: M(NEQ) – INTEGER array *Input*

On entry: M(i) 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) – REAL (KIND=nag_wp) array *Output*

On exit: GA(i) must contain $g_i(z(y(a)))$, for $i = 1,2,\ldots,$NLBC.

6: IUSER($*$) – INTEGER array *User Workspace*
7: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

GAFUN is called with the parameters IUSER and RUSER as supplied to D02TLF. You are free to use the arrays IUSER and RUSER to supply information to GAFUN as an alternative to using COMMON global variables.

GAFUN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02TLF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: GBFUN – SUBROUTINE, supplied by the user. *External Procedure*

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:

```
SUBROUTINE GBFUN (YB, NEQ, M, NRBC, GB, IUSER, RUSER)
INTEGER              NEQ, M(NEQ), NRBC, IUSER(*)
REAL (KIND=nag_wp)  YB(NEQ,0:*), GB(NRBC), RUSER(*)
```

1: YB(NEQ, $0:*$) – REAL (KIND=nag_wp) array *Input*

On entry: 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 *Input*

On entry: the number of differential equations.

3: M(NEQ) – INTEGER array *Input*

On entry: M(i) 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) – REAL (KIND=nag_wp) array *Output*

On exit: GB(i) must contain $\bar{g}_i(z(y(b)))$, for $i = 1, 2, \ldots,$ NRBC.

6: IUSER$(*)$ – INTEGER array *User Workspace*

7: RUSER$(*)$ – REAL (KIND=nag_wp) array *User Workspace*

GBFUN is called with the parameters IUSER and RUSER as supplied to D02TLF. You are free to use the arrays IUSER and RUSER to supply information to GBFUN as an alternative to using COMMON global variables.

GBFUN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02TLF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: GAJAC – SUBROUTINE, supplied by the user. *External Procedure*

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:

```
SUBROUTINE GAJAC (YA, NEQ, M, NLBC, DGADY, IUSER, RUSER)
INTEGER             NEQ, M(NEQ), NLBC, IUSER(*)
REAL (KIND=nag_wp) YA(NEQ,0:*), DGADY(NLBC,NEQ,0:*), RUSER(*)
```

1: YA(NEQ, $0 : *$) – REAL (KIND=nag_wp) array *Input*

On entry: 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 *Input*

On entry: the number of differential equations.

3: M(NEQ) – INTEGER array *Input*

On entry: M(i) 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(NLBC, NEQ, $0 : *$) – REAL (KIND=nag_wp) array *Input/Output*

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$(j) - 1$. Only nonzero partial derivatives need be set.

6: IUSER$(*)$ – INTEGER array *User Workspace*

7: RUSER$(*)$ – REAL (KIND=nag_wp) array *User Workspace*

GAJAC is called with the parameters IUSER and RUSER as supplied to D02TLF. You are free to use the arrays IUSER and RUSER to supply information to GAJAC as an alternative to using COMMON global variables.

GAJAC must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02TLF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: GBJAC – SUBROUTINE, supplied by the user. *External Procedure*

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:

```
SUBROUTINE GBJAC (YB, NEQ, M, NRBC, DGBDY, IUSER, RUSER)
INTEGER             NEQ, M(NEQ), NRBC, IUSER(*)
REAL (KIND=nag_wp) YB(NEQ,0:*), DGBDY(NRBC,NEQ,0:*), RUSER(*)
```

1: YB(NEQ, 0 : $*$) – REAL (KIND=nag_wp) array *Input*

On entry: 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 *Input*

On entry: the number of differential equations.

3: M(NEQ) – INTEGER array *Input*

On entry: M(i) 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(NRBC, NEQ, 0 : $*$) – REAL (KIND=nag_wp) array *Input/Output*

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$(j) - 1$. Only nonzero partial derivatives need be set.

6: IUSER($*$) – INTEGER array *User Workspace*
7: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

GBJAC is called with the parameters IUSER and RUSER as supplied to D02TLF. You are free to use the arrays IUSER and RUSER to supply information to GBJAC as an alternative to using COMMON global variables.

