Development of High Resolution Schemes in OpenFOAM

B. Tech. Project Stage 2 Report

By

Potluri Vachan Deep

Roll # 140100101

Under the guidance of

Prof. Bhalchandra Puranik

Mechanical Engineering Department

Indian Institute of Technology Bombay

Dissertation Approval

It is hereby certified that this B.Tech. Project Stage 2 submission titled **Development of High Resolution Schemes in OpenFOAM** by Potluri Vachan Deep (Roll # 140100101) is approved for submission.

Date: 19th April 2018 Prof. Bhalchandra Puranik

Place: IIT Bombay

Declaration of Academic Integrity

I declare that this dissertation represents my ideas in my own words and where others' ideas and words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity. I understand that any violation will be cause for disciplinary action as per the rules and regulations of the Institute.

Date: 19th April 2018 Potluri Vachan Deep

Place: IIT Bombay (140100101)

Acknowledgment

I would like to express my sincere gratitude to my guide Prof. Bhalchandra Puranik for his invaluable support throughout the project. I am thankful to him for giving me more than required freedom and also for his apt and precise guidance during the course of this project. I would like to thank Abhimanyu for helping me in running parallel simulations. I am also grateful to Shardul for giving access to his PC for the purpose of running parallel simulations.

Date: 19th April 2018 Potluri Vachan Deep

Place: IIT Bombay (140100101)

Contents

Chapter 1

Introduction to Compressible Flows

Compressible flows occur in situations where change in density of a fluid is significant. Liquids generally become compressible only at large values of pressure which is not the case in most practical purposes. Gases, however, can become compressible and thus, in general, the study of compressible flows is referred to as Gas Dynamics in literature.

Most commonly encountered gas is air, whose flow can be treated either as incompressible (over a car) or compressible (over a space shuttle) depending on the flow conditions. It is widely accepted that compressibility effects become significant in a perfect gas when free-stream Mach number becomes more than 0.3. This can be justified by scale of magnitude analysis.

Experiments and simulations are broadly the two different approaches to study any phenomenon. Performing Gas Dynamics experiments typically requires very high speed flow which causes high pressures and temperatures. Generating and handling such conditions is not trivial and this renders numerical simulations as the preferred way of analysis. The scope of this project is the latter one.

1.1 Governing equations

The governing equations of Gas Dynamics are obtained by first applying conservation of mass, momentum and energy to an arbitrary control volume and then, assuming sufficient smoothness of conserved variables. For deriving Gas Dynamics equations, effects of viscousity and conduction are ignored. These assumptions lead to the Euler equations which read as [1]

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} + \frac{\partial \mathbf{H}}{\partial z} = 0. \tag{1.1}$$

Where,

$$\mathbf{Q} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix}, \qquad \mathbf{F} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(\rho E + p) \end{bmatrix}, \qquad \mathbf{G} = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ v(\rho E + p) \end{bmatrix}, \qquad \mathbf{H} = \begin{bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ w(\rho E + p) \end{bmatrix}. \tag{1.2}$$

Here, E is the total specific energy $e + (u^2 + v^2 + w^2)/2$ with e being specific internal energy. ρ, u, v and w refer to density and velocities in 3 dimensional cartesian system. Equation (1.1) is also called the conservative form of Euler equations. Equation (1.1) gives 5 equations in 6 unknowns ρ, u, v, w, p and e. In general, the

system is closed by using the perfect gas equation of state $p = \rho RT$ which yields the relation $p = (\gamma - 1)\rho e$ with γ being the specific heat ratio.

1.2 Finite Volume Method

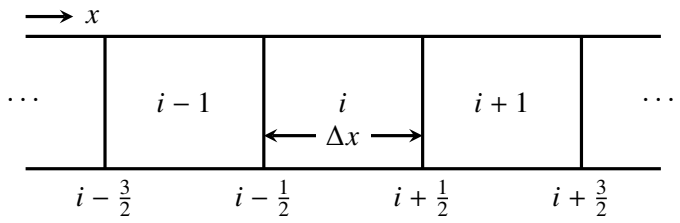


Figure 1.1: Finite Volume discretization of domain

Consider, for simplicity, solving a 1D conservation equation numerically.

$$\frac{\partial q}{\partial t} + \frac{\partial f}{\partial x} = 0 \tag{1.3}$$

Here, q is a conserved variable and $f = f(q)$ is its flux. In Finite Volume Method (FVM), the 1D domain is discretized as shown in Figure 1.1 [2]. The faces of a representative cell i are denoted by $i \pm \frac{1}{2}$.

