NAG Library Routine Document

D01TDF

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

D01TDF computes the weights and abscissae of a Gaussian quadrature rule using the method of Golub and Welsch.

2 Specification

```
SUBROUTINE D01TDF (N, A, B, C, MUZERO, WEIGHT, ABSCIS, IFAIL)
INTEGER                 N, IFAIL
REAL (KIND=nag_wp) A(N), B(N), C(N), MUZERO, WEIGHT(N), ABSCIS(N)
```

3 Description

A tri-diagonal system of equations is formed from the coefficients of an underlying three-term recurrence formula:

$$p(j)(x) = (a(j)x + b(j))p(j-1)(x) - c(j)p(j-2)(x)$$

for a set of othogonal polynomials $p(j)$ induced by the quadrature. This is described in greater detail in the D01 Chapter Introduction. The user is required to specify the three-term recurrence and the value of the integral of the chosen weight function over the chosen interval.

As described in Golub and Welsch (1969) the abscissae are computed from the eigenvalues of this matrix and the weights from the first component of the eigenvectors.

LAPACK routines are used for the linear algebra to speed up computation.

4 References

Golub G H and Welsch J H (1969) Calculation of Gauss quadrature rules *Math. Comput.* **23** 221–230

5 Arguments

1: N – INTEGER *Input*

On entry: n, the number of Gauss points required. The resulting quadrature rule will be exact for all polynomials of degree $2n - 1$.

Constraint: N > 0.

2: A(N) – REAL (KIND=nag_wp) array *Input*

On entry: A contains the coefficients $a(j)$.

3: B(N) – REAL (KIND=nag_wp) array *Input/Output*

On entry: B contains the coefficients $b(j)$.

On exit: elements of B are altered to make the underlying eigenvalue problem symmetric.

4: C(N) – REAL (KIND=nag_wp) array *Input/Output*

On entry: C contains the coefficients $c(j)$.

On exit: elements of C are altered to make the underlying eigenvalue problem symmetric.

5: MUZERO – REAL (KIND=nag_wp) *Input*

On entry: MUZERO contains the definite integral of the weight function for the interval of interest.

6: WEIGHT(N) – REAL (KIND=nag_wp) array *Output*

On exit: WEIGHT(j) contains the weight corresponding to the jth abscissa.

7: ABSCIS(N) – REAL (KIND=nag_wp) array *Output*

On exit: ABSCIS(j) the jth abscissa.

8: 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 $= 1$

The number of weights and abscissae requested (N) is less than 1: N $= \langle value \rangle$.

IFAIL $= 4$

Unexpected failure in eigenvalue computation. Please contact NAG.

IFAIL $= 5$

The algorithm failed to converge. The ith diagonal was not zero: $i = \langle value \rangle$.

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

In general the computed weights and abscissae are accurate to a reasonable multiple of ***machine precision***.

8 Parallelism and Performance

D01TDF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

D01TDF 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

The weight function must be non-negative to obtain sensible results. This and the validity of MUZERO are not something that the routine can check, so please be particularly careful. If possible check the computed weights and abscissae by integrating a function with a function for which you already know the integral.

10 Example

This example program generates the weights and abscissae for the 4-point Gauss rules: Legendre, Chebyshev1, Chebyshev2, Jacobi, Laguerre and Hermite.