GBJAC must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02TLF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: GUESS – SUBROUTINE, supplied by the user. *External Procedure*

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$(i) - 1$. 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$(i) - 1$. You should **not** call FFUN to compute $y_i^{(m_i)}$.

If D02TLF is being used in conjunction with D02TXF as part of a continuation process, then GUESS is not called by D02TLF after the call to D02TXF.

The specification of GUESS is:

```
SUBROUTINE GUESS (X, NEQ, M, Y, DYM, IUSER, RUSER)
INTEGER             NEQ, M(NEQ), IUSER(*)
REAL (KIND=nag_wp) X, Y(NEQ,0:*), DYM(NEQ), RUSER(*)
```

1: X – REAL (KIND=nag_wp) *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) – INTEGER array *Input*

On entry: $\mathrm{M}(i)$ contains m_i, the order of the ith differential equation, for $i = 1, 2, \ldots, \mathrm{NEQ}$.

4: $\mathrm{Y}(\mathrm{NEQ}, 0 : *)$ – REAL (KIND=nag_wp) array *Output*

On exit: $\mathrm{Y}(i, j)$ must contain $y_i^{(j)}(x)$, for $i = 1, 2, \ldots, \mathrm{NEQ}$ and $j = 0, 1, \ldots, \mathrm{M}(i) - 1$.
Note: $y_i^{(0)}(x) = y_i(x)$.

5: DYM(NEQ) – REAL (KIND=nag_wp) array *Output*

On exit: $\mathrm{DYM}(i)$ must contain $y_i^{(m_i)}(x)$, for $i = 1, 2, \ldots, \mathrm{NEQ}$.

6: $\mathrm{IUSER}(*)$ – INTEGER array *User Workspace*
7: $\mathrm{RUSER}(*)$ – REAL (KIND=nag_wp) array *User Workspace*

GUESS is called with the parameters IUSER and RUSER as supplied to D02TLF. You are free to use the arrays IUSER and RUSER to supply information to GUESS as an alternative to using COMMON global variables.

GUESS must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02TLF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

8: $\mathrm{RCOMM}(*)$ – REAL (KIND=nag_wp) array *Communication 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 D02TVF.

On entry: this must be the same array as supplied to D02TVF and **must** remain unchanged between calls.

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

9: $\mathrm{ICOMM}(*)$ – INTEGER array *Communication 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 D02TVF.

On entry: this must be the same array as supplied to D02TVF and **must** remain unchanged between calls.

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

10: IUSER($*$) – INTEGER array *User Workspace*

11: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

IUSER and RUSER are not used by D02TLF, but are passed directly to FFUN, FJAC, GAFUN, GBFUN, GAJAC, GBJAC and GUESS and may be used to pass information to these routines as an alternative to using COMMON global variables.

12: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, because for this routine the values of the output parameters may be useful even if IFAIL $\neq 0$ on exit, the recommended value is -1. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Note: D02TLF may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the routine:

IFAIL $= 1$

Either the setup routine 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 routines 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 routines 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: $\langle$*value*$\rangle$.
Results have been generated which may be useful.
Try increasing this number or relaxing the error requirements.

IFAIL $= -99$

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

See Section 3.8 in the Essential Introduction for further information.

IFAIL $= -399$

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

See Section 3.7 in the Essential Introduction for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.6 in the Essential Introduction for further information.

7 Accuracy

The accuracy of the solution is determined by the parameter TOLS in the prior call to D02TVF (see Sections 3 and 9 in D02TVF 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 D02TZF.