In FVM, the discrete values Q of a numerical solution are considered as cell averaged values of the smooth solution q. For a 1D grid, it is not necessary to associate Q_i to a specific location. However, for complex multi-dimensional grid, it is assumed later (to calculate fluxes) that Q is associated to the cell centroid. From here on, cell center refers to the centroid of the cell. In mathematical terms, for 1D, this implies the following.

$$Q_i^n = \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} q\left(x, t^n\right) \mathrm{d}x \tag{1.4}$$

To obtain FVM formulation, integrate Equation (1.3) in the limits of cell i; i.e.; from $x = x_{i-\frac{1}{2}}$ to $x = x_{i+\frac{1}{2}}$ between time levels t^n and t^{n+1}. This results in the expression

$$Q_i^{n+1} - Q_i^n = \frac{1}{\Delta x} \int_{t^n}^{t^{n+1}} \left[f_{i-\frac{1}{2}}(t) - f_{i+\frac{1}{2}}(t) \right] \mathrm{d}t. \tag{1.5}$$

Equation (1.5) is obtained from Equation (1.3) after integrating and using Equation (1.4)[1]. Here, $f_{i\pm\frac{1}{2}}(t) = f\left(x_{i\pm\frac{1}{2}}, t\right) = f\left(u\left(x_{i\pm\frac{1}{2}}, t\right)\right)$. The aim of any numerical technique is to approximate $f_{i\pm\frac{1}{2}}(t)$ and hence the integral on R.H.S of Equation (1.5).

Equation (1.5) is referred to as *discrete* form of Finite Volume formulation. To obtain the *semi-discrete* form, integrate Equation (1.3) using Equation (1.4) within the limits of cell i which results in

$$\frac{\mathrm{d}Q_i}{\mathrm{d}t} = \frac{1}{\Delta x}\left(f_{i-\frac{1}{2}}(t) - f_{i+\frac{1}{2}}(t)\right). \tag{1.6}$$

The residual R for an equation is defined as the R.H.S of Equation (1.6).

$$R_i = \frac{1}{\Delta x}\left(f_{i-\frac{1}{2}} - f_{i+\frac{1}{2}}\right) \tag{1.7}$$

[1]In fact, Equation (1.3) is derived from Equation (1.5) which is a conservation/balance equation, by assuming smoothness of q and taking the limit $\Delta x \to 0$. This fact makes Equation (1.5) valid even for a discontinuous solution while Equation (1.3) becomes invalid.

The semi-discrete equation can then be re-written as follows.

$$\frac{\mathrm{d}Q_i}{\mathrm{d}t} = R_i(t) \tag{1.8}$$

1.3 Objective of this project

The aim of this project is to develop high resolution schemes for the Euler equations. In particular, a flux scheme compatible with RK3 time integration and higher order limited interpolation schemes is desired. For this purpose an open source CFD software known as OpenFOAM is used. The existing solvers in OpenFOAM are modified to achieve the purpose. OpenFOAM version 4 is used in this project.

1.4 Organization of this report

Chapter 2 discusses the basic working details about working of OpenFOAM. A reader well equipped with OpenFOAM may opt to skip entire Chapter 2. Chapter 3 discusses the theory of High Resolution schemes to some extent and their implementation in OpenFOAM. Chapter 4 shows the benchmarking results of two ausmPlusUpFoamRK3 – a solver based on AUSM$^+$-up flux scheme and rhoCentralFoamRK3 – a solver based on Kurganov-Tadmor flux scheme. Both the mentioned solvers use TVD-RK3 time integration scheme. The report concludes with Chapter 5 where performance summary of both the solvers, and future work is discussed.

Chapter 2

Introduction to OpenFOAM

OpenFOAM is an open source CFD software which comes with pre-processing, solving and post-processing tools built-in; and is being maintained and upgraded by the OpenFOAM Foundation. In simple words, it is a well structured collection of C++ applications and utilities to solve certain kind of problems by Finite Volume Method. Any user is free create new applications in OpenFOAM using the base applications and utilities provided.

OpenFOAM contains utilities for solving both fluid and solid mechanics problems. Although the type of solvers available are as many as in a commercial CFD software, the availability of source code allows users to explore and come up with better solvers.

2.1 Overview of OpenFOAM

OpenFOAM is a C++ library to create executables, known as *applications* [3]. The source code for all built-in applications is provided to the user along with the executable for the application. OpenFOAM uses the object-oriented programming nature and *templates* mechanism of C++ to the fullest for building solvers in a modular way. This allows ease in creating new applications. The applications in OpenFOAM are of two broad types [3].

1. **Solvers**: those which solve certain governing equations using the Finite Volume Method.
2. **Utilities**: those which are mainly used for pre-processing and post-processing.

The structure of OpenFOAM applications allows representing the governing equation as it is in the solver. For example, to solve the equation

$$\frac{\partial q}{\partial t} + \nabla \cdot \mathbf{f} = 0$$

where, q is the conserved variable and $\mathbf{f}$ is it's flux; the code required to be written is as follows.

```
1 solve(fvm::ddt(q) + fvc::div(f) = 0);
```

Here, `fvm` and `fvc` are `namespaces` which contain the functions `ddt` for time derivative and `div` for divergence operation. This line of code automatically solves the system of equations generated by Finite Volume discretization, using the user specified schemes for `ddt` and `div` on the *fields* `q` and `f` respectively.

2.2 Case Structure

In OpenFOAM terminology, a case refers to directory which contains required files of a simulation organised in a specific manner. Figure 2.1 shows the case structure in OpenFOAM. The description of each of the folders in case directory is as follows.

