NAG Fortran Library Routine Document

D03PWF

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

D03PWF calculates a numerical flux function using a modified HLL (Harten-Lax-van Leer) Approximate Riemann Solver for the Euler equations in conservative form. The routine is designed primarily for use with the upwind discretisation routines D03PFF, D03PLF or D03PSF, but may also be applicable to other conservative upwind schemes requiring numerical flux functions.

2 Specification

```
SUBROUTINE D03PWF(ULEFT, URIGHT, GAMMA, FLUX, IFAIL)
INTEGER           IFAIL
real              ULEFT(3), URIGHT(3), GAMMA, FLUX(3)
```

3 Description

D03PWF calculates a numerical flux function at a single spatial point using a modified HLL (Harten-Lax-van Leer) Approximate Riemann Solver (see Toro (1992), Toro (1996) and Toro *et al.* (1994)) for the Euler equations (for a perfect gas) in conservative form. The user must supply the *left* and *right* solution values at the point where the numerical flux is required, i.e., the initial left and right states of the Riemann problem defined below. In the routines D03PFF, D03PLF and D03PSF, the left and right solution values are derived automatically from the solution values at adjacent spatial points and supplied to the subroutine argument NUMFLX from which the user may call D03PWF. The Euler equations for a perfect gas in conservative form are:

$$\frac{\partial U}{\partial t}+\frac{\partial F}{\partial x}=0, \tag{1}$$

with

$$U=\begin{bmatrix}\rho\\ m\\ e\end{bmatrix} \quad \text{and} \quad F=\begin{bmatrix}m\\ \frac{m^2}{\rho}+(\gamma-1)\left(e-\frac{m^2}{2\rho}\right)\\ \frac{me}{\rho}+\frac{m}{\rho}(\gamma-1)\left(e-\frac{m^2}{2\rho}\right)\end{bmatrix}, \tag{2}$$

where ρ is the density, m is the momentum, e is the specific total energy and γ is the (constant) ratio of specific heats. The pressure p is given by

$$p=(\gamma-1)\left(e-\frac{\rho u^2}{2}\right), \tag{3}$$

where $u=m/\rho$ is the velocity.

The routine calculates an approximation to the numerical flux function $F(U_L,U_R)=F(U^*(U_L,U_R))$, where $U=U_L$ and $U=U_R$ are the left and right solution values, and $U^*(U_L,U_R)$ is the intermediate state $\omega(0)$ arising from the similarity solution $U(y,t)=\omega(y/t)$ of the Riemann problem defined by

$$\frac{\partial U}{\partial t}+\frac{\partial F}{\partial y}=0, \tag{4}$$

with U and F as in (2), and initial piecewise constant values $U=U_L$ for $y<0$ and $U=U_R$ for $y>0$. The spatial domain is $-\infty<y<\infty$, where $y=0$ is the point at which the numerical flux is required.

4 References

Toro E F (1992) The weighted average flux method applied to the Euler equations *Phil. Trans. R. Soc. Lond.* **A341** 499–530

Toro E F (1996) *Riemann Solvers and Upwind Methods for Fluid Dynamics* Springer-Verlag

Toro E F, Spruce M and Spears W (1994) Restoration of the contact surface in the HLL Riemann solver *J. Shock Waves* **4** 25–34

5 Parameters

1: ULEFT(3) – ***real*** array *Input*

On entry: ULEFT(i) must contain the left value of the component U_i for $i = 1, 2, 3$. That is, ULEFT(1) must contain the left value of ρ, ULEFT(2) must contain the left value of m and ULEFT(3) must contain the left value of e.

2: URIGHT(3) – ***real*** array *Input*

On entry: URIGHT(i) must contain the right value of the component U_i for $i = 1, 2, 3$. That is, URIGHT(1) must contain the right value of ρ, URIGHT(2) must contain the right value of m and URIGHT(3) must contain the right value of e.