8 Parallelism and Performance

D02TLF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

D02TLF 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 routine. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

If D02TLF returns with IFAIL $= 2$, 3, 4 or 5 and the call to D02TLF was a part of some continuation procedure for which successful calls to D02TLF have already been made, then it is possible that the adjustment(s) to the continuation parameter(s) between calls to D02TLF 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 D02TVF, D02TXF, D02TYF and D02TZF, 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

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

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 D02TXF for NMESH, IPMESH and MESH in the call to D02TXF 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:

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

10.1 Program Text

```
!   D02TLF Example Program Text
!   Mark 25 Release. NAG Copyright 2014.

    Module d02tlfe_mod

!     D02TLF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                            :: ffun, fjac, gafun, gajac, gbfun, &
                                           gbjac, guess
!     .. Parameters ..
      Integer, Parameter, Public        :: mmax = 3, neq = 3, nin = 5,      &
                                           nlbc = 3, nout = 6, nrbc = 3
!     .. Local Scalars ..
      Real (Kind=nag_wp), Public, Save  :: omega
      Real (Kind=nag_wp), Public, Save  :: one = 1.0_nag_wp
      Real (Kind=nag_wp), Public, Save  :: sqrofr
!     .. Local Arrays ..
      Integer, Public, Save             :: m(neq) = (/1,3,2/)
    Contains
      Subroutine ffun(x,y,neq,m,f,iuser,ruser)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: x
        Integer, Intent (In)                 :: neq
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: f(neq)
        Real (Kind=nag_wp), Intent (Inout)   :: ruser(*)
        Real (Kind=nag_wp), Intent (In)      :: y(neq,0:*)
        Integer, Intent (Inout)              :: iuser(*)
        Integer, Intent (In)                 :: m(neq)
!       .. Executable Statements ..
        f(1) = y(2,0)
        f(2) = -(y(1,0)*y(2,2)+y(3,0)*y(3,1))*sqrofr
        f(3) = (y(2,0)*y(3,0)-y(1,0)*y(3,1))*sqrofr
        Return
      End Subroutine ffun
      Subroutine fjac(x,y,neq,m,dfdy,iuser,ruser)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)       :: x
        Integer, Intent (In)                  :: neq
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout)    :: dfdy(neq,neq,0:*), ruser(*)
        Real (Kind=nag_wp), Intent (In)       :: y(neq,0:*)
        Integer, Intent (Inout)               :: iuser(*)
        Integer, Intent (In)                  :: m(neq)
!       .. Executable Statements ..
        dfdy(1,2,0) = one
        dfdy(2,1,0) = -y(2,2)*sqrofr
        dfdy(2,2,2) = -y(1,0)*sqrofr
        dfdy(2,3,0) = -y(3,1)*sqrofr
        dfdy(2,3,1) = -y(3,0)*sqrofr
        dfdy(3,1,0) = -y(3,1)*sqrofr
        dfdy(3,2,0) = y(3,0)*sqrofr
        dfdy(3,3,0) = y(2,0)*sqrofr
        dfdy(3,3,1) = -y(1,0)*sqrofr
        Return
      End Subroutine fjac
      Subroutine gafun(ya,neq,m,nlbc,ga,iuser,ruser)

!       .. Scalar Arguments ..
        Integer, Intent (In)                  :: neq, nlbc