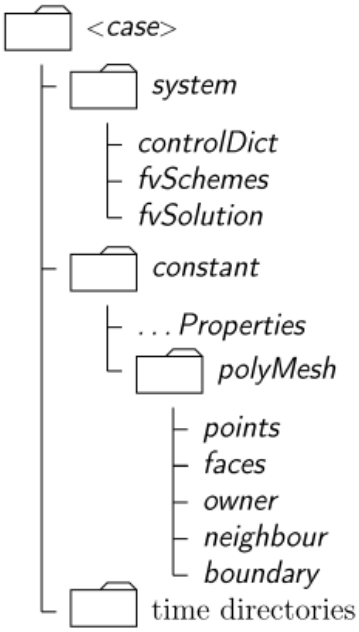


Figure 2.1: Case structure in OpenFOAM [3]

1. The *system* directory contains data related to simulation controls. It consists of *at least* the following files.
   - *controlDict* contains the overall simulation parameters such as start time, stop time, time step control, data output options etc.
   - *fvSolution* contains the solver parameters for matrix equations (which are generated by discretization) such as the type of solver, tolerances etc.
   - *fvSchemes* contains the details of discretization schemes used in the solution.
2. The *constant* contains the full details of mesh (in the sub directory *polyMesh*) and files containing the properties of fluid (*thermoPhysicalProperties*, *turbulenceProperties* etc.). The files and their contents depend on the problem being solved and the solver to be used.
3. Time directories contain the result of the simulation. They contain data for certain fields such as pressure, temperature, velocity etc. depending on the type of solver. The special time directory *0*, which must be present at the start of simulation, contains the initial conditions and boundary conditions of relevant fields in seperate files. The name of any time directory is the time at which the data contained by it is produced during solution.

2.3 Mesh details

In OpenFOAM mesh can be generated either using built-in tools (*blockMesh*, etc.) or from third party mesh generation tools (GEOMESH, etc.) or even by importing meshes generated using other CFD softwares (ANSYS, etc.). The complete details of the mesh are stored as certain files in *polyMesh* sub-directory of

constant directory of a case.

OpenFOAM uses bottom-up approach to discretize a volume as shown in Figure 2.2. The entire domain is treated as a *volume* or collection of *volumes*. A volume is divided into *cells*, which are further divided into *faces*. Each face is comprised of *points*. The boundary of domain is made of *patches*, each of which is a collection of *faces*.

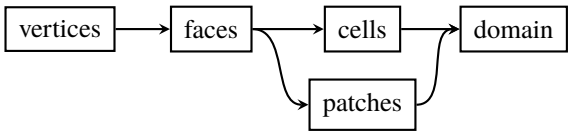


Figure 2.2: Mesh Structure in OpenFOAM

The mesh is built-up from points. To define a point, it's co-ordinate values are required. A face is defined as a collection of such points. A cell is defined as collection of faces. Finally, a volume is defined as a collection of cells. OpenFOAM uses *lists* (see Appendix section B.1) to define points, faces and cells. The *polyMesh* (see Figure 2.1) accordingly contains files for vertices, faces, *owner* and *neighbour* cells, along with a file for domain boundaries.

In OpenFOAM, each face is assigned an owner and a neighbour cell. This classification is done to assign directionality to a face. OpenFOAM treats the face normal pointing from owner to neighbour as positive. For internal faces, the cell with least index is considered the owner, and the other cell is treated as neighbour. For faces belonging to boundary patches, the cell connected to the face is treated as owner. The neighbour is assigned an index -1.

2.4 Solver architecture – Objects used in solvers

A solver is an application at the top most level of source code, which uses many pre-compiled objects or classes. Solver combines the user input data contained in the files of a case (Figure 2.1), with the basic pre-compiled classes and functions to perform simulation.

Every solver in OpenFOAM uses two objects of the classes *Time* and *fvMesh*, to take care of time control and the mesh. A solver uses an instance of *Time* class to read the time controls in *controlDict*, for keeping track of simulation time and data output.

An instance of *fvMesh* contains all the geometric information related to mesh. This *fvMesh* object is also responsible to handle dynamic meshes and local mesh refinement, if any, during the simulation.

To solve a set of equations, the relevant fields are defined at discrete points in space and time. For example, temperature field is a scalar field and velocity field is a vector field. Some of these fields (such as temperature) are defined inside the cells, while some others (such as mass flux) are defined on the surfaces of the cells. To capture all such possible fields, OpenFOAM uses the *template* mechanism of C++. Base classes such as *GeometricField*, *scalar* and *vector* are defined, and used as a template for more detailed description of a field. For example, *surfaceScalarField* stores value of a scalar field that is defined on the faces (mass flux for example), and is a class defined using templates of *scalar*, *surfaceMesh* and *GeometricField*. Similarly a *volVectorField* is a class used for a vector field, whose value is defined at the cell centers (velocity field for example). There are many other such classes, all having functions useful for arithmetic operations on fields.