3: GAMMA – ***real*** *Input*

On entry: the ratio of specific heats γ.

Constraint: GAMMA > 0.0.

4: FLUX(3) – ***real*** array *Output*

On exit: FLUX(i) contains the numerical flux component $\hat{F}_i$ for $i = 1, 2, 3$.

5: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. Users who are unfamiliar with this parameter should refer to Chapter P01 for details.

On exit: IFAIL $= 0$ unless the routine detects an error (see Section 6).

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, for users not familiar with this parameter the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

Note: if the left and/or right values of ρ or p (from (3)) are found to be negative, then the routine will terminate with an error exit (IFAIL $= 2$). If the routine is being called from the user-supplied subroutine NUMFLX in D03PFF etc., then a **soft fail** option (IFAIL $= 1$ or -1) is recommended so that a recalculation of the current time step can be forced using the IRES parameter.

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, GAMMA ≤ 0.0.

IFAIL $= 2$

On entry, the left and/or right density or derived pressure value is less than 0.0.

7 Accuracy

The routine performs an exact calculation of the HLL numerical flux function, and so the result will be accurate to machine precision.

8 Further Comments

The routine must only be used to calculate the numerical flux for the Euler equations in exactly the form given by (2), with ULEFT(i) and URIGHT(i) containing the left and right values of ρ, m and e for $i = 1,2,3$ respectively. The time taken by the routine is independent of the input parameters.

9 Example

This example uses D03PLF and D03PWF to solve the Euler equations in the domain $0 \leq x \leq 1$ for $0 < t \leq 0.035$ with initial conditions for the primitive variables $\rho(x,t)$, $u(x,t)$ and $p(x,t)$ given by

$$\begin{array}{llllllll} \rho(x,0) = 5.99924, & u(x,0) & = & 19.5975, & p(x,0) & = & 460.894, & \text{for} \quad x < 0.5, \\ \rho(x,0) = 5.99242, & u(x,0) & = & -6.19633, & p(x,0) & = & 46.095, & \text{for} \quad x > 0.5. \end{array}$$

This test problem is taken from Toro (1996) and its solution represents the collision of two strong shocks travelling in opposite directions, consisting of a left facing shock (travelling slowly to the right), a right travelling contact discontinuity and a right travelling shock wave. There is an exact solution to this problem (see Toro (1996)) but the calculation is lengthy and has therefore been omitted.

