NAG Library Routine Document

D02TXF

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

D02TXF allows a solution to a nonlinear two-point boundary value problem computed by D02TLF to be used as an initial approximation in the solution of a related nonlinear two-point boundary value problem in a continuation call to D02TLF.

2 Specification

```
SUBROUTINE D02TXF (MXMESH, NMESH, MESH, IPMESH, RCOMM, ICOMM, IFAIL)
INTEGER            MXMESH, NMESH, IPMESH(MXMESH), ICOMM(*), IFAIL
REAL (KIND=nag_wp) MESH(MXMESH), RCOMM(*)
```

3 Description

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

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

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

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

and the right boundary conditions at b as

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

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

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

First, D02TVF must be called to specify the initial mesh, error requirements and other details. 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. D02TYF can be used to compute the approximate solution anywhere on the interval $[a,b]$ using interpolation.

If the boundary value problem being solved is one of a sequence of related problems, for example as part of some continuation process, then D02TXF should be used between calls to D02TLF. This avoids the overhead of a complete initialization when the setup routine D02TVF is used. D02TXF allows the solution values computed in the previous call to D02TLF to be used as an initial approximation for the solution in the next call to D02TLF.

You must specify the new initial mesh. The previous mesh can be obtained by a call to D02TZF. It may be used unchanged as the new mesh, in which case any fixed points in the previous mesh remain as

fixed points in the new mesh. Fixed and other points may be added or subtracted from the mesh by manipulation of the contents of the array argument IPMESH. Initial values for the solution components on the new mesh are computed by interpolation on the values for the solution components on the previous mesh.

The 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 Arguments

1: MXMESH – INTEGER *Input*

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

Constraint: this must be identical to the value supplied for the argument MXMESH in the prior call to D02TVF.

2: NMESH – INTEGER *Input*

On entry: the number of points to be used in the new initial mesh. It is strongly recommended that if this routine is called that the suggested value (see below) for NMESH is used. In this case the arrays MESH and IPMESH returned by D02TZF can be passed to this routine without any modification.

Suggested value: $(n^* + 1)/2$, where n^* is the number of mesh points used in the previous mesh as returned in the argument NMESH of D02TZF.

Constraint: $6 \leq \text{NMESH} \leq (\text{MXMESH} + 1)/2$.

3: MESH(MXMESH) – REAL (KIND=nag_wp) array *Input*

On entry: the NMESH points to be used in the new initial mesh as specified by IPMESH.

Suggested value: the argument MESH returned from a call to D02TZF.

Constraint: $\text{MESH}(i_j) < \text{MESH}(i_{j+1})$, for $j = 1, 2, \ldots, \text{NMESH} - 1$, the values of $i_1, i_2, \ldots, i_{\text{NMESH}}$ are defined in IPMESH.

$\text{MESH}(i_1)$ must contain the left boundary point, a, and $\text{MESH}(i_{\text{NMESH}})$ must contain the right boundary point, b, as specified in the previous call to D02TVF.

4: IPMESH(MXMESH) – INTEGER array *Input*

On entry: specifies the points in MESH to be used as the new initial mesh. Let $\{i_j : j = 1, 2, \ldots, \text{NMESH}\}$ be the set of array indices of IPMESH such that $\text{IPMESH}(i_j) = 1$ or 2 and $1 = i_1 < i_2 < \cdots < i_{\text{NMESH}}$. Then $\text{MESH}(i_j)$ will be included in the new initial mesh.

If $\text{IPMESH}(i_j) = 1$, $\text{MESH}(i_j)$ will be a fixed point in the new initial mesh.

If $\text{IPMESH}(k) = 3$ for any k, then $\text{MESH}(k)$ will not be included in the new mesh.

Suggested value: the argument IPMESH returned in a call to D02TZF.