OpenFOAM uses it's own Input-Output(I/O) format for writing/reading data generated during simulation (see Appendix section B.1). Classes such as *IOobject, IOdictionary*, etc. are used in this regard. The data of

some fields may have to be written into files, while data of some other fields might just be intermediate data, which is not required after the simulation. A solver declares the fields used according to the above criteria using *IOobject*. For illustration, see Listing 2.1 which is a sample of code from a relevant file of a solver named laplacianFoam (see Section 2.5).

Listing 2.1: Declaration of fields using *IOobject*

```
1  Info<< "Reading field T\n" << endl;
2
3  volScalarField T
4  (
5      IOobject
6      (
7          "T",
8          runTime.timeName(),
9          mesh,
10         IOobject::MUST_READ,
11         IOobject::AUTO_WRITE
12     ),
13     mesh
14 );
```

Here, the *volScalarField* T is initialized using *IOobject* class. The fourth and fifth arguments (lines 10,11) of `IOobject` in Listing 2.1 together imply that the field `T` *must* be read (from previous time directories), and it's data is to be written to time directories according to data output controls specified in *controlDict*. Also, since the field `T` is defined as a `volScalarField`, information about cell centers of the mesh has to be linked to this field. The third argument (line 9) does this job. The second argument (line 8) associates a time to the object `T`, which gets updated during the simulation.

Functions which perform operations on fields are grouped into `namespaces`. The most commonly used `namespaces` are *fvc* and *fvm*. These `namespaces` have functions for implementing mathematical operations (such as calculating divergence, gradient, etc.) in a discrete sense on *fields*. Specifically, *fvm* contains functions for implicit operations while *fvc* contains functions for explicit operations[1].

2.5 A walk through laplacianFoam[2]

laplacianFoam is one of the simplest solvers in OpenFOAM. This solver is meant to solve the transient conduction equation.

$$\frac{\partial T}{\partial t} = \nabla^2 \left(D_T T \right) \tag{2.1}$$

Here, T is the temperature, t is time, D_T (a constant) is the thermal diffusivity and ∇^2 is the laplacian operator. Listing 2.2 shows the lines of code of the file `laplacianFoam.C` which is the main file for the solver laplacianFoam.

Listing 2.2: `laplacianFoam.C`

```
32 #include "fvCFD.H"
33 #include "simpleControl.H"
34
35 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

[1] See Appendix section B.2 for more details on *fvm* and *fvc*

[2] The purpose of this section is just to shed some light on the way solvers work in OpenFOAM and may be skipped without loss of continuity.

