NAG Library Routine Document

D02TVF

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

D02TVF is a setup routine which must be called prior to the first call of the nonlinear two-point boundary value solver D02TLF.

2 Specification

```
SUBROUTINE D02TVF (NEQ, M, NLBC, NRBC, NCOL, TOLS, MXMESH, NMESH, MESH,    &
                   IPMESH, RCOMM, LRCOMM, ICOMM, LICOMM, IFAIL)
INTEGER            NEQ, M(NEQ), NLBC, NRBC, NCOL, MXMESH, NMESH,           &
                   IPMESH(MXMESH), LRCOMM, ICOMM(LICOMM), LICOMM,          &
                   IFAIL
REAL (KIND=nag_wp) TOLS(NEQ), MESH(MXMESH), RCOMM(LRCOMM)
```

3 Description

D02TVF and its associated routines (D02TLF, D02TXF, 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

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

See Section 9 for information on how boundary value problems of a more general nature can be treated.

D02TVF is used to specify an initial mesh, error requirements and other details. D02TLF is then used to solve the boundary value problem.

The solution routine D02TLF proceeds as follows. A modified Newton method is applied to the equations

$$y_i^{(m_i)}(x) - f_i(x, z(y(x))) = 0, \quad i = 1, \ldots, n$$

and the boundary conditions. To solve these equations numerically the components y_i are approximated by piecewise polynomials v_{ij} using a monomial basis on the jth mesh sub-interval. The coefficients of

the polynomials v_{ij} form the unknowns to be computed. Collocation is applied at Gaussian points

$$v_{ij}^{(m_i)}(x_{jk}) - f_i(x_{jk}, z(v(x_{jk}))) = 0, \quad i = 1, 2, \ldots, n,$$

where x_{jk} is the kth collocation point in the jth mesh sub-interval. Continuity at the mesh points is imposed, that is

$$v_{ij}(x_{j+1}) - v_{i,j+1}(x_{j+1}) = 0, \quad i = 1, 2, \ldots, n,$$

where x_{j+1} is the right-hand end of the jth mesh sub-interval. The linearized collocation equations and boundary conditions, together with the continuity conditions, form a system of linear algebraic equations, an almost block diagonal system which is solved using special linear solvers. To start the modified Newton process, an approximation to the solution on the initial mesh must be supplied via the procedure argument GUESS of D02TLF.

The solver attempts to satisfy the conditions

$$\frac{\|y_i - v_i\|}{(1.0 + \|v_i\|)} \leq \text{TOLS}(i), \quad i = 1, 2, \ldots, n, \tag{1}$$

where v_i is the approximate solution for the ith solution component and TOLS is supplied by you. The mesh is refined by trying to equidistribute the estimated error in the computed solution over all mesh sub-intervals, and an extrapolation-like test (doubling the number of mesh sub-intervals) is used to check for (1).

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

Gill P E, Murray W and Wright M H (1981) *Practical Optimization* Academic Press

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

Schwartz I B (1983) Estimating regions of existence of unstable periodic orbits using computer-based techniques *SIAM J. Sci. Statist. Comput.* **20(1)** 106–120

5 Arguments

1: NEQ – INTEGER *Input*

On entry: n, the number of ordinary differential equations to be solved.

Constraint: NEQ ≥ 1.

2: M(NEQ) – INTEGER array *Input*

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

Constraint: $1 \leq \text{M}(i) \leq 4$, for $i = 1, 2, \ldots, n$.

3: NLBC – INTEGER *Input*

On entry: p, the number of left boundary conditions defined at the left-hand end, a (= MESH(1)).

Constraint: NLBC ≥ 1.

4: NRBC – INTEGER *Input*

On entry: q, the number of right boundary conditions defined at the right-hand end, b (= MESH(NMESH)).

Constraints:

NRBC ≥ 1;

$$\text{NLBC} + \text{NRBC} = \sum_{i=1}^{n} \text{M}(i).$$

5: NCOL – INTEGER *Input*

On entry: the number of collocation points to be used in each mesh sub-interval.

Constraint: $m_{\max} \leq \text{NCOL} \leq 7$, where $m_{\max} = \max(\text{M}(i))$.