Constraints:

IPMESH(k) = 1, 2 or 3, for $k = 1, 2, \ldots, i_{\mathrm{NMESH}}$;
IPMESH(1) = IPMESH(i_{NMESH}) = 1.

5: 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 D02TLF.

On entry: this must be the same array as supplied to D02TLF and **must** remain unchanged between calls.

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

6: 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 D02TLF.

On entry: this must be the same array as supplied to D02TLF and **must** remain unchanged between calls.

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

7: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation 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, if you are not familiar with this argument, the recommended value is 0. **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).

Errors or warnings detected by the routine:

IFAIL = 1

An element of IPMESH was set to -1 before NMESH elements containing 1 or 2 were detected.

Expected ⟨*value*⟩ elements of IPMESH to be 1 or 2, but ⟨*value*⟩ such elements found.

IPMESH(i) $\neq -1$, 1, 2 or 3 for some i.

On entry, IPMESH(1) = ⟨*value*⟩.
Constraint: IPMESH(1) = 1.

On entry, MXMESH = ⟨*value*⟩ and MXMESH = ⟨*value*⟩ in D02TVF.
Constraint: MXMESH = MXMESH in D02TVF.

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

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

The entries in MESH are not strictly increasing.

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

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

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

The solver routine does not appear to have been called.

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

IFAIL $= -99$

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

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

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

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

D02TXF is not threaded in any implementation.

9 Further Comments

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

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

– maintain fixed points in the mesh using the argument IPMESH to ensure that the remeshing process does not inadvertently remove mesh points from areas of known interest.

In the absence of any other information about the expected behaviour of the solution, using the values suggested in Section 5 for NMESH, IPMESH and MESH is strongly recommended.

10 Example

This example illustrates the use of continuation, solution on an infinite range, and solution of a system of two differential equations of orders 3 and 2. See also D02TLF, D02TVF, D02TYF and D02TZF, for the illustration of other facilities.

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

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

with boundary conditions

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

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

Some solutions of interest are for $\gamma = 1$, small n and $s \to 0$. An added complication is the infinite range, which we approximate by $[0, L]$. We choose $n = 0.2$ and first solve for $L = 60.0, s = 0.24$ using the initial approximations $f(x) = -x^2e^{-x}$ and $g(x) = 1.0 - e^{-x}$, which satisfy the boundary conditions, on a uniform mesh of 21 points. Simple continuation on the parameters L and s using the values $L = 120.0, s = 0.144$ and then $L = 240.0, s = 0.0864$ is used to compute further solutions. We use the suggested values for NMESH, IPMESH and MESH in the call to D02TXF prior to a continuation call, that is only every second point of the preceding mesh is used.