9.1 Program Text

```
*     D03PWF Example Program Text
*     Mark 19 Revised. NAG Copyright 1999.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          NPDE, NPTS, NCODE, NXI, NEQN, NIW, NWKRES,
     +                 LENODE, MLU, NW
      PARAMETER        (NPDE=3,NPTS=141,NCODE=0,NXI=0,
     +                 NEQN=NPDE*NPTS+NCODE,NIW=NEQN+24,
     +                 NWKRES=NPDE*(2*NPTS+3*NPDE+32)+7*NPTS+4,
     +                 LENODE=9*NEQN+50,MLU=3*NPDE-1,NW=(3*MLU+1)
     +                 *NEQN+NWKRES+LENODE)
*     .. Scalars in Common ..
      real             EL0, ER0, GAMMA, RL0, RR0, UL0, UR0
*     .. Local Scalars ..
      real             D, P, TOUT, TS, V
      INTEGER          I, IFAIL, IND, ITASK, ITOL, ITRACE, K
      CHARACTER        LAOPT, NORM
*     .. Local Arrays ..
      real             ALGOPT(30), ATOL(1), RTOL(1), U(NPDE,NPTS),
     +                 UE(3,9), W(NW), X(NPTS), XI(1)
      INTEGER          IW(NIW)
*     .. External Subroutines ..
      EXTERNAL         BNDARY, D03PEK, D03PLF, D03PLP, NUMFLX
*     .. Common blocks ..
      COMMON           /INIT/EL0, ER0, RL0, RR0, UL0, UR0
      COMMON           /PARAMS/GAMMA
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03PWF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*
*     Problem parameters
*
      GAMMA = 1.4e0
      RL0 = 5.99924e0
      RR0 = 5.99242e0
      UL0 = 5.99924e0*19.5975e0
```

```
      UR0 = -5.99242e0*6.19633e0
      EL0 = 460.894e0/(GAMMA-1.0e0) + 0.5e0*RL0*19.5975e0**2
      ER0 = 46.095e0/(GAMMA-1.0e0) + 0.5e0*RR0*6.19633e0**2
*
*     Initialise mesh
*
      DO 20 I = 1, NPTS
         X(I) = 1.0e0*(I-1.0e0)/(NPTS-1.0e0)
   20 CONTINUE
*
*     Initial values
*
      DO 40 I = 1, NPTS
         IF (X(I).LT.0.5e0) THEN
            U(1,I) = RL0
            U(2,I) = UL0
            U(3,I) = EL0
         ELSE IF (X(I).EQ.0.5e0) THEN
            U(1,I) = 0.5e0*(RL0+RR0)
            U(2,I) = 0.5e0*(UL0+UR0)
            U(3,I) = 0.5e0*(EL0+ER0)
         ELSE
            U(1,I) = RR0
            U(2,I) = UR0
            U(3,I) = ER0
         END IF
   40 CONTINUE
*
      ITRACE = 0
      ITOL = 1
      NORM = '2'
      ATOL(1) = 0.5e-2
      RTOL(1) = 0.5e-3
      XI(1) = 0.0e0
      LAOPT = 'B'
      IND = 0
      ITASK = 1
      DO 60 I = 1, 30
         ALGOPT(I) = 0.0e0
   60 CONTINUE
*
*     Theta integration
*
      ALGOPT(1) = 2.0e0
      ALGOPT(6) = 2.0e0
      ALGOPT(7) = 2.0e0
*
*     Max. time step
*
      ALGOPT(13) = 0.5e-2
*
      TS = 0.0e0
      TOUT = 0.035e0
      IFAIL = 0
*
      CALL D03PLF(NPDE,TS,TOUT,D03PLP,NUMFLX,BNDARY,U,NPTS,X,NCODE,
     +            D03PEK,NXI,XI,NEQN,RTOL,ATOL,ITOL,NORM,LAOPT,ALGOPT,W,
     +            NW,IW,NIW,ITASK,ITRACE,IND,IFAIL)
*
      WRITE (NOUT,99998) TS
      WRITE (NOUT,99999)
*
*     Read exact data at output points
*
      DO 80 I = 1, 9
         READ (NIN,*) UE(1,I), UE(2,I), UE(3,I)
   80 CONTINUE
*
*     Calculate density, velocity and pressure
*
      K = 0
```

```
      DO 100 I = 15, NPTS - 14, 14
         D = U(1,I)
         V = U(2,I)/D
         P = D*(GAMMA-1.0e0)*(U(3,I)/D-0.5e0*V**2)
         K = K + 1
         WRITE (NOUT,99996) X(I), D, UE(1,K), V, UE(2,K), P, UE(3,K)
  100 CONTINUE
*
      WRITE (NOUT,99997) IW(1), IW(2), IW(3), IW(5)
      STOP
*
99999 FORMAT (4X,'X',6X,'APPROX D',3X,'EXACT D',4X,'APPROX V',3X,'EXAC',
     +       'T V',4X,'APPROX P',3X,'EXACT P')
99998 FORMAT (/' T = ',F6.3,/)
99997 FORMAT (/' Number of integration steps in time = ',I6,/' Number ',
     +       'of function evaluations = ',I6,/' Number of Jacobian ',
     +       'evaluations =',I6,/' Number of iterations = ',I6,/)
99996 FORMAT (1X,e8.2,6(1X,e10.4))
      END
*
      SUBROUTINE BNDARY(NPDE,NPTS,T,X,U,NCODE,V,VDOT,IBND,G,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IBND, IRES, NCODE, NPDE, NPTS
*     .. Array Arguments ..
      real              G(NPDE), U(NPDE,NPTS), V(*), VDOT(*), X(NPTS)
*     .. Scalars in Common ..
      real              EL0, ER0, RL0, RR0, UL0, UR0
*     .. Common blocks ..
      COMMON            /INIT/EL0, ER0, RL0, RR0, UL0, UR0
*     .. Executable Statements ..
      IF (IBND.EQ.0) THEN
         G(1) = U(1,1) - RL0
         G(2) = U(2,1) - UL0
         G(3) = U(3,1) - EL0
      ELSE
         G(1) = U(1,NPTS) - RR0
         G(2) = U(2,NPTS) - UR0
         G(3) = U(3,NPTS) - ER0
      END IF
      RETURN
      END
*
      SUBROUTINE NUMFLX(NPDE,T,X,NCODE,V,ULEFT,URIGHT,FLUX,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NCODE, NPDE
*     .. Array Arguments ..
      real              FLUX(NPDE), ULEFT(NPDE), URIGHT(NPDE), V(*)
*     .. Scalars in Common ..
      real              GAMMA
*     .. Local Scalars ..
      INTEGER           IFAIL
*     .. External Subroutines ..
      EXTERNAL          D03PWF
*     .. Common blocks ..
      COMMON            /PARAMS/GAMMA
*     .. Save statement ..
      SAVE              /PARAMS/
*     .. Executable Statements ..
*
      IFAIL = 0
      CALL D03PWF(ULEFT,URIGHT,GAMMA,FLUX,IFAIL)
      RETURN
      END
```