```
36
37 int main(int argc, char *argv[])
38 {
39     #include "setRootCase.H"
40
41     #include "createTime.H"
42     #include "createMesh.H"
43
44     simpleControl simple(mesh);
45
46     #include "createFields.H"
47
48     // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
49
50     Info<< "\nCalculating temperature distribution\n" << endl;
51
52     while (simple.loop())
53     {
54         Info<< "Time = " << runTime.timeName() << nl << endl;
55
56         while (simple.correctNonOrthogonal())
57         {
58             solve
59             (
60                 fvm::ddt(T) - fvm::laplacian(DT, T)
61             );
62         }
63
64         #include "write.H"
65
66         Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
67             << "  ClockTime = " << runTime.elapsedClockTime() << " s"
68             << nl << endl;
69     }
70
71     Info<< "End\n" << endl;
72
73     return 0;
74 }
```

<32-33> `fvCFD.H` loads many other header files, which contain declarations of useful functions. Two of them are the files `fvm.H` and `fvc.H`, which define the `namespaces` *fvc* and *fvm*.

`simpleControl.H` contains the declaration of a class named *simpleControl*. The purpose of this class would be described later.

<37-39> The parameters of the main function `argc` and `argv` stand for *argument count* and *argument value* respectively. `argv` contains the data of additional parameters with which the application (solver) `laplacianFoam` is run[3]. `setRootCase.H` stores the passed parameters in memory and checks the validity of parameters.

<41-42> `createTime.H` defines an object `runTime` which is of class *Time*. All the simulation run time settings and data output settings in the *system/controlDict* file of the case directory are stored into the object `runTime`.

`createMesh.H` defines an object `mesh` of class *fvMesh*. This `mesh` object is linked to `runTime` and additionally contains the mesh data from *constant/polyMesh/* directory of the case directory.

<44> An object `simple` of type *simpleControl* is defined which is linked to `mesh` and additionally reads data from *system/fvSolution* file. This file contains information about the matrix solvers[4] to be used,

[3]The typical way of running a simulation is by typing the name of the solver in the *terminal* while in the case directory. The simulation can also be run while being in a different directory by passing case directory as an additional parameter along with the name of solver. This is an example of the use of `argc` and `argv`.

and the tolerances for convergence of any iterations during solution.

<46> `createFields.H` defines the temperature variable `T`, initializes it using the data in the file *0/T* and, defines a variable `DT` and sets it's value equal to the value specified in *constant/transportProperties*.

<52> `simple.loop()` checks whether steady-state is reached (for a steady-state simulation) or whether simulation stop time has been attained (for a transient simulation). The choice of the type of simulation (steady-state or transient) can be made by specifying appropriate *value* for the *keyword* (see Appendix section B.1) `ddtSchemes` in the *system/fvSchemes* file of the case directory.

<58-61> These lines form the core of the solver. `solve` function updates the data of `T`. `fvm::ddt` function generates matrix for $(\partial T/\partial t)$ based on discretization scheme specified as value of `ddtSchemes` keyword in *system/fvSchemes* file.

`fvm::laplacian` function generates matrix for laplacian term. OpenFOAM treats `fvm::laplacian(DT,T)` as $\int_P \nabla \cdot (D_T \nabla T)$ with P being a representative cell volume of integration. Discretization of this term in a Finite Volume sense is done by using Gauss divergence theorem to convert the volume integral to surface integral.

$$\begin{aligned} \int_P \nabla \cdot \left(D_T \nabla T\right) &= \int_{\partial P} D_T (\nabla T) \cdot \mathrm{d}\mathbf{S} \\ &\approx \sum_f \left(D_T\right) \frac{\partial T}{\partial n} \left\|\mathbf{S}_f\right\|_2 \end{aligned} \tag{2.2}$$

In Equation (2.2) ∂P is the boundary of P. f is a representative face of P, $\mathbf{S}_f$ is the surface area of the face f, $\|\bullet\|_2$ represents the Euclidean norm and $(\partial T/\partial n)$ represents gradient of T normal to surface. The value of keyword `laplacianSchemes` in *system/fvSchemes* file dictates how $(\partial T/\partial n)$ is approximated[5].

<64> `write.H` checks whether data output must be done at current time based on certain keyword values in *system/controlDict* file.

It can be observed that a solver is just an application that uses previously declared classes and functions to solve the required set of equations. Additionally, some book keeping is done to store the values of required fields.

In the following chapter, Compressible Flow solvers (which are much more involved than laplacianFoam) are considered and implementation of certain new schemes within OpenFOAM framework is discussed.

[4]Matrices are generated when the governing equation (Equation (2.1)) is discretized by FVM. These matrix equations are solved differently based on the type of discretization.