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)      :: ga(nlbc)
        Real (Kind=nag_wp), Intent (Inout)    :: ruser(*)
        Real (Kind=nag_wp), Intent (In)       :: ya(neq,0:*)
        Integer, Intent (Inout)               :: iuser(*)
        Integer, Intent (In)                  :: m(neq)
!       .. Executable Statements ..
        ga(1) = ya(1,0)
        ga(2) = ya(2,0)
        ga(3) = ya(3,0) - omega
        Return
      End Subroutine gafun
      Subroutine gbfun(yb,neq,m,nrbc,gb,iuser,ruser)

!       .. Scalar Arguments ..
        Integer, Intent (In)                  :: neq, nrbc
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)      :: gb(nrbc)
        Real (Kind=nag_wp), Intent (Inout)    :: ruser(*)
        Real (Kind=nag_wp), Intent (In)       :: yb(neq,0:*)
        Integer, Intent (Inout)               :: iuser(*)
        Integer, Intent (In)                  :: m(neq)
!       .. Executable Statements ..
        gb(1) = yb(1,0)
        gb(2) = yb(2,0)
        gb(3) = yb(3,0) + omega
        Return
      End Subroutine gbfun
      Subroutine gajac(ya,neq,m,nlbc,dgady,iuser,ruser)

!       .. Scalar Arguments ..
        Integer, Intent (In)                  :: neq, nlbc
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout)    :: dgady(nlbc,neq,0:*), ruser(*)
        Real (Kind=nag_wp), Intent (In)       :: ya(neq,0:*)
        Integer, Intent (Inout)               :: iuser(*)
        Integer, Intent (In)                  :: m(neq)
!       .. Executable Statements ..
        dgady(1,1,0) = one
        dgady(2,2,0) = one
        dgady(3,3,0) = one
        Return
      End Subroutine gajac
      Subroutine gbjac(yb,neq,m,nrbc,dgbdy,iuser,ruser)

!       .. Scalar Arguments ..
        Integer, Intent (In)                  :: neq, nrbc
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout)    :: dgbdy(nrbc,neq,0:*), ruser(*)
        Real (Kind=nag_wp), Intent (In)       :: yb(neq,0:*)
        Integer, Intent (Inout)               :: iuser(*)
        Integer, Intent (In)                  :: m(neq)
!       .. Executable Statements ..
        dgbdy(1,1,0) = one
        dgbdy(2,2,0) = one
        dgbdy(3,3,0) = one
        Return
      End Subroutine gbjac
      Subroutine guess(x,neq,m,y,dym,iuser,ruser)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)       :: x
        Integer, Intent (In)                  :: neq
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)      :: dym(neq)
        Real (Kind=nag_wp), Intent (Inout)    :: ruser(*), y(neq,0:*)
        Integer, Intent (Inout)               :: iuser(*)
        Integer, Intent (In)                  :: m(neq)
!       .. Executable Statements ..
        y(1,0) = -(x-0.5_nag_wp)*(x*(x-one))**2
        y(2,0) = -x*(x-one)*(5._nag_wp*x*(x-one)+one)
        y(2,1) = -(2._nag_wp*x-one)*(10.0_nag_wp*x*(x-one)+one)
        y(2,2) = -12.0_nag_wp*(5._nag_wp*x*(x-one)+x)
        y(3,0) = -8.0_nag_wp*omega*(x-0.5_nag_wp)**3
        y(3,1) = -24.0_nag_wp*omega*(x-0.5_nag_wp)**2
        dym(1) = y(2,0)
        dym(2) = -120.0_nag_wp*(x-0.5_nag_wp)
        dym(3) = -56.0_nag_wp*omega*(x-0.5_nag_wp)
        Return
      End Subroutine guess
    End Module d02tlfe_mod
    Program d02tlfe