9.2 Program Data

```
D03PWF Example Program Data
 0.5999e+01   0.1960e+02   0.4609e+03
 0.5999e+01   0.1960e+02   0.4609e+03
 0.5999e+01   0.1960e+02   0.4609e+03
 0.5999e+01   0.1960e+02   0.4609e+03
 0.5999e+01   0.1960e+02   0.4609e+03
 0.1428e+02   0.8690e+01   0.1692e+04
 0.1428e+02   0.8690e+01   0.1692e+04
 0.1428e+02   0.8690e+01   0.1692e+04
 0.3104e+02   0.8690e+01   0.1692e+04
```

9.3 Program Results

```
 D03PWF Example Program Results

 T =  0.035

    X      APPROX D   EXACT D    APPROX V   EXACT V    APPROX P   EXACT P
 0.10E+00 0.5999E+01 0.5999E+01 0.1960E+02 0.1960E+02 0.4609E+03 0.4609E+03
 0.20E+00 0.5999E+01 0.5999E+01 0.1960E+02 0.1960E+02 0.4609E+03 0.4609E+03
 0.30E+00 0.5999E+01 0.5999E+01 0.1960E+02 0.1960E+02 0.4609E+03 0.4609E+03
 0.40E+00 0.5999E+01 0.5999E+01 0.1960E+02 0.1960E+02 0.4609E+03 0.4609E+03
 0.50E+00 0.5999E+01 0.5999E+01 0.1960E+02 0.1960E+02 0.4609E+03 0.4609E+03
 0.60E+00 0.1422E+02 0.1428E+02 0.8658E+01 0.8690E+01 0.1687E+04 0.1692E+04
 0.70E+00 0.1426E+02 0.1428E+02 0.8670E+01 0.8690E+01 0.1688E+04 0.1692E+04
 0.80E+00 0.1944E+02 0.1428E+02 0.8678E+01 0.8690E+01 0.1691E+04 0.1692E+04
 0.90E+00 0.3100E+02 0.3104E+02 0.8676E+01 0.8690E+01 0.1687E+04 0.1692E+04

 Number of integration steps in time =    699
 Number of function evaluations =   1714
 Number of Jacobian evaluations =     1
 Number of iterations =      2
```