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

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

10.1 Program Text

```
!   D02TXF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module d02txfe_mod

!     D02TXF 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 ..
      Real (Kind=nag_wp), Parameter    :: half = 0.5_nag_wp
      Real (Kind=nag_wp), Parameter    :: three = 3.0_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: one = 1.0_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: two = 2.0_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: xsplit = 30.0_nag_wp
      Integer, Parameter, Public       :: m1 = 3, m2 = 2, mmax = 3, neq = 2,   &
                                             nin = 5, nlbc = 3, nleft = 15,      &
                                             nout = 6, nrbc = 2, nright = 10
!     .. Local Scalars ..
      Real (Kind=nag_wp), Public, Save :: el, en, s
    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)
!       .. Local Scalars ..
        Real (Kind=nag_wp)              :: t1, y11, y20
!       .. Executable Statements ..
        t1 = half*(three-en)*y(1,0)
        y11 = y(1,1)
        y20 = y(2,0)
        f(1) = (el**3)*(one-y20**2) + (el**2)*s*y11 - el*(t1*y(1,2)+en*y11**2)
        f(2) = (el**2)*s*(y20-one) - el*(t1*y(2,1)+(en-one)*y11*y20)
        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) = -two*el**3*y(2,0)
        dfdy(1,1,0) = -el*half*(three-en)*y(1,2)
        dfdy(1,1,1) = el**2*s - el*two*en*y(1,1)
        dfdy(1,1,2) = -el*half*(three-en)*y(1,0)
        dfdy(2,2,0) = el**2*s - el*(en-one)*y(1,1)
        dfdy(2,2,1) = -el*half*(three-en)*y(1,0)
        dfdy(2,1,0) = -el*half*(three-en)*y(2,1)
        dfdy(2,1,1) = -el*(en-one)*y(2,0)
        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(1,1)
        ga(3) = ya(2,0)
        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(1,1)
        gb(2) = yb(2,0) - one
        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(2,1,1) = one
        dgady(3,2,0) = 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(1,1,1) = one
        dgbdy(2,2,0) = 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)
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: ex, expmx
!       .. Intrinsic Procedures ..
        Intrinsic                      :: exp
!       .. Executable Statements ..
        ex = x*el
        expmx = exp(-ex)
        y(1,0) = -ex**2*expmx
        y(1,1) = (-two*ex+ex**2)*expmx
        y(1,2) = (-two+4.0E0_nag_wp*ex-ex**2)*expmx
        y(2,0) = one - expmx
        y(2,1) = expmx
        dym(1) = (6.0E0_nag_wp-6.0E0_nag_wp*ex+ex**2)*expmx
        dym(2) = -expmx
        Return
      End Subroutine guess
    End Module d02txfe_mod
    Program d02txfe

!     D02TXF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02tlf, d02tvf, d02txf, d02tyf, d02tzf, nag_wp
      Use d02txfe_mod, Only: el, en, ffun, fjac, gafun, gajac, gbfun, gbjac,   &
                             guess, m1, m2, mmax, neq, nin, nlbc, nleft, nout, &
                             nrbc, nright, one, s, two, xsplit
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: dx, el_init, ermx, s_init, xx
      Integer                          :: i, iermx, ifail, ijermx, j, licomm,  &