!     D02TLF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02tlf, d02tvf, d02txf, d02tyf, d02tzf, nag_wp
      Use d02tlfe_mod, Only: ffun, fjac, gafun, gajac, gbfun, gbjac, guess, m, &
                             mmax, neq, nin, nlbc, nout, nrbc, omega, one,     &
                             sqrofr
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                    :: dx, ermx, r
      Integer                               :: i, iermx, ifail, ijermx, j,      &
                                               licomm, lrcomm, mxmesh, ncol,    &
                                               ncont, nmesh
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable       :: mesh(:), rcomm(:), tol(:), y(:,:)
      Real (Kind=nag_wp)                    :: ruser(1)
      Integer, Allocatable                  :: icomm(:), ipmesh(:)
      Integer                               :: iuser(2)
!     .. Intrinsic Procedures ..
      Intrinsic                             :: real, sqrt
!     .. Executable Statements ..
      Write (nout,*) 'D02TLF Example Program Results'
      Write (nout,*)
!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) ncol, nmesh, mxmesh

      Allocate (mesh(mxmesh),tol(neq),y(neq,0:mmax-1),ipmesh(mxmesh))

      Read (nin,*) omega
      Read (nin,*) tol(1:neq)

      dx = one/real(nmesh-1,kind=nag_wp)
      mesh(1) = 0.0_nag_wp
      Do i = 2, nmesh - 1
        mesh(i) = mesh(i-1) + dx
      End Do
      mesh(nmesh) = one

      ipmesh(1) = 1
      ipmesh(2:nmesh-1) = 2
      ipmesh(nmesh) = 1

!     Workspace query to get size of rcomm and icomm
      ifail = 0
      Call d02tvf(neq,m,nlbc,nrbc,ncol,tol,mxmesh,nmesh,mesh,ipmesh,ruser,0, &
        iuser,2,ifail)
      lrcomm = iuser(1)
      licomm = iuser(2)
      Allocate (rcomm(lrcomm),icomm(licomm))

!     Initialize integrator for given problem.
      ifail = 0
      Call d02tvf(neq,m,nlbc,nrbc,ncol,tol,mxmesh,nmesh,mesh,ipmesh,rcomm, &
        lrcomm,icomm,licomm,ifail)

!     Number of continuation steps (last r=100**ncont, sqrofr=10**ncont)
      Read (nin,*) ncont
!     Initialize problem continuation parameter.
      Read (nin,*) r
      sqrofr = sqrt(r)

contn: Do j = 1, ncont
        Write (nout,99999) tol(1), r

!       Solve problem.
        ifail = -1
        Call d02tlf(ffun,fjac,gafun,gbfun,gajac,gbjac,guess,rcomm,icomm,iuser, &
          ruser,ifail)

!       Extract mesh
        ifail = -1
        Call d02tzf(mxmesh,nmesh,mesh,ipmesh,ermx,iermx,ijermx,rcomm,icomm, &
          ifail)
        If (ifail==1) Then
          Exit contn
        End If

!       Print mesh and error statistics.
        Write (nout,99998) nmesh, ermx, iermx, ijermx
        Write (nout,99997)(i,ipmesh(i),mesh(i),i=1,nmesh)
!       Print solution components on mesh.
        Write (nout,99996)
        Do i = 1, nmesh
          ifail = 0
          Call d02tyf(mesh(i),y,neq,mmax,rcomm,icomm,ifail)
          Write (nout,99995) mesh(i), y(1:neq,0)
        End Do

        If (j==ncont) Exit contn

!       Modify continuation parameter.
        r = 100.0_nag_wp*r
        sqrofr = sqrt(r)
!       Select mesh for continuation and call continuation primer routine.
        ipmesh(2:nmesh-1) = 2
        ifail = 0
        Call d02txf(mxmesh,nmesh,mesh,ipmesh,rcomm,icomm,ifail)

      End Do contn
```