[5]In fact, the keyword also describes how the integral of Equation (2.2) is approximated. In Equation (2.2), the *Gaussian* approximation is used, as might be clear from the R.H.S. of the equation.

Chapter 3

Implementation of High Resolution Schemes

In computational Gas Dynamics terminology, a High Resolution scheme is expected to have the following attributes.

1. Atleast 2^{nd} order accuracy in smooth parts of solution
2. Non-smeared, non-oscillatory capture of discontinuities

In a Finite Difference Method, where derivatives are approximated by Taylor's series expansion, higher accuracy is obtained by taking more terms in the expansion. In FVM, higher accuracy is obtained by using better approximations to the R.H.S of Equation (1.5).

3.1 Insight into Compressible Flow solvers

The governing equations for compressible flows (Equation (1.1)) are *hyperbolic* in nature. Mathematically, hyperbolic systems are most ill behaved in terms of devising numerical techniques/schemes. This is due to the fact information propagation in these systems is by waves. Numerical technique used must be able to capture this phenomenon. It has been established over years that only *conservative* and *upwind* schemes along with some other criteria can give accurate results.

Reconsidering Equation (1.5), the fact that it has been obtained by integrating the governing Partial Differential Equation implies that the technique used for discretization is conservative.

$$Q_i^{n+1} - Q_i^n = \frac{1}{\Delta x} \int_{t^n}^{t^{n+1}} \left[f_{i-\frac{1}{2}}(t) - f_{i+\frac{1}{2}}(t) \right] dt \tag{1.5}$$

To ensure upwind nature of the technique, the integral on R.H.S of Equation (1.5) has to be approximated appropriately. Define numerical flux (F) as

$$F_{i\pm\frac{1}{2}} = \frac{1}{\Delta t} \int_{t^n}^{t^{n+1}} f_{i\pm\frac{1}{2}}(t) \mathrm{d}t. \tag{3.1}$$

Using Equation (3.1), Equation (1.5) can now be re-cast as

$$Q_i^{n+1} - Q_i^n = \frac{\Delta t}{\Delta x} \left(F_{i-\frac{1}{2}} - F_{i+\frac{1}{2}} \right). \tag{3.2}$$

Many of the schemes proposed for inviscid Euler equations obtain $F_{i\pm\frac{1}{2}}$ by solving a *Riemann problem* at the interfaces $i \pm \frac{1}{2}$ approximately. Spatial accuracy depends on how the data available at discrete points is

reconstructed to solve a Riemann problem at the interface, and hence to obtain the numerical flux. To obtain higher temporal accuracy, the semi-discrete equation (Equation (1.6)) is treated as an ODE and solved in multiple steps. In each step, upwind nature of the scheme is preserved by obtaining numerical fluxes using the method described above (i.e.; by solving a Riemann problem). By now, it has been widely accepted that 2^{nd} order accuracy in space is sufficient to produce good results. For the temporal part, upto 4^{th} order accurate schemes have been used depending upon the problem being solved.

3.2 Compressible Flow solvers in OpenFOAM

It has been described in Section 2.5 how laplacianFoam works. Solvers in OpenFOAM are built in a modular way. In a sense, they are just an ordered compilation of existing utilities. As a result, a single solver can be used with many kinds of discretization techniques to solve a set of equations. Most of the control on discretization is given to user through various files in the case directory. Compressible flow solvers' algorithm can be split into three steps.

1. Reconstruction
2. Flux calculation
3. Time stepping

In step 1, the data of fields available at cell centers of the domain is interpolated to faces, seting up a Riemann Problem at each face. The Riemann Problem is solved in step 2 to obtain the fluxes. The fluxes, along with previous time step values, are used to update required fields in step 3.

The main solver for compressible flows in OpenFOAM is rhoCentralFoam. Unlike laplacianFoam, where user has complete choice of all the discretization schemes being used, in rhoCentralFoam or in any other compressible flow solver, the flux scheme is specific to a solver. This is because, as stated in Section 3.1, most of the flux schemes involve solving an approximate Riemann Problem at the faces in the domain. Thus, the computation of flux of a variable requires the data of all the variables. This would not be possible by simple Finite Volume operations provided by *fvm* and *fvc* which take only a single field as a parameter (refer Appendix section B.2). To summarize, in a compressible flow solver, the algorithm for step 2 is unique and can't be changed by the user. rhoCentralFoam uses the central upwind scheme of Kurganov and Tadmor [4] (see Subsection 3.3.2) for flux calculation.

3.3 High Resolution schemes in OpenFOAM

High resolution schemes involve using better approximations in all the three steps: Reconstruction, Flux calculation and Time integration. This section considers implementation of each of the three steps in OpenFOAM. For simplicity, 1D formulation on a uniform grid is considered in this section. Noting that flux scheme is fixed for a given solver in OpenFOAM, implementing a different flux scheme would require making a new solver in OpenFOAM.