                                          lrcomm, mxmesh, ncol, ncont, nmesh
!     .. Local Arrays ..
```

```
      Real (Kind=nag_wp), Allocatable  :: mesh(:), rcomm(:)
      Real (Kind=nag_wp)               :: ruser(1), tol(neq), y(neq,0:mmax-1)
      Integer, Allocatable             :: icomm(:), ipmesh(:)
      Integer                          :: iuser(2), m(neq)
!     .. Intrinsic Procedures ..
      Intrinsic                        :: min, real
!     .. Executable Statements ..
      Write (nout,*) 'D02TXF Example Program Results'
      Write (nout,*)
!     Skip heading in data file
      Read (nin,*)
!     Read method parameters
      Read (nin,*) ncol, nmesh, mxmesh
      Read (nin,*) tol(1:neq)

      Allocate (mesh(mxmesh),ipmesh(mxmesh))
!     Read problem (initial) parameters
      Read (nin,*) en, el_init, s_init
!     Initialize data
      el = el_init
      s = s_init
      m(1) = m1
      m(2) = m2

      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) = 1.0_nag_wp

      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))

!     ifail: behaviour on error exit
!             =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
      ifail = 0
      Call d02tvf(neq,m,nlbc,nrbc,ncol,tol,mxmesh,nmesh,mesh,ipmesh,rcomm,     &
        lrcomm,icomm,licomm,ifail)

!     Initialize number of continuation steps in el and s
      Read (nin,*) ncont
cont: Do j = 1, ncont
        Write (nout,99997) tol(1), el, s

!       Solve
        ifail = -1
        Call d02tlf(ffun,fjac,gafun,gbfun,gajac,gbjac,guess,rcomm,icomm,iuser, &
          ruser,ifail)
        If (ifail/=0) Then
          Exit cont
        End If

!       Extract mesh
        ifail = 0
        Call d02tzf(mxmesh,nmesh,mesh,ipmesh,ermx,iermx,ijermx,rcomm,icomm,    &
          ifail)

        Write (nout,99996) nmesh, ermx, iermx, ijermx
!       Print solution components on mesh
        Write (nout,99999)
```

```
!       Left side domain [0,xsplit], evaluate at nleft+1 uniform grid points.
        dx = xsplit/real(nleft,kind=nag_wp)/el
        xx = 0.0_nag_wp
        Do i = 0, nleft
          ifail = 0
          Call d02tyf(xx,y,neq,mmax,rcomm,icomm,ifail)
          Write (nout,99998) xx*el, y(1,0), y(2,0)
          xx = xx + dx
        End Do

!       Right side domain (xsplit,L], evaluate at nright uniform grid points.
        dx = (el-xsplit)/real(nright,kind=nag_wp)/el
        xx = xsplit/el
        Do i = 1, nright
          xx = min(1.0_nag_wp,xx+dx)
          ifail = 0
          Call d02tyf(xx,y,neq,mmax,rcomm,icomm,ifail)
          Write (nout,99998) xx*el, y(1,0), y(2,0)
        End Do

!       Select mesh for continuation and update continuation parameters.
        If (j<ncont) Then
          el = two*el
          s = 0.6_nag_wp*s
          nmesh = (nmesh+1)/2
          ifail = 0
          Call d02txf(mxmesh,nmesh,mesh,ipmesh,rcomm,icomm,ifail)
        End If
      End Do cont

99999 Format (/,' Solution on original interval:',/,6X,'x',8X,'f',10X,'g')
99998 Format (1X,F8.2,2(1X,F10.4))
99997 Format (/,/,' Tolerance = ',E8.1,'  L = ',F8.3,'  S =',F7.4)
99996 Format (/,' Used a mesh of ',I4,' points',/,' Maximum error = ',E10.2,   &
        '  in interval ',I4,' for component ',I4)
    End Program d02txfe
```