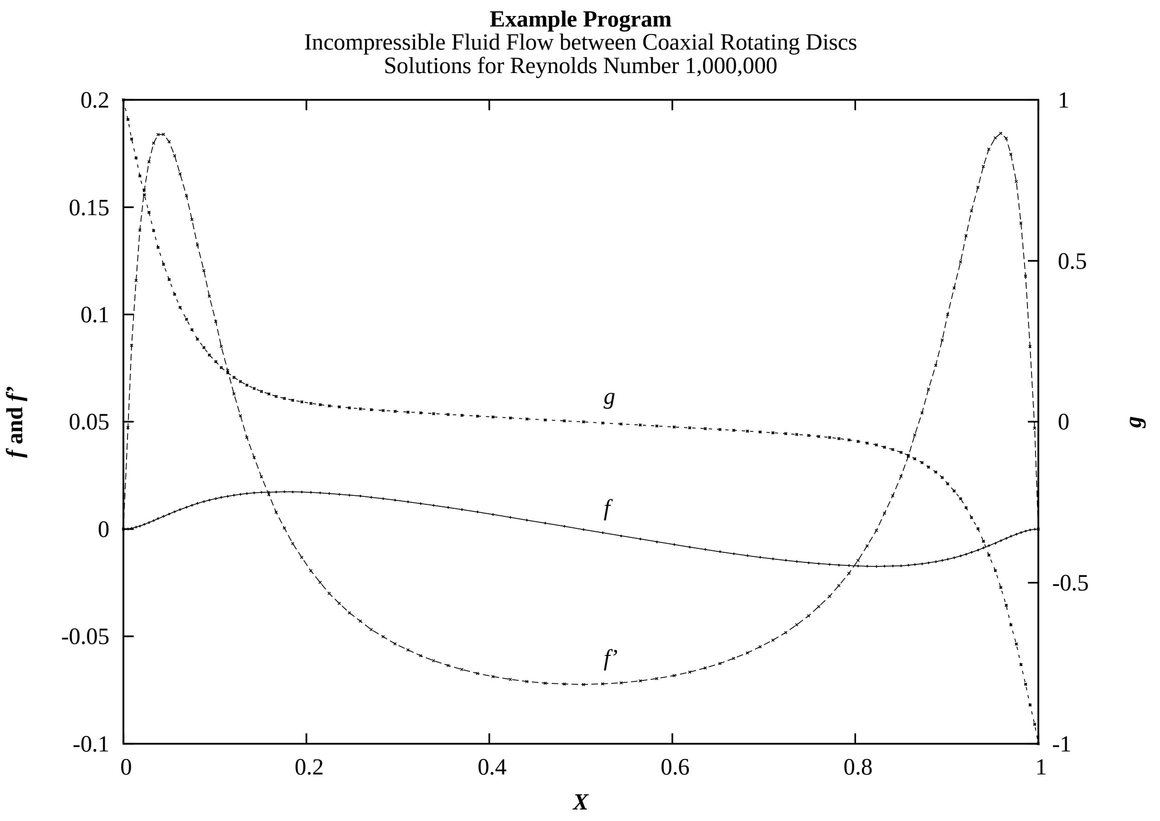
```
99999 Format (/' Tolerance = ',1P,E8.1,'  R = ',E10.3)
99998 Format (/' Used a mesh of ',I4,' points'/' Maximum error = ',E10.2, &
        '  in interval ',I4,' for component ',I4/)
99997 Format (/' Mesh points:'/4(I4,'(',I1,')',E11.4))
99996 Format (/'      x        f        f''       g')
99995 Format (' ',F8.3,1X,3F9.4)
    End Program d02tlfe
```