3.3.1 Higher Order Reconstruction

In conservative schemes, the update formula (Equation (1.5)) requires calculation of numerical flux F at the interfaces of a cell. To obtain the flux, as mentioned in Section 3.1, many schemes solve a Riemann Problem

at the interface approximately. The accuracy of the solution also depends on how well the Riemann Problem is set up, along with how well it is solved.

Let $Q^L_{i+\frac{1}{2}}$ represent the value of Q to the left of $i+\frac{1}{2}$ interface. Similarly, let $Q^R_{i+\frac{1}{2}}$ represent the value to the right. These are commonly called *left interpolated value* and *right interpolated value* respectively. The job of reconstruction is to come up with sensible estimates for $Q^{L,R}_{i+\frac{1}{2}}$. The simplest approximation could be the following, where the value of Q is assumed constant within the cell (see Figure 3.1).

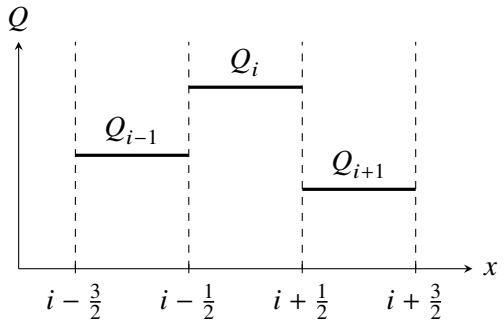


Figure 3.1: First order reconstruction

$$\begin{aligned} Q^L_{i+\frac{1}{2}} &= Q_i \\ Q^R_{i+\frac{1}{2}} &= Q_{i+1} \end{aligned} \tag{3.3}$$

In a sense, Equation (3.3) represents 1^{st} order reconstruction since $Q^{L,R}_{i+\frac{1}{2}}$ are being computed using data only from a single cell. Equation (3.3) is also referred to as upwind reconstruction or interpolation[1]. It has been observed that upwind reconstruction gives smeared or diffused results, meaning, the jump of a discontinuity doesn't get captured well. However, the solution is generally observed to be *stable* (see Appendix section A.1) when upwind interpolation scheme is used.

In higher order reconstruction, data from more than one cell has to be used for obtaining left and right interpolated values. It would be logical to obtain these values by fitting non-constant profile for Q within a cell. For example, fitting a linear profile would results in 2^{nd} order reconstruction. It has been observed that although higher order reconstruction schemes capture the jump better (compared to upwind schemes), they produce unphysical oscillations in the vicinity of a discontinuity becauses they lag sufficient upwind nature required. This may lead to instability in solution.

Over time, researchers have come up with various approaches to avoid these oscillations. One of these is to use a blend of upwind and higher order schemes to capture discontinuities in a non-diffused and non-oscillatory manner. Such a scheme is called a *Limited Interpolation scheme*.

In a limited scheme, $Q^L_{i+\frac{1}{2}}$ is approximated as

$$Q^L_{i+\frac{1}{2}} = Q^{L,U}_{i+\frac{1}{2}} + \beta \left[Q^{L,H}_{i+\frac{1}{2}} - Q^{L,U}_{i+\frac{1}{2}} \right]. \tag{3.4}$$

Where, $Q^{L,U}_{i+\frac{1}{2}}$ represents $Q^L_{i+\frac{1}{2}}$ approximated by the upwind scheme and $Q^{L,H}_{i+\frac{1}{2}}$ represents $Q^L_{i+\frac{1}{2}}$ approximated by a higher order scheme. β is called the *limiter*. The function of a limiter is to locally give appropriate weights to upwind and higher order scheme, depending on the gradient of Q. If the solution is smooth without too much changes, more weight can be given to higher order reconstruction; and vice–versa, if the

[1]Strictly speaking, Equation (3.3) represents 1^{st} order upwind scheme. There are higher order upwind schemes which use data from more than one cell on one side of a face. However, for simplicity, 1^{st} order upwind will be referred to as upwind scheme in this report.

solution is rapidly changing. In OpenFOAM, for limited schemes, centered linear interpolation is the higher order scheme. This implies that on a uniform grid,

$$Q^{L,H}_{i+\frac{1}{2}} = Q^{R,H}_{i+\frac{1}{2}} = \frac{Q_i + Q_{i+1}}{2} \quad \text{in OpenFOAM.}$$

For details on how β is calculated in OpenFOAM, see Appendix section B.3.

3.3.2 Kurganov-Tadmor flux scheme

1D euler equations can be written as

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} = 0. \tag{3.5}$$

Where,

$$\mathbf{Q} = \begin{bmatrix} \rho \\ \rho u \\ \rho E \end{bmatrix}, \qquad \mathbf{F} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho E u + p u \end{bmatrix} \tag{3.6}$$

Let $\mathbf{F}^*$ represent the numerical flux vector at a face which has to be formulated. Let $\mathbf{Q}^L$ and $\mathbf{Q}^R$ represent left and right interpolated values of $\mathbf{Q}$ at the face being considered. The Kurganov-Tadmor flux formulation is given as [4]

$$\mathbf{F}^* = \frac{1}{2}\left\{\left[\mathbf{F}\left(\mathbf{Q}^L\right) + \mathbf{F}\left(\mathbf{Q}^R\right)\right] - a^*\left[\mathbf{Q}^R - \mathbf{Q}^L\right]\right\}. \tag{3.7}$$

Here $\mathbf{F}(\mathbf{Q})$ represents that this flux vector is calculated using the state vector $\mathbf{Q}$ according to Equation (3.6) where, a^* is given by

$$a^* = \max_i \left\{ \max\left\{\left|\lambda_i^L\right|, \left|\lambda_i^R\right|\right\} \right\} \quad \text{for } i = 1,\ 2,\ 3. \tag{3.8}$$

Here, $\lambda_i^{L,R}$ represents eigen values calculated using the state $\mathbf{Q}^{L,R}$. The eigen values of 1D euler equation system are $u - a$, u and $u + a$ where a is the speed of sound given by $\sqrt{\gamma p/\rho}$. On a side note, the numerical flux $\mathbf{F}^*$ for this scheme is obtained by assuming a two wave solution to the Riemann Problem set up at interface with the waves moving at speeds of $\pm a^*$.