10.2 Program Data

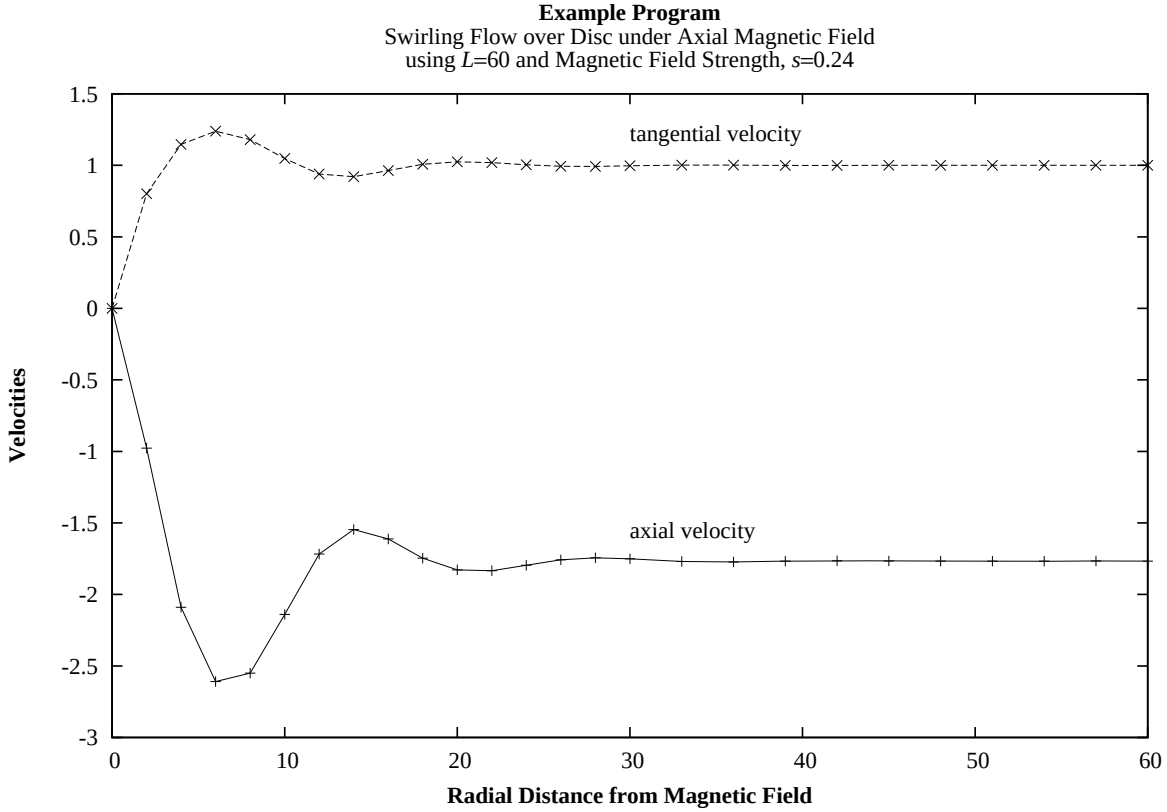
```
D02TXF Example Program Data
   6  21  250    : (method parameters) ncol, nmesh, mxmesh
   1.0E-5 1.0E-5 : tolerances
   0.2 60.0 0.24 : (problem parameters) en, el_init, s_init
   3             : ncont
```