6: TOLS(NEQ) – REAL (KIND=nag_wp) array *Input*

On entry: TOLS(i) must contain the error requirement for the ith solution component.

Constraint: $100 \times$ ***machine precision*** $<$ TOLS(i) < 1.0, for $i = 1, 2, \ldots, n$.

7: MXMESH – INTEGER *Input*

On entry: the maximum number of mesh points to be used during the solution process.

Constraint: MXMESH $\geq 2 \times$ NMESH $- 1$.

8: NMESH – INTEGER *Input*

On entry: the number of points to be used in the initial mesh of the solution process.

Constraint: NMESH ≥ 6.

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

On entry: the positions of the initial NMESH mesh points. The remaining elements of MESH need not be set. You should try to place the mesh points in areas where you expect the solution to vary most rapidly. In the absence of any other information the points should be equally distributed on $[a, b]$.

MESH(1) must contain the left boundary point, a, and MESH(NMESH) must contain the right boundary point, b.

Constraint: MESH(i) $<$ MESH($i + 1$), for $i = 1, 2, \ldots,$ NMESH $- 1$.

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

On entry: IPMESH(i) specifies whether or not the initial mesh point defined in MESH(i), for $i = 1, 2, \ldots,$ NMESH, should be a fixed point in all meshes computed during the solution process. The remaining elements of IPMESH need not be set.

IPMESH(i) $= 1$
Indicates that MESH(i) should be a fixed point in all meshes.

IPMESH(i) $= 2$
Indicates that MESH(i) is not a fixed point.

Constraints:

IPMESH(1) = 1 and IPMESH(NMESH) = 1, (i.e., the left and right boundary points, a and b, must be fixed points, in all meshes);
IPMESH(i) = 1 or 2, for $i = 2, 3, \ldots,$ NMESH − 1.

11: RCOMM(LRCOMM) – REAL (KIND=nag_wp) array *Communication Array*

On exit: contains information for use by D02TLF. This **must** be the same array as will be supplied to D02TLF. The contents of this array **must** remain unchanged between calls.

12: LRCOMM – INTEGER *Input*

On entry: the dimension of the array RCOMM as declared in the (sub)program from which D02TVF is called. If LRCOMM = 0, a communication array size query is requested. In this case there is an immediate return with communication array dimensions stored in ICOMM; ICOMM(1) contains the required dimension of RCOMM, while ICOMM(2) contains the required dimension of ICOMM.

Constraint: LRCOMM = 0, or
LRCOMM $\geq 51 +$ NEQ $+$ MXMESH $\times (2 + m^* + k_n) - k_n +$ MXMESH/2, where
$m^* = \sum_{i=1}^{n}$ M(i) and $k_n =$ NCOL $\times$ NEQ.

13: ICOMM(LICOMM) – INTEGER array *Communication Array*

On exit: contains information for use by D02TLF. This **must** be the same array as will be supplied to D02TLF. The contents of this array **must** remain unchanged between calls. If LRCOMM = 0, a communication array size query is requested. In this case, on immediate return, ICOMM(1) will contain the required dimension for RCOMM while ICOMM(2) will contain the required dimension for ICOMM.

14: LICOMM – INTEGER *Input*

On entry: the dimension of the array ICOMM as declared in the (sub)program from which D02TVF is called. If LRCOMM = 0, a communication array size query is requested. In this case ICOMM need only be of dimension 2 in order to hold the required communication array dimensions for the given problem and algorithmic parameters.

Constraints:

if LRCOMM = 0, LICOMM ≥ 2;
otherwise LICOMM $\geq 23 +$ NEQ $+$ MXMESH.

15: 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 $= 1$

On entry, IPMESH(1) or IPMESH(NMESH) does not equal 1.

On entry, IPMESH$(i) \neq 1$ or 2 for some i.

On entry, LICOMM $= \langle value \rangle$.
Constraint: LICOMM $\geq \langle value \rangle$.

On entry, LRCOMM $= \langle value \rangle$.
Constraint: LRCOMM $= 0$ or LRCOMM $\geq \langle value \rangle$.

On entry, M$(\langle value \rangle) = \langle value \rangle$.
Constraint: $1 \leq$ M$(i) \leq 4$ for all i.