10.1 Program Text

```
!   D01TDF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.
    Module d01tdfe_mod

!     D01TDF 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                             :: basic_types
!     .. Parameters ..
      Integer, Parameter, Public         :: chebyshev1 = 2, chebyshev2 = 3,      &
                                            hermite = 6, jacobi = 4,             &
                                            laguerre = 5, legendre = 1
    Contains
      Subroutine basic_types(rulekind,alpha,beta,n,a,b,c,muzero)
!       This procedure supplies the coefficients of the three term
!       recurrence relationship for various classical orthogonal
!       polynomials.

!       .. Use Statements ..
        Use nag_library, Only: s14aaf, x01aaf
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: alpha, beta
        Real (Kind=nag_wp), Intent (Out) :: muzero
        Integer, Intent (In)            :: n, rulekind
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: a(1:n), b(1:n), c(1:n)
!       .. Local Scalars ..
```

```
        Real (Kind=nag_wp)             :: abl, mypi
        Integer                        :: i, ifail
!       .. Intrinsic Procedures ..
        Intrinsic                      :: real, sqrt
!       .. Executable Statements ..
        mypi = x01aaf(mypi)

        Select Case (rulekind)

        Case (legendre)
          muzero = 2.0_nag_wp
!         P(x) in [-1, 1), w(x) = 1.0
          Do i = 1, n
            a(i) = (real(2*i-1,kind=nag_wp))/real(i,kind=nag_wp)
            b(i) = 0.0_nag_wp
            c(i) = real(i-1,kind=nag_wp)/real(i,kind=nag_wp)
          End Do

        Case (chebyshev1)

          muzero = mypi
!         c(i)= (i-1)/i
!         T (x) in [-1,1], w(x) = (1-x**2)**(-0.5)
          Do i = 1, n
            a(i) = 2.0_nag_wp
            b(i) = 0.0_nag_wp
            c(i) = 1.0_nag_wp
          End Do
          If (n>0) Then
            a(1) = 1.0_nag_wp
          End If

        Case (chebyshev2)

          muzero = mypi/2.0_nag_wp
!         u(x) in [-1, 11, W(x) = (1-x**2)** 0.5;
          Do i = 1, n
            a(i) = 2.0_nag_wp
            b(i) = 0.0_nag_wp
            c(i) = 1.0_nag_wp
          End Do

        Case (jacobi)
          ifail = 0
          muzero = 2**(alpha+beta+1)*s14aaf(alpha+1,ifail)*                    &
            s14aaf(beta+1,ifail)/s14aaf(alpha+beta+2,ifail)
!         P(alpha,beta)(x) in [-1, 11, w(x) = (1-x)**alpha*(l+x)**beta
!         alpha> -1 and beta > -1
          If (n>0) Then
            a(1) = 0.5_nag_wp*(alpha+beta+2)
            b(1) = 0.5_nag_wp*(alpha-beta)
            c(1) = 0.0_nag_wp
          End If
          Do i = 2, n
            abl = 2.0_nag_wp*real(i,kind=nag_wp)*(real(i,kind=nag_wp)+alpha+   &
              beta)
            a(i) = (2.0_nag_wp*real(i,kind=nag_wp)+alpha+beta-1.0_nag_wp)*     &
              (2.0_nag_wp*real(i,kind=nag_wp)+alpha+beta)/abl
            abl = (2.0_nag_wp*real(i,kind=nag_wp)+alpha+beta-2.0_nag_wp)*abl
            b(i) = (2.0_nag_wp*real(i,kind=nag_wp)+alpha+beta-1.0_nag_wp)*     &
              (alpha**2-beta**2)/abl
            c(i) = 2.0_nag_wp*(real(i,kind=nag_wp)-1.0_nag_wp+alpha)*          &
              (real(i,kind=nag_wp)-1.0_nag_wp+beta)*                           &
              (2.0_nag_wp*real(i,kind=nag_wp)+alpha+beta)/abl
          End Do

        Case (laguerre)
          ifail = 0
          muzero = s14aaf(alpha+1.0_nag_wp,ifail)
!         L(alpha)(x) in [0, infinity), w(x) = exp(-x)*x**alpha,
!         alpha > -1
```

```
          Do i = 1, n
            a(i) = -1.0_nag_wp/real(i,kind=nag_wp)
            b(i) = (2.0_nag_wp*real(i,kind=nag_wp)-1.0_nag_wp+alpha)/          &
              real(i,kind=nag_wp)
            c(i) = (real(i,kind=nag_wp)-1.0_nag_wp+alpha)/real(i,kind=nag_wp)
          End Do

        Case (hermite)
          muzero = sqrt(mypi)
!         H(x) in (-infinity,+infinity), w(x) = exp(-x**2)
          Do i = 1, n
            a(i) = 2.0_nag_wp
            b(i) = 0.0_nag_wp
            c(i) = 2.0_nag_wp*real(i-1,kind=nag_wp)
          End Do

        End Select
      End Subroutine basic_types
    End Module d01tdfe_mod

    Program d01tdfe
!     D01TDF Example Program Text

!     .. Use Statements ..
      Use nag_library, Only: d01tdf, nag_wp
      Use d01tdfe_mod, Only: basic_types, chebyshev1, chebyshev2, hermite,     &
                             jacobi, laguerre, legendre
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter               :: n = 4, nout = 6
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: alpha, beta, muzero
      Integer                          :: ifail, j, rule
!     .. Local Arrays ..
      Real (Kind=nag_wp)               :: a(1:n), b(1:n), c(1:n), t(1:n),      &
                                          w(1:n)
!     .. Executable Statements ..

      Write (nout,*) 'D01TDF Example Program Results '
      Write (6,*)
      rule = legendre
      Do rule = 1, 6
        ifail = -1

        Select Case (rule)

        Case (legendre)
          Call basic_types(rule,alpha,beta,n,a,b,c,muzero)
          Write (nout,*) 'Using the Gauss-Legendre Rule'
          Call d01tdf(n,a,b,c,muzero,w,t,ifail)
          If (ifail==0) Then
            Write (nout,99998)
            Write (nout,99997)(j,t(j),w(j),j=1,n)
            Write (6,*)
          End If

        Case (chebyshev1)
          Call basic_types(rule,alpha,beta,n,a,b,c,muzero)
          Write (nout,*) 'Using the Chebyshev Rule 1'
          Call d01tdf(n,a,b,c,muzero,w,t,ifail)
          If (ifail==0) Then
            Write (nout,99998)
            Write (nout,99997)(j,t(j),w(j),j=1,n)
            Write (6,*)
          End If

        Case (chebyshev2)
          Call basic_types(rule,alpha,beta,n,a,b,c,muzero)
          Write (nout,*) 'Using the Chebyshev Rule 2'
          Call d01tdf(n,a,b,c,muzero,w,t,ifail)
```

```
          If (ifail==0) Then
            Write (nout,99998)
            Write (nout,99997)(j,t(j),w(j),j=1,n)
            Write (6,*)
          End If

        Case (jacobi)
          alpha = 0.5_nag_wp
          beta = 0.5_nag_wp
          Call basic_types(rule,alpha,beta,n,a,b,c,muzero)
          Write (nout,99999) alpha, beta
          Call d01tdf(n,a,b,c,muzero,w,t,ifail)
          If (ifail==0) Then
            Write (nout,99998)
            Write (nout,99997)(j,t(j),w(j),j=1,n)
            Write (6,*)
          End If

        Case (laguerre)
          Call basic_types(rule,alpha,beta,n,a,b,c,muzero)
          Write (nout,*) 'Using the Laguerre Rule'
          Call d01tdf(n,a,b,c,muzero,w,t,ifail)
          If (ifail==0) Then
            Write (nout,99998)
            Write (nout,99997)(j,t(j),w(j),j=1,n)
            Write (6,*)
          End If

        Case (hermite)
          Call basic_types(rule,alpha,beta,n,a,b,c,muzero)
          Write (nout,*) 'Using the Hermite Rule'
          Call d01tdf(n,a,b,c,muzero,w,t,ifail)
          If (ifail==0) Then
            Write (nout,99998)
            Write (nout,99997)(j,t(j),w(j),j=1,n)
            Write (6,*)
          End If

        End Select

      End Do
99999 Format (' Using the Jacobi Rule: alpha = ',F10.5,'  beta = ',F10.5)

99998 Format (/,'   j    ','         Abscissa        ','          Weight')
99997 Format (I8,D25.15,D25.15)
    End Program d01tdfe
```