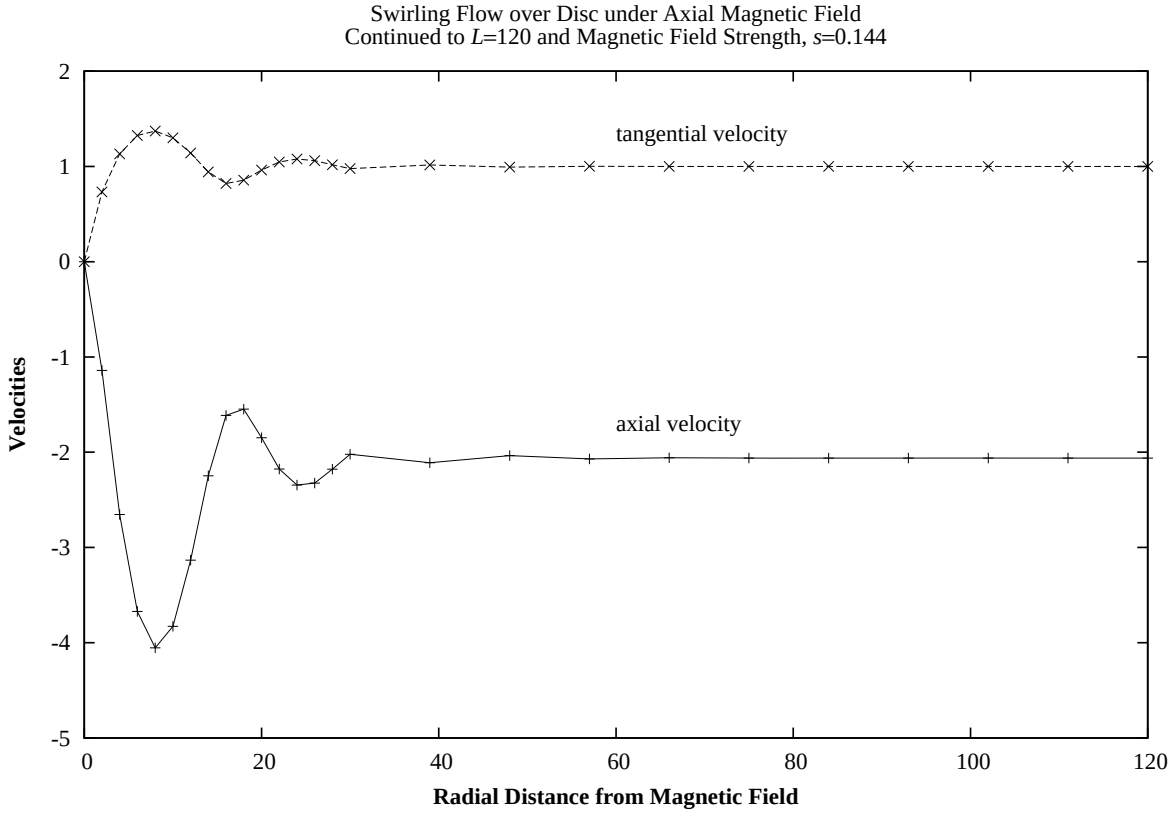
10.3 Program Results

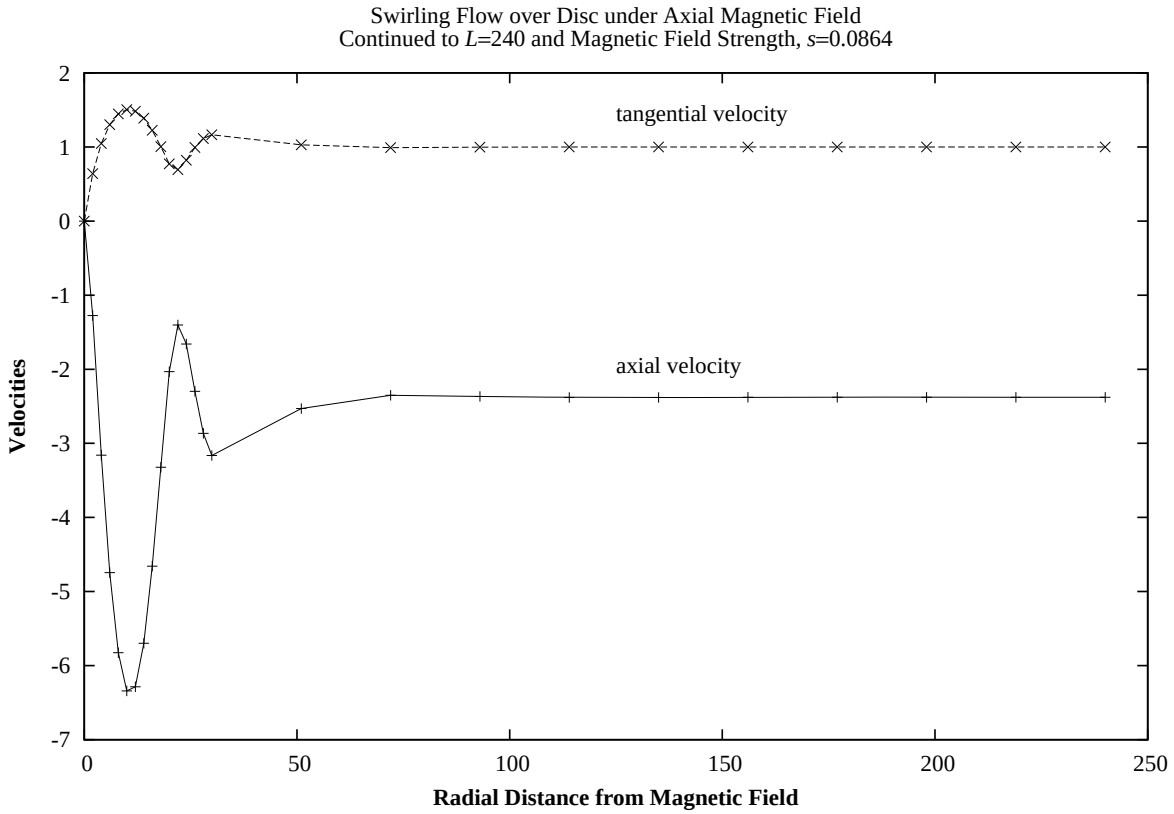
```
 D02TXF Example Program Results



 Tolerance =  0.1E-04  L =   60.000  S = 0.2400

 Used a mesh of   21 points
 Maximum error =   0.27E-07  in interval    7 for component    1

 Solution on original interval:
      x        f          g
     0.00     0.0000     0.0000
     2.00    -0.9769     0.8011
     4.00    -2.0900     1.1459
     6.00    -2.6093     1.2389
     8.00    -2.5498     1.1794
    10.00    -2.1397     1.0478
    12.00    -1.7176     0.9395
    14.00    -1.5465     0.9206
    16.00    -1.6127     0.9630
    18.00    -1.7466     1.0068
    20.00    -1.8286     1.0244
    22.00    -1.8338     1.0185
```

```
   24.00    -1.7956     1.0041
   26.00    -1.7582     0.9940
   28.00    -1.7445     0.9926
   30.00    -1.7515     0.9965
   33.00    -1.7695     1.0019
   36.00    -1.7730     1.0018
   39.00    -1.7673     0.9998
   42.00    -1.7645     0.9993
   45.00    -1.7659     0.9999
   48.00    -1.7672     1.0002
   51.00    -1.7671     1.0001
   54.00    -1.7666     0.9999
   57.00    -1.7665     0.9999
   60.00    -1.7666     1.0000


 Tolerance =  0.1E-04  L =  120.000  S = 0.1440

 Used a mesh of   21 points
 Maximum error =   0.69E-05  in interval    7 for component    2

 Solution on original interval:
      x        f          g
    0.00     0.0000     0.0000
    2.00    -1.1406     0.7317
    4.00    -2.6531     1.1315
    6.00    -3.6721     1.3250
    8.00    -4.0539     1.3707
   10.00    -3.8285     1.3003
   12.00    -3.1339     1.1407
   14.00    -2.2469     0.9424
   16.00    -1.6146     0.8201