On entry, MXMESH $= \langle value \rangle$ and NMESH $= \langle value \rangle$.
Constraint: MXMESH $\geq 2 \times$ NMESH $- 1$.

On entry, NCOL $= \langle value \rangle$ and max(M$(i)) = \langle value \rangle$.
Constraint: max(M$(i)) \leq$ NCOL ≤ 7.

On entry, NEQ $= \langle value \rangle$.
Constraint: NEQ ≥ 1.

On entry, NLBC $= \langle value \rangle$, NRBC $= \langle value \rangle$ and sum (M$(i)) = \langle value \rangle$.
Constraint: NLBC + NRBC $=$ sum (M(i)).

On entry, NLBC $= \langle value \rangle$ and NRBC $= \langle value \rangle$.
Constraint: NLBC ≥ 1 and NRBC ≥ 1.

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

On entry, the elements of MESH are not strictly increasing.

On entry, TOLS$(\langle value \rangle) = \langle value \rangle$.
Constraint: TOLS$(i) > \langle value \rangle$ for all i.

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

D02TVF 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:

– use a large value for NCOL;

– cluster the initial 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 before they are detected automatically by the algorithm.

9.1 Nonseparated Boundary Conditions

A boundary value problem with nonseparated boundary conditions can be treated by transformation to an equivalent problem with separated conditions. As a simple example consider the system

$$y_1' = f_1(x, y_1, y_2)$$

$$y_2' = f_2(x, y_1, y_2)$$

on $[a, b]$ subject to the boundary conditions

$$\begin{aligned} g_1(y_1(a)) &= 0 \\ g_2(y_2(a), y_2(b)) &= 0. \end{aligned}$$

By adjoining the trivial ordinary differential equation

$$r' = 0,$$

which implies $r(a) = r(b)$, and letting $r(b) = y_2(b)$, say, we have a new system

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2) \\ y_2' &= f_2(x, y_1, y_2) \\ r' &= 0, \end{aligned}$$

subject to the separated boundary conditions

$$\begin{aligned} g_1(y_1(a)) &= 0 \\ g_2(y_2(a), r(a)) &= 0 \\ y_2(b) - r(b) &= 0. \end{aligned}$$

There is an obvious overhead in adjoining an extra differential equation: the system to be solved is increased in size.

9.2 Multipoint Boundary Value Problems

Multipoint boundary value problems, that is problems where conditions are specified at more than two points, can also be transformed to an equivalent problem with two boundary points. Each sub-interval defined by the multipoint conditions can be transformed onto the interval $[0, 1]$, say, leading to a larger set of differential equations. The boundary conditions of the transformed system consist of the original boundary conditions and the conditions imposed by the requirement that the solution components be continuous at the interior break-points. For example, consider the equation

$$y^{(3)} = f\left(t, y, y^{(1)}, y^{(2)}\right) \quad \text{on} \quad [a, c]$$

subject to the conditions

$$\begin{aligned} y(a) &= A \\ y(b) &= B \\ y^{(1)}(c) &= C \end{aligned}$$

where $a < b < c$. This can be transformed to the system

$$\left.\begin{aligned} y_1^{(3)} &= f\left(t, y_1, y_1^{(1)}, y_1^{(2)}\right) \\ y_2^{(3)} &= f\left(t, y_2, y_2^{(1)}, y_2^{(2)}\right) \end{aligned}\right\} \quad \text{on } [0,1]$$

where

$$\begin{aligned} y_1 &\equiv y \quad \text{on} \quad [a,b] \\ y_2 &\equiv y \quad \text{on} \quad [b,c], \end{aligned}$$

subject to the boundary conditions

$$\begin{aligned} y_1(0) &= A \\ y_1(1) &= B \\ y_2^{(1)}(1) &= C \\ y_2(0) &= B \quad (\text{from } y_1(1) = y_2(0)) \\ y_1^{(1)}(1) &= y_2^{(1)}(0) \\ y_1^{(2)}(1) &= y_2^{(2)}(0). \end{aligned}$$

In this instance two of the resulting boundary conditions are nonseparated but they may next be treated as described above.