3.3.3 AUSM$^+$-up flux scheme

AUSM stands for *Advection Upstream Splitting Method*. It was originally put forward by Liou and has been improved since then. AUSM$^+$-up is one of the recent schemes from this family suggested by Liou [5]. The idea of AUSM family schemes is to treat the flux vector as a combination of advection and pressure flux. The advection and pressure fluxes are then treated independently.

Re-consider 1D Euler equations (Equations (3.5) and (3.6)). The flux, $\mathbf{F}$, is now split into advection and pressure term. In other words, all terms containing u are treated seperately as advection terms. The expression for $\mathbf{F}$ can be re-written as

$$\mathbf{F}_{1/2} = \rho_{1/2} u_{1/2} \begin{bmatrix} 1 \\ u \\ H \end{bmatrix}_{L/R} + \begin{bmatrix} 0 \\ p_{1/2} \\ 0 \end{bmatrix}. \tag{3.9}$$

Here,

$$\{\bullet\}_{L/R} \triangleq \begin{cases} \{\bullet\}_L & \text{if } u_{1/2} > 0 \\ \{\bullet\}_R & \text{otherwise} \end{cases} \tag{3.10}$$

and H is the total enthalpy, related to E as $H = E + p/\rho$. All the variables with subscript 1/2 represent local values at the interface being considered. Variables with subscript L or R represent values to the left and right of interface respectively.

Further, $u_{1/2}$ is re-written as $u_{1/2} = a_{1/2}M_{1/2}$. Now, there are four quantities to be estimated to obtain the flux: $\rho_{1/2}$, $a_{1/2}$, $M_{1/2}$ and $p_{1/2}$.

1. $\rho_{1/2} = \rho_{L/R}$
2. $a_{1/2} = \min\left(\hat{a}_L, \hat{a}_R\right)$, where

$$\hat{a} = \frac{(a^*)^2}{\max(a^*, |u|)} \quad \text{and} \quad (a^*)^2 = \frac{2(\gamma-1)}{\gamma+1}H$$

3. Define M_L and M_R as shown in Equation (3.11). Note that the Mach number definition here is different from conventional definition.

$$M_{L,R} \triangleq \frac{u_{L,R}}{a_{1/2}} \tag{3.11}$$

The interface Mach number is expanded in terms of left and right Mach numbers.

$$M_{1/2} = \mathcal{M}^+_{(4)}(M_L) + \mathcal{M}^-_{(4)}(M_R) + M_p \tag{3.12}$$

Where,

$$\mathcal{M}^\pm_{(1)}(M) = \frac{1}{2}(M \pm |M|), \tag{3.13}$$

$$\mathcal{M}^\pm_{(2)}(M) = \pm\frac{1}{4}(M \pm 1)^2, \tag{3.14}$$

$$\mathcal{M}^\pm_{(4)}(M) = \begin{cases} \mathcal{M}^\pm_{(1)}(M) & \text{if } |M| \geq 1 \\ \mathcal{M}^\pm_{(2)}(M)\left[1 \mp 16\beta\mathcal{M}^\pm_{(2)}\right] & \text{otherwise} \end{cases} \tag{3.15}$$

And

$$M_p = -\frac{K_p}{f_a}\max(1 - \sigma\overline{M}^2, 0)\frac{p_R - p_L}{\left(\frac{\rho_R+\rho_L}{2}\right)\left(a_{1/2}\right)^2} \tag{3.16}$$

where, $\overline{M}^2 = (M_L^2 + M_R^2)/2$ with K_p, β and σ being constants such that $0 \leq K_p \leq 1$, $\beta = 1/8$ and $\sigma \leq 1$. f_a is also a constant (see below).

4. The interface pressure is approximated as

$$p_{1/2} = \mathcal{P}^+_{(5)}(M_L) \times p_L + \mathcal{P}^-_{(5)}(M_R) \times p_R + p_u, \tag{3.17}$$

where,

$$\mathcal{P}^\pm_{(5)}(M) = \begin{cases} \frac{1}{M}\mathcal{M}^\pm_{(1)}(M) & \text{if } |M| \geq 1 \\ \mathcal{M}^\pm_{(2)}(M)\left[(\pm 2 - M) \mp 16\alpha M\mathcal{M}^\mp_{(2)}(M)\right] & \text{otherwise} \end{cases} \tag{3.18}$$

$$p_u = -\left[K_u f_a\right] \times \left[\mathcal{P}^+_{(5)}(M_L)\mathcal{P}^-_{(5)}(M_R)\right] \times (\rho_L + \rho_R)a_{1/2}(u_R - u_L) \tag{3.19}$$

Here, K_u is a constant such that $0 \leq K_u \leq 1$. α is a constant such that $-\frac{3}{4} \leq \alpha \leq \frac{3}{16}$. α and f_a are related as

$$\alpha = \frac{3}{16}\left(-4 + 5f_a^2\right).$$

Liou [5] proposes to choose $f_a = 1 \implies \alpha = \frac{3}{16}$; $\beta = \frac{1}{8}$; $K_u = 0.75$; $K_p = 0.25$ and $\sigma = 1$.

3.3.4 Higher Order time integration schemes

It has been stated in Section 3.2, that user has the choice of reconstruction and time stepping schemes for a compressible flow solver. OpenFOAM offers many higher (2^{nd}) order limited spatial reconstruction schemes (MUSCL, vanLeer, SuperBee etc.) which perform sufficiently well for many problems. However, the time schemes offered are at most 2^{nd} order accurate. An attempt is made in this project to implement a 3^{rd} order accurate temporal scheme: TVD-RK3.

TVD-RK3 is a Total Variation Diminishing (refer Appendix section A.1) type Runge-Kutta (RK) method involving three substeps. Hence the name TVD-RK3. The formulation of TVD-RK3 is as given below [6].

$$\begin{aligned} Q_i^1 &= Q_i^n + \Delta t R_i^n \\ Q_i^2 &= \frac{3}{4}Q_i^n + \frac{1}{4}Q_i^1 + \frac{1}{4}\Delta t R_i^1 \\ Q_i^{n+1} &= \frac{1}{3}Q_i^n + \frac{2}{3}Q_i^2 + \frac{2}{3}\Delta t R_i^2 \end{aligned} \tag{3.20}$$

Here, the residuals R_i^1 and R_i^2 are computed using Q^1 and Q^2 respectively. It must noted here that since the scheme involves multiple flux computations, a single time integration class[2] is not sufficient for time stepping due to reasons similar to those described in Section 3.2. The crux of the argument is that since flux scheme is unique for a solver, a TVD-RK3 time integration class cannot be made. Hence, a new solver has to be made to implement TVD-RK3 scheme comprimising