NAG Library Routine Document

D02TYF

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

D02TYF interpolates on the solution of a general two-point boundary value problem computed by D02TKF.

2 Specification

```
SUBROUTINE D02TYF (X, Y, NEQ, MMAX, RWORK, IWORK, IFAIL)

INTEGER            NEQ, MMAX, IWORK(*), IFAIL
REAL (KIND=nag_wp) X, Y(NEQ,0:MMAX-1), RWORK(*)
```

3 Description

D02TYF and its associated routines (D02TKF, D02TVF, D02TXF and D02TZF) solve the two-point boundary value problem for a nonlinear mixed order system of ordinary differential equations

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

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

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

and the right boundary conditions at b as

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

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, D02TKF 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 $[a, b]$ using interpolation.

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

Grossman C (1992) Enclosures of the solution of the Thomas–Fermi equation by monotone discretization *J. Comput. Phys.* **98** 26–32

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

5 Parameters

1: X – REAL (KIND=nag_wp) *Input*

On entry: x, the independent variable.

Constraint: $a \leq \mathrm{X} \leq b$.

2: Y(NEQ,0 : MMAX − 1) – REAL (KIND=nag_wp) array *Output*

On exit: $\mathrm{Y}(i,j)$ contains an approximation to $y_i^{(j)}(x)$, for $i = 1,2,\ldots,\mathrm{NEQ}$ and $j = 0,1,\ldots,m_i - 1$. The remaining elements of Y (where $m_i < \mathrm{MMAX}$) are initialized to 0.0.

3: NEQ – INTEGER *Input*

On entry: the number of differential equations.

Constraint: NEQ must be the same value as supplied to D02TVF.

4: MMAX – INTEGER *Input*

On entry: the maximal order of the differential equations, $\max(m_i)$, for $i = 1,2,\ldots,\mathrm{NEQ}$.

Constraint: MMAX must contain the maximum value of the components of the parameter M as supplied to D02TVF.

5: RWORK(∗) – REAL (KIND=nag_wp) array *Communication Array*

Note: the dimension of the array RWORK must be at least LRWORK (see D02TVF).

On entry: this must be the same array as supplied to D02TKF and **must** remain unchanged between calls.

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

6: IWORK(∗) – INTEGER array *Communication Array*

Note: the dimension of the array IWORK must be at least LIWORK (see D02TVF).

On entry: this must be the same array as supplied to D02TKF 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 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: D02TYF may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the routine:

IFAIL $= 1$

On entry, an invalid value for NEQ, MMAX ($\neq \max(m_i)$ for some i) or X (outside the range $[a, b]$) was detected, or an invalid call to D02TYF was made, for example without a previous call to the solver routine D02TKF.

IFAIL $= 2$

The solver routine D02TKF did not converge to a solution or did not satisfy the error requirements. The last solution computed by D02TKF, for which convergence was obtained, has been used for interpolation by D02TYF. The results returned by D02TYF should be treated with extreme caution as regarding either their quality or accuracy. See Section 8.

7 Accuracy

If D02TYF returns the value IFAIL $= 0$, the computed values of the solution components y_i should be of similar accuracy to that specified by the parameter TOLS of D02TVF. Note that during the solution process the error in the derivatives $y_i^{(j)}$, for $j = 1, 2, \ldots, m_i - 1$, has not been controlled and that the derivative values returned by D02TYF are computed via differentiation of the piecewise polynomial approximation to y_i. See also Section 8.

8 Further Comments

If D02TYF returns the value IFAIL $= 2$, and the solver routine D02TKF returned IFAIL $= 5$, then the accuracy of the interpolated values may be proportional to the quantity ERMX as returned by D02TZF.

If D02TKF returned any other nonzero value for IFAIL, then nothing can be said regarding either the quality or accuracy of the values computed by D02TYF.

9 Example

The following example is used to illustrate that a system with singular coefficients can be treated without modification of the system definition. See also D02TKF, D02TVF, D02TXF and D02TZF, for the illustration of other facilities.

Consider the Thomas–Fermi equation used in the investigation of potentials and charge densities of ionized atoms. See Grossman (1992), for example, and the references therein. The equation is

$$y'' = x^{-1/2} y^{3/2}$$

with boundary conditions

$$y(0) = 1, \qquad y(a) = 0, \qquad a > 0.$$

The coefficient $x^{-1/2}$ implies a singularity at the left-hand boundary $x = 0$.

We use the initial approximation $y(x) = 1 - x/a$, which satisfies the boundary conditions, on a uniform mesh of six points. For illustration we choose $a = 1$, as in Grossman (1992). Note that in FFUN and FJAC (see D02TKF) we have taken the precaution of setting the function value and Jacobian value to 0.0 in case a value of y becomes negative, although starting from our initial solution profile this proves unnecessary during the solution phase. Of course the true solution $y(x)$ is positive for all $x < a$.