9.3 High Order Systems

Systems of ordinary differential equations containing derivatives of order greater than four can always be reduced to systems of order suitable for treatment by D02TVF and its related routines. For example suppose we have the sixth-order equation

$$y^{(6)} = -y.$$

Writing the variables $y_1 = y$ and $y_2 = y^{(4)}$ we obtain the system

$$\begin{aligned} y_1^{(4)} &= y_2 \\ y_2^{(2)} &= -y_1 \end{aligned}$$

which has maximal order four, or writing the variables $y_1 = y$ and $y_2 = y^{(3)}$ we obtain the system

$$\begin{aligned} y_1^{(3)} &= y_2 \\ y_2^{(3)} &= -y_1 \end{aligned}$$

which has maximal order three. The best choice of reduction by choosing new variables will depend on the structure and physical meaning of the system. Note that you will control the error in each of the variables y_1 and y_2. Indeed, if you wish to control the error in certain derivatives of the solution of an equation of order greater than one, then you should make those derivatives new variables.

9.4 Fixed Points and Singularities

The solver routine D02TLF employs collocation at Gaussian points in each sub-interval of the mesh. Hence the coefficients of the differential equations are not evaluated at the mesh points. Thus, fixed points should be specified in the mesh where either the coefficients are singular, or the solution has less smoothness, or where the differential equations should not be evaluated. Singular coefficients at boundary points often arise when physical symmetry is used to reduce partial differential equations to ordinary differential equations. These do not pose a direct numerical problem for using this code but they can severely impact its convergence.

9.5 Numerical Jacobians

The solver routine D02TLF requires an external routine FJAC to evaluate the partial derivatives of f_i with respect to the elements of $z(y)$ ($= \left(y_1, y_1^1, \ldots, y_1^{(m_1-1)}, y_2, \ldots, y_n^{(m_n-1)}\right)$). In cases where the partial derivatives are difficult to evaluate, numerical approximations can be used. However, this approach might have a negative impact on the convergence of the modified Newton method. You could consider the use of symbolic mathematic packages and/or automatic differentiation packages if available to you.

See Section 10 in D02TZF for an example using numerical approximations to the Jacobian. There central differences are used and each f_i is assumed to depend on all the components of z. This requires two evaluations of the system of differential equations for each component of z. The perturbation used depends on the size of each component of z and a minimum quantity dependent on the ***machine precision***. The cost of this approach could be reduced by employing an alternative difference scheme and/or by only perturbing the components of z which appear in the definitions of the f_i. A discussion on the choice of perturbation factors for use in finite difference approximations to partial derivatives can be found in Gill *et al.* (1981).

10 Example

The following example is used to illustrate the treatment of nonseparated boundary conditions. See also D02TLF, D02TXF, D02TYF and D02TZF, for the illustration of other facilities.

The following equations model of the spread of measles. See Schwartz (1983). Under certain assumptions the dynamics of the model can be expressed as

$$\begin{aligned} y_1' &= \mu - \beta(x) y_1 y_3 \\ y_2' &= \beta(x) y_1 y_3 - y_2/\lambda \\ y_3' &= y_2/\lambda - y_3/\eta \end{aligned}$$

subject to the periodic boundary conditions

$$y_i(0) = y_i(1), \quad i = 1, 2, 3.$$

Here y_1, y_2 and y_3 are respectively the proportions of susceptibles, infectives and latents to the whole population. λ ($= 0.0279$ years) is the latent period, η ($= 0.01$ years) is the infectious period and μ ($= 0.02$) is the population birth rate. $\beta(x) = \beta_0(1.0 + \cos 2\pi x)$ is the contact rate where $\beta_0 = 1575.0$.

The nonseparated boundary conditions are treated as described in Section 9 by adjoining the trivial differential equations

$$\begin{aligned} y_4' &= 0 \\ y_5' &= 0 \\ y_6' &= 0 \end{aligned}$$

that is y_4, y_5 and y_6 are constants. The boundary conditions of the augmented system can then be posed in the separated form

$$\begin{aligned} y_1(0) - y_4(0) &= 0 \\ y_2(0) - y_5(0) &= 0 \\ y_3(0) - y_6(0) &= 0 \\ y_1(1) - y_4(1) &= 0 \\ y_2(1) - y_5(1) &= 0 \\ y_3(1) - y_6(1) &= 0. \end{aligned}$$

This is a relatively easy problem and an (arbitrary) initial guess of 1 for each component suffices, even though two components of the solution are much smaller than 1.