10.2 Program Data

None.

10.3 Program Results

```
 D01TDF Example Program Results

 Using the Gauss-Legendre Rule

   j             Abscissa                 Weight
       1   -0.861136311594053D+00    0.347854845137454D+00
       2   -0.339981043584856D+00    0.652145154862546D+00
       3    0.339981043584856D+00    0.652145154862546D+00
       4    0.861136311594052D+00    0.347854845137454D+00

 Using the Chebyshev Rule 1

   j             Abscissa                 Weight
       1   -0.923879532511287D+00    0.785398163397449D+00
       2   -0.382683432365090D+00    0.785398163397447D+00
       3    0.382683432365090D+00    0.785398163397449D+00
       4    0.923879532511287D+00    0.785398163397450D+00
```

```
 Using the Chebyshev Rule 2

   j            Abscissa                Weight
       1   -0.809016994374947D+00    0.217078713422706D+00
       2   -0.309016994374947D+00    0.568319449974742D+00
       3    0.309016994374948D+00    0.568319449974742D+00
       4    0.809016994374947D+00    0.217078713422706D+00

 Using the Jacobi Rule: alpha =    0.50000  beta =    0.50000

   j            Abscissa                Weight
       1   -0.809016994374947D+00    0.217078713422706D+00
       2   -0.309016994374947D+00    0.568319449974742D+00
       3    0.309016994374947D+00    0.568319449974742D+00
       4    0.809016994374947D+00    0.217078713422706D+00

 Using the Laguerre Rule

   j            Abscissa                Weight
       1    0.523526076738269D+00    0.453008746558608D+00
       2    0.215664876326909D+01    0.381616960171800D+00
       3    0.513738754617671D+01    0.507946275722408D-01
       4    0.101824376138159D+02    0.806591150110031D-03

 Using the Hermite Rule

   j            Abscissa                Weight
       1   -0.165068012388578D+01    0.813128354472451D-01
       2   -0.524647623275290D+00    0.804914090005513D+00
       3    0.524647623275290D+00    0.804914090005512D+00
       4    0.165068012388578D+01    0.813128354472453D-01
```