9.1 Program Text

```
!   D02TYF Example Program Text
!   Mark 24 Release. NAG Copyright 2012.

    Module d02tyfe_mod

!     D02TYF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter                   :: mmax = 2, neq = 1, nin = 5,      &
                                              nlbc = 1, nmesh_out = 11,        &
                                              nout = 6, nrbc = 1
!     .. Local Scalars ..
      Real (Kind=nag_wp)                   :: a
!     .. Local Arrays ..
      Integer                              :: m(1) = (/2/)
    Contains
      Subroutine ffun(x,y,neq,m,f)

!       .. 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 (In)      :: y(neq,0:*)
        Integer, Intent (In)                 :: m(neq)
!       .. Intrinsic Procedures ..
        Intrinsic                            :: sqrt
!       .. Executable Statements ..
        If (y(1,0)<=0.0E0_nag_wp) Then
          f(1) = 0.0_nag_wp
        Else
          f(1) = (y(1,0))**1.5_nag_wp/sqrt(x)
        End If
        Return
      End Subroutine ffun
      Subroutine fjac(x,y,neq,m,dfdy)

!       .. 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:*)
        Real (Kind=nag_wp), Intent (In)      :: y(neq,0:*)
        Integer, Intent (In)                 :: m(neq)
!       .. Intrinsic Procedures ..
        Intrinsic                            :: sqrt
!       .. Executable Statements ..
        If (y(1,0)<=0.0E0_nag_wp) Then
          dfdy(1,1,0) = 0.0_nag_wp
        Else
          dfdy(1,1,0) = 1.5_nag_wp*sqrt(y(1,0))/sqrt(x)
        End If
        Return
      End Subroutine fjac
      Subroutine gafun(ya,neq,m,nlbc,ga)
```

```
!       .. Scalar Arguments ..
        Integer, Intent (In)                 :: neq, nlbc
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: ga(nlbc)
        Real (Kind=nag_wp), Intent (In)      :: ya(neq,0:*)
        Integer, Intent (In)                 :: m(neq)
!       .. Executable Statements ..
        ga(1) = ya(1,0) - 1.0_nag_wp
        Return
      End Subroutine gafun
      Subroutine gbfun(yb,neq,m,nrbc,gb)

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

!       .. Scalar Arguments ..
        Integer, Intent (In)                 :: neq, nlbc
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout)   :: dgady(nlbc,neq,0:*)
        Real (Kind=nag_wp), Intent (In)      :: ya(neq,0:*)
        Integer, Intent (In)                 :: m(neq)
!       .. Executable Statements ..
        dgady(1,1,0) = 1.0_nag_wp
        Return
      End Subroutine gajac
      Subroutine gbjac(yb,neq,m,nrbc,dgbdy)

!       .. Scalar Arguments ..
        Integer, Intent (In)                 :: neq, nrbc
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout)   :: dgbdy(nrbc,neq,0:*)
        Real (Kind=nag_wp), Intent (In)      :: yb(neq,0:*)
        Integer, Intent (In)                 :: m(neq)
!       .. Executable Statements ..
        dgbdy(1,1,0) = 1.0_nag_wp
        Return
      End Subroutine gbjac
      Subroutine guess(x,neq,m,y,dym)

!       .. 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)   :: y(neq,0:*)
        Integer, Intent (In)                 :: m(neq)
!       .. Executable Statements ..
        y(1,0) = 1.0_nag_wp - x/a
        y(1,1) = -1.0_nag_wp/a
        dym(1) = 0.0_nag_wp
        Return
      End Subroutine guess
    End Module d02tyfe_mod
    Program d02tyfe

!     D02TYF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02tkf, d02tvf, d02tyf, d02tzf, nag_wp
      Use d02tyfe_mod, Only: a, ffun, fjac, gafun, gajac, gbfun, gbjac, guess, &
                             m, mmax, neq, nin, nlbc, nmesh_out, nout, nrbc
!     .. Implicit None Statement ..
```

```
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                    :: ainc, ermx, x
      Integer                               :: i, iermx, ifail, ijermx, liwork, &
                                               lrwork, mxmesh, ncol, nmesh
      Logical                               :: failed
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable       :: mesh(:), rwork(:), tol(:), y(:,:)
      Integer, Allocatable                  :: ipmesh(:), iwork(:)
!     .. Intrinsic Procedures ..
      Intrinsic                             :: real
!     .. Executable Statements ..
      Write (nout,*) 'D02TYF Example Program Results'
      Write (nout,*)
!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) ncol, nmesh, mxmesh
      liwork = mxmesh*(11*neq+6)
      lrwork = mxmesh*(109*neq**2+78*neq+7)
      Allocate (mesh(mxmesh),tol(neq),rwork(lrwork),y(neq,0:mmax-1), &
        ipmesh(mxmesh),iwork(liwork))

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

      ainc = a/real(nmesh-1,kind=nag_wp)
      mesh(1) = 0.0E0_nag_wp
      Do i = 2, nmesh - 1
        mesh(i) = mesh(i-1) + ainc
      End Do
      mesh(nmesh) = a

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

!     Initialize
      ifail = 0
      Call d02tvf(neq,m,nlbc,nrbc,ncol,tol,mxmesh,nmesh,mesh,ipmesh,rwork, &
        lrwork,iwork,liwork,ifail)

      Write (nout,99999) tol(1), a

!     Solve
      ifail = -1
      Call d02tkf(ffun,fjac,gafun,gbfun,gajac,gbjac,guess,rwork,iwork,ifail)

      failed = ifail /= 0

!     Extract mesh.
      ifail = -1
      Call d02tzf(mxmesh,nmesh,mesh,ipmesh,ermx,iermx,ijermx,rwork,iwork, &
        ifail)

      If (ifail/=1) Then
!       Print mesh statistics
        Write (nout,99998) nmesh, ermx, iermx, ijermx
        Write (nout,99997)(i,ipmesh(i),mesh(i),i=1,nmesh)
      End If
      If (.Not. failed) Then
!       Print solution on output mesh.
        Write (nout,99996)
        x = 0.0_nag_wp
        ainc = a/real(nmesh_out-1,kind=nag_wp)
        Do i = 1, nmesh_out
          ifail = 0
          Call d02tyf(x,y,neq,mmax,rwork,iwork,ifail)
          Write (nout,99995) x, y(1,0:1)
          x = x + ainc
        End Do
      End If
```