10.1 Program Text

```
!   D02TVF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module d02tvfe_mod

!     D02TVF 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, Public :: one = 1.0_nag_wp
      Real (Kind=nag_wp), Parameter    :: two = 2.0_nag_wp
      Real (Kind=nag_wp), Parameter    :: zero = 0.0_nag_wp
      Integer, Parameter, Public       :: mmax = 1, neq = 6, nin = 5,          &
                                            nlbc = 3, nout = 6, nrbc = 3
!     .. Local Scalars ..
      Real (Kind=nag_wp), Public, Save :: beta0, eta, lambda, mu
!     .. Local Arrays ..
      Integer, Public, Save            :: m(neq) = (/1,1,1,1,1,1/)
    Contains
      Subroutine ffun(x,y,neq,m,f,iuser,ruser)

!       .. Use Statements ..
        Use nag_library, Only: x01aaf
!       .. 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)              :: beta
!       .. Intrinsic Procedures ..
        Intrinsic                       :: cos
!       .. Executable Statements ..
        beta = beta0*(one+cos(two*x01aaf(beta)*x))
        f(1) = mu - beta*y(1,0)*y(3,0)
        f(2) = beta*y(1,0)*y(3,0) - y(2,0)/lambda
        f(3) = y(2,0)/lambda - y(3,0)/eta
        f(4:6) = zero
        Return
      End Subroutine ffun
      Subroutine fjac(x,y,neq,m,dfdy,iuser,ruser)

!       .. Use Statements ..
        Use nag_library, Only: x01aaf
!       .. 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)
!       .. Local Scalars ..
        Real (Kind=nag_wp)              :: beta
!       .. Intrinsic Procedures ..
        Intrinsic                       :: cos
!       .. Executable Statements ..
```

```
        beta = beta0*(one+cos(two*x01aaf(beta)*x))
        dfdy(1,1,0) = -beta*y(3,0)
        dfdy(1,3,0) = -beta*y(1,0)
        dfdy(2,1,0) = beta*y(3,0)
        dfdy(2,2,0) = -one/lambda
        dfdy(2,3,0) = beta*y(1,0)
        dfdy(3,2,0) = one/lambda
        dfdy(3,3,0) = -one/eta
        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) - ya(4,0)
        ga(2) = ya(2,0) - ya(5,0)
        ga(3) = ya(3,0) - ya(6,0)
        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(1,0) - yb(4,0)
        gb(2) = yb(2,0) - yb(5,0)
        gb(3) = yb(3,0) - yb(6,0)
        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(1,4,0) = -one
        dgady(2,2,0) = one
        dgady(2,5,0) = -one
        dgady(3,3,0) = one
        dgady(3,6,0) = -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(1,4,0) = -one
        dgbdy(2,2,0) = one
        dgbdy(2,5,0) = -one
        dgbdy(3,3,0) = one
        dgbdy(3,6,0) = -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(1:3,0) = one
        y(4,0) = y(1,0)
        y(5,0) = y(2,0)
        y(6,0) = y(3,0)
        dym(1:neq) = zero
        Return
      End Subroutine guess
    End Module d02tvfe_mod
    Program d02tvfe

!     D02TVF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02tlf, d02tvf, d02tyf, d02tzf, nag_wp
      Use d02tvfe_mod, Only: beta0, eta, ffun, fjac, gafun, gajac, gbfun,      &
                             gbjac, guess, lambda, m, mmax, mu, neq, nin,      &
                             nlbc, nout, nrbc, one
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: dx, ermx
      Integer                          :: i, iermx, ifail, ijermx, licomm,     &
                                          lrcomm, mxmesh, ncol, nmesh
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: mesh(:), rcomm(:), tols(:), y(:,:)
      Real (Kind=nag_wp)               :: ruser(1)
      Integer, Allocatable             :: icomm(:), ipmesh(:)
      Integer                          :: iuser(2)
!     .. Intrinsic Procedures ..
      Intrinsic                        :: real
!     .. Executable Statements ..
      Write (nout,*) 'D02TVF Example Program Results'
      Write (nout,*)
!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) ncol, nmesh, mxmesh
      Allocate (mesh(mxmesh),tols(neq),y(neq,0:mmax-1),ipmesh(mxmesh))

      Read (nin,*) beta0, eta, lambda, mu
      Read (nin,*) tols(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,tols,mxmesh,nmesh,mesh,ipmesh,ruser,0,  &
        iuser,2,ifail)
      lrcomm = iuser(1)
      licomm = iuser(2)
      Allocate (rcomm(lrcomm),icomm(licomm))

!     Initialize
      ifail = 0
      Call d02tvf(neq,m,nlbc,nrbc,ncol,tols,mxmesh,nmesh,mesh,ipmesh,rcomm,    &
        lrcomm,icomm,licomm,ifail)

!     Solve
      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
!       Print mesh statistics
        Write (nout,99999) nmesh, ermx, iermx, ijermx
        Write (nout,99998)(i,ipmesh(i),mesh(i),i=1,nmesh)

!       Print solution on mesh.
        Write (nout,99997)
        Do i = 1, nmesh

          ifail = 0
          Call d02tyf(mesh(i),y,neq,mmax,rcomm,icomm,ifail)
          Write (nout,99996) mesh(i), y(1:3,0)

        End Do
      End If

99999 Format (/,' Used a mesh of ',I4,' points',/,' Maximum error = ',E10.2,   &
        '  in interval ',I4,' for component ',I4,/)
99998 Format (/,' Mesh points:',/,4(I4,'(',I1,')',F7.4))
99997 Format (/,' Computed solution at mesh points',/,'    x       y1   ',     &
        '      y2         y3')
99996 Format (1X,F6.3,1X,3E11.3)
    End Program d02tvfe
```