10.2 Program Data

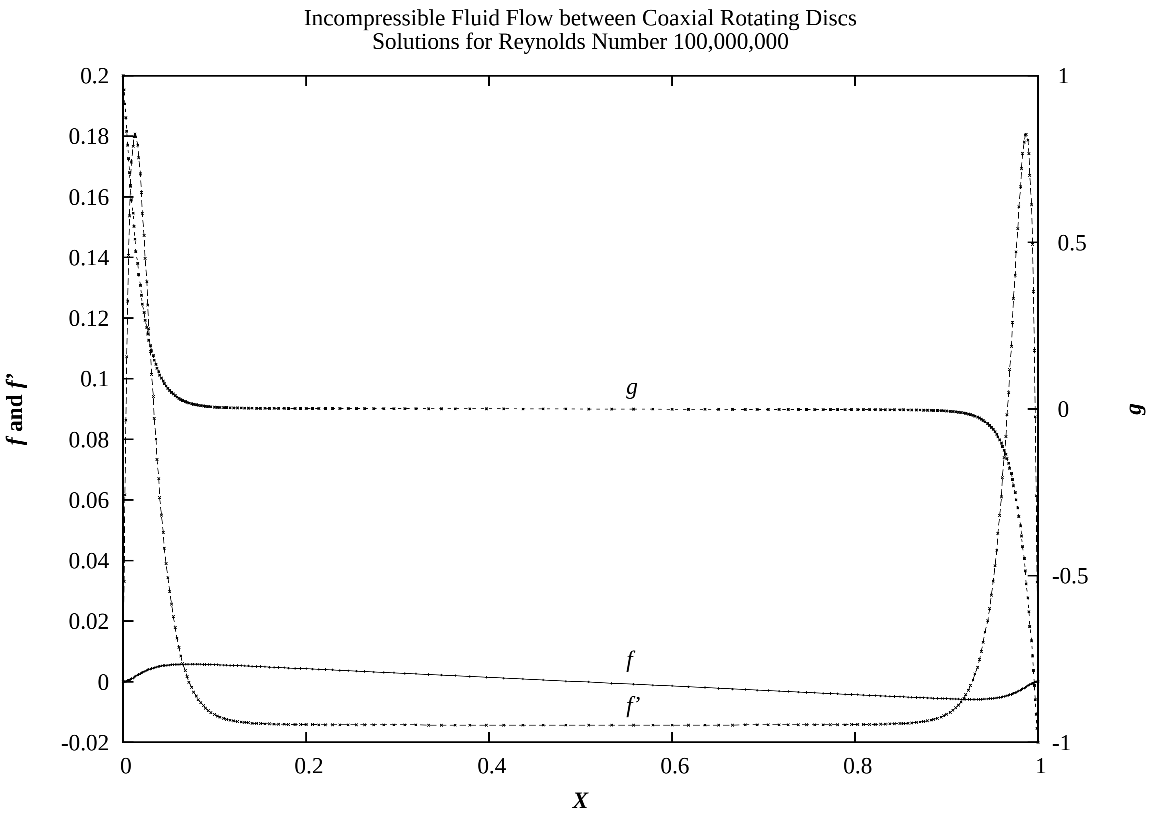
```
D02TLF Example Program Data
   7  11  51                  : ncol, nmesh, mxmesh
   1.0                        : omega
   1.0E-4 1.0E-4 1.0E-4       : tol(1:neq)
   3                          : ncount
   1.0E+6                     : r
```