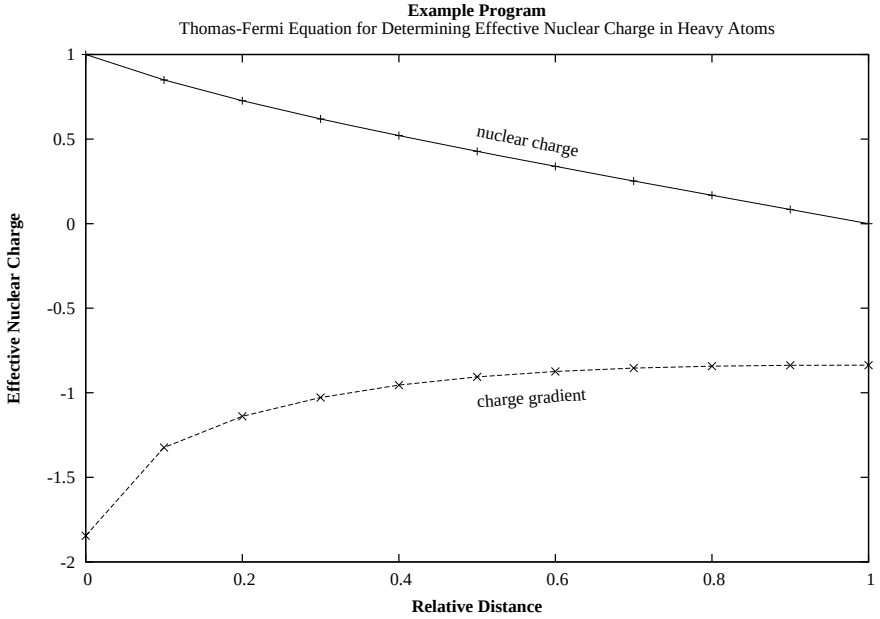
```
99999 Format (//' Tolerance = ',E8.1,' A = ',F8.2)
99998 Format (/' Used a mesh of ',I4,' points'/' Maximum error = ',E10.2, &
        '  in interval ',I4,' for component ',I4/)
99997 Format (/' Mesh points:'/4(I4,'(',I1,')',E11.4))
99996 Format (/' Computed solution'/'       x     solution   derivative')
99995 Format (' ',F8.2,2F11.5)

    End Program d02tyfe
```

9.2 Program Data

```
D02TYF Example Program Data
   4  6  100                  : ncol, nmesh, mxmesh
   1.0                        : a
   1.0E-5                     : tol
```

9.3 Program Results

```
 D02TYF Example Program Results



 Tolerance =  0.1E-04 A =     1.00

 Used a mesh of   11 points
 Maximum error =   0.31E-05  in interval    1 for component    1


 Mesh points:
   1(1) 0.0000E+00   2(3) 0.1000E+00   3(2) 0.2000E+00   4(3) 0.3000E+00
   5(2) 0.4000E+00   6(3) 0.5000E+00   7(2) 0.6000E+00   8(3) 0.7000E+00
   9(2) 0.8000E+00  10(3) 0.9000E+00  11(1) 0.1000E+01

 Computed solution
       x     solution   derivative
     0.00    1.00000   -1.84496
     0.10    0.84944   -1.32330
     0.20    0.72721   -1.13911
     0.30    0.61927   -1.02776
     0.40    0.52040   -0.95468
     0.50    0.42754   -0.90583
     0.60    0.33867   -0.87372
     0.70    0.25239   -0.85369
     0.80    0.16764   -0.84248
     0.90    0.08368   -0.83756
     1.00    0.00000   -0.83655
```


Example Program
Thomas-Fermi Equation for Determining Effective Nuclear Charge in Heavy Atoms
Effective Nuclear Charge
Relative Distance
nuclear charge
charge gradient
1
0.5
0
-0.5
-1
-1.5
-2
0
0.2
0.4
0.6
0.8
1