10.2 Program Data

```
D02TVF Example Program Data
 5  11  100                  : ncol, nmesh, mxmesh
 1575.0 0.01 0.0279 0.02     : beta0, eta, lambda, mu
 1.0E-5 1.0E-5 1.0E-5
 1.0E-5 1.0E-5 1.0E-5        : tols(1:neq)
```

10.3 Program Results

```
 D02TVF Example Program Results


 Used a mesh of   21 points
 Maximum error =   0.14E-07  in interval    5 for component    1


 Mesh points:
   1(1) 0.0000   2(3) 0.0500   3(2) 0.1000   4(3) 0.1500
   5(2) 0.2000   6(3) 0.2500   7(2) 0.3000   8(3) 0.3500
   9(2) 0.4000  10(3) 0.4500  11(2) 0.5000  12(3) 0.5500
  13(2) 0.6000  14(3) 0.6500  15(2) 0.7000  16(3) 0.7500
  17(2) 0.8000  18(3) 0.8500  19(2) 0.9000  20(3) 0.9500
  21(1) 1.0000
```

```
Computed solution at mesh points
   x       y1          y2          y3
 0.000   0.752E-01  0.180E-04  0.498E-05
 0.050   0.761E-01  0.789E-04  0.219E-04
 0.100   0.766E-01  0.315E-03  0.892E-04
 0.150   0.758E-01  0.101E-02  0.298E-03
 0.200   0.726E-01  0.225E-02  0.713E-03
 0.250   0.678E-01  0.311E-02  0.108E-02
 0.300   0.641E-01  0.256E-02  0.984E-03
 0.350   0.629E-01  0.129E-02  0.550E-03
 0.400   0.633E-01  0.414E-03  0.197E-03
 0.450   0.643E-01  0.912E-04  0.478E-04
 0.500   0.653E-01  0.159E-04  0.881E-05
 0.550   0.663E-01  0.277E-05  0.151E-05
 0.600   0.673E-01  0.628E-06  0.313E-06
 0.650   0.683E-01  0.219E-06  0.964E-07
 0.700   0.693E-01  0.124E-06  0.487E-07
 0.750   0.703E-01  0.116E-06  0.409E-07
 0.800   0.713E-01  0.170E-06  0.551E-07
 0.850   0.723E-01  0.370E-06  0.113E-06
 0.900   0.733E-01  0.111E-05  0.322E-06
 0.950   0.743E-01  0.420E-05  0.118E-05
 1.000   0.752E-01  0.180E-04  0.498E-05
```