10.3 Program Results

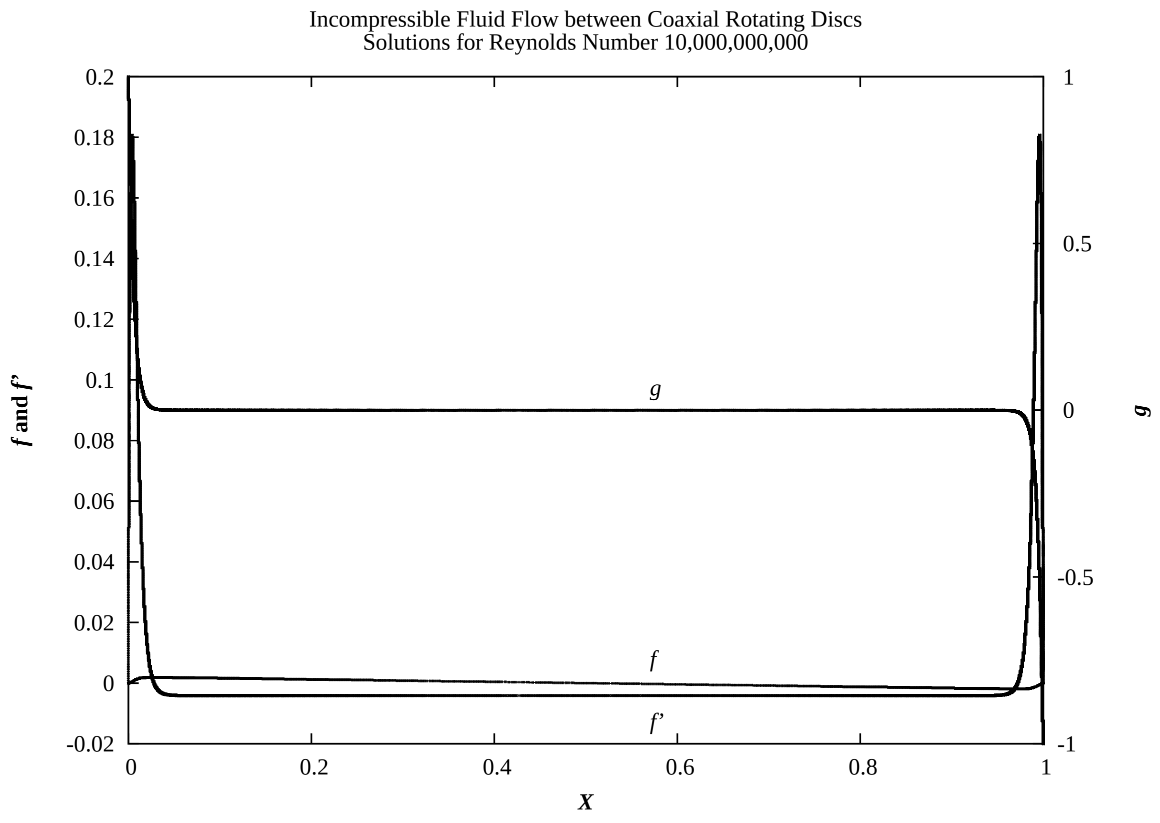
```
 D02TLF Example Program Results


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

 Used a mesh of   21 points
 Maximum error =   0.62E-09  in interval   20 for component    3


 Mesh points:
   1(1) 0.0000E+00   2(3) 0.5000E-01   3(2) 0.1000E+00   4(3) 0.1500E+00
   5(2) 0.2000E+00   6(3) 0.2500E+00   7(2) 0.3000E+00   8(3) 0.3500E+00
   9(2) 0.4000E+00  10(3) 0.4500E+00  11(2) 0.5000E+00  12(3) 0.5500E+00
  13(2) 0.6000E+00  14(3) 0.6500E+00  15(2) 0.7000E+00  16(3) 0.7500E+00
  17(2) 0.8000E+00  18(3) 0.8500E+00  19(2) 0.9000E+00  20(3) 0.9500E+00
  21(1) 0.1000E+01

      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

 Used a mesh of   21 points
 Maximum error =   0.45E-08  in interval    6 for component    3


 Mesh points:
   1(1) 0.0000E+00   2(3) 0.1757E-01   3(2) 0.3515E-01   4(3) 0.5203E-01
   5(2) 0.6891E-01   6(3) 0.8593E-01   7(2) 0.1030E+00   8(3) 0.1351E+00
   9(2) 0.1672E+00  10(3) 0.2306E+00  11(2) 0.2939E+00  12(3) 0.4713E+00
  13(2) 0.6486E+00  14(3) 0.7455E+00  15(2) 0.8423E+00  16(3) 0.8824E+00
  17(2) 0.9225E+00  18(3) 0.9449E+00  19(2) 0.9673E+00  20(3) 0.9836E+00
  21(1) 0.1000E+01

      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

 Used a mesh of   21 points
 Maximum error =   0.31E-05  in interval    7 for component    3


 Mesh points:
   1(1) 0.0000E+00   2(3) 0.6256E-02   3(2) 0.1251E-01   4(3) 0.1851E-01
   5(2) 0.2450E-01   6(3) 0.3076E-01   7(2) 0.3702E-01   8(3) 0.4997E-01
   9(2) 0.6292E-01  10(3) 0.9424E-01  11(2) 0.1256E+00  12(3) 0.4190E+00
  13(2) 0.7125E+00  14(3) 0.8246E+00  15(2) 0.9368E+00  16(3) 0.9544E+00
  17(2) 0.9719E+00  18(3) 0.9803E+00  19(2) 0.9886E+00  20(3) 0.9943E+00
  21(1) 0.1000E+01

      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
```



Example Program
Incompressible Fluid Flow between Coaxial Rotating Discs
Solutions for Reynolds Number 1,000,000
0.2
0.15
0.1
0.05
0
-0.05
-0.1
1
0.5
0
-0.5
-1
0
0.2
0.4
0.6
0.8
1
g
f
f'
f and f'
g
x



Incompressible Fluid Flow between Coaxial Rotating Discs
Solutions for Reynolds Number 100,000,000
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
g
f
f'
f and f'
g
x



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