   18.00    -1.5472     0.8549
   20.00    -1.8483     0.9623
   22.00    -2.1761     1.0471
   24.00    -2.3451     1.0778
   26.00    -2.3236     1.0600
   28.00    -2.1784     1.0165
   30.00    -2.0214     0.9775
   39.00    -2.1109     1.0155
   48.00    -2.0362     0.9931
   57.00    -2.0709     1.0023
   66.00    -2.0588     0.9995
   75.00    -2.0616     1.0000
   84.00    -2.0615     1.0001
   93.00    -2.0611     0.9999
  102.00    -2.0614     1.0000
  111.00    -2.0613     1.0000
  120.00    -2.0613     1.0000


 Tolerance =  0.1E-04  L =  240.000  S = 0.0864

 Used a mesh of   81 points
 Maximum error =   0.33E-06  in interval   19 for component    2

 Solution on original interval:
      x        f          g
    0.00     0.0000     0.0000
    2.00    -1.2756     0.6404
    4.00    -3.1604     1.0463
    6.00    -4.7459     1.3011
    8.00    -5.8265     1.4467
   10.00    -6.3412     1.5036
   12.00    -6.2862     1.4824
   14.00    -5.6976     1.3886
   16.00    -4.6568     1.2263
   18.00    -3.3226     1.0042
   20.00    -2.0328     0.7718
   22.00    -1.4035     0.6943
   24.00    -1.6603     0.8218
```

```
 26.00    -2.2975     0.9928
 28.00    -2.8661     1.1139
 30.00    -3.1641     1.1641
 51.00    -2.5307     1.0279
 72.00    -2.3520     0.9919
 93.00    -2.3674     0.9975
114.00    -2.3799     1.0003
135.00    -2.3800     1.0002
156.00    -2.3792     1.0000
177.00    -2.3791     1.0000
198.00    -2.3792     1.0000
219.00    -2.3792     1.0000
240.00    -2.3792     1.0000
```




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


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