NAG Toolbox for MATLAB

d03px

1 Purpose

d03px calculates a numerical flux function using an Exact Riemann Solver for the Euler equations in conservative form. It is designed primarily for use with the upwind discretization schemes d03pf, d03pl or d03ps, but may also be applicable to other conservative upwind schemes requiring numerical flux functions.

2 Syntax

```
[flux, ifail] = d03px(uleft, uright, gamma, tol, niter)
```

3 Description

d03px calculates a numerical flux function at a single spatial point using an Exact Riemann Solver (see Toro 1996 and Toro 1989) for the Euler equations (for a perfect gas) in conservative form. You 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 d03pf, d03pl and d03ps, the left and right solution values are derived automatically from the solution values at adjacent spatial points and supplied to the (sub)program argument user-supplied (sub)program **numflx** from which you may call d03px.

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 function calculates 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.

The algorithm is termed an Exact Riemann Solver although it does in fact calculate an approximate solution to a true Riemann problem, as opposed to an Approximate Riemann Solver which involves some form of alternative modelling of the Riemann problem. The approximation part of the Exact Riemann Solver is a Newton–Raphson iterative procedure to calculate the pressure, and you must supply a tolerance **tol** and a maximum number of iterations **niter**. Default values for these parameters can be chosen.

A solution cannot be found by this function if there is a vacuum state in the Riemann problem (loosely characterised by zero density), or if such a state is generated by the interaction of two non-vacuum data states. In this case a Riemann solver which can handle vacuum states has to be used (see Toro 1996).

4 References

Toro E F 1989 A weighted average flux method for hyperbolic conservation laws *Proc. Roy. Soc. Lond.* **A423** 401–418

Toro E F 1996 *Riemann Solvers and Upwind Methods for Fluid Dynamics* Springer–Verlag

5 Parameters

5.1 Compulsory Input Parameters

1: **uleft(3) – double array**

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) – double array**

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 – double scalar**

The ratio of specific heats, γ.

Constraint: **gamma** > 0.0.

4: **tol – double scalar**

The tolerance to be used in the Newton–Raphson procedure to calculate the pressure. If **tol** is set to zero then the default value of 1.0×10^{-6} is used.

Constraint: **tol** ≥ 0.0.

5: **niter – int32 scalar**

The maximum number of Newton–Raphson iterations allowed. If **niter** is set to zero then the default value of 20 is used.

Constraint: **niter** ≥ 0.

5.2 Optional Input Parameters

None.

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters

1: **flux(3) – double array**

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

2: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail = 1

On entry, **gamma** ≤ 0.0,
or **tol** < 0.0,
or **niter** < 0.

ifail = 2

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

ifail = 3

A vacuum condition has been detected therefore a solution cannot be found using this function. You are advised to check your problem formulation.

ifail = 4

The internal Newton–Raphson iterative procedure used to solve for the pressure has failed to converge. The value of **tol** or **niter** may be too small, but if the problem persists try an Approximate Riemann Solver (d03pu, d03pv or d03pw).

7 Accuracy

The algorithm is exact apart from the calculation of the pressure which uses a Newton–Raphson iterative procedure, the accuracy of which is controlled by the parameter **tol**. In some cases the initial guess for the Newton–Raphson procedure is exact and no further iterations are required.

8 Further Comments

d03px 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.

For some problems the function may fail or be highly inefficient in comparison with an Approximate Riemann Solver (e.g., d03pu, d03pv or d03pw). Hence it is advisable to try more than one Riemann solver and to compare the performance and the results.

The time taken by the function is independent of all input parameters other than **tol**.

9 Example

```
uleft = [5.99924;
     117.5701059;
     2304.275075187626];
uright = [5.99924;
     117.5701059;
     2304.275075187626];
gamma = 1.4;
tol = 0;
niter = int32(0);
[flux, ifail] = d03px(uleft, uright, gamma, tol, niter)
```

```
flux =
   1.0e+04 *
    0.0118
    0.2765
    5.4190
ifail =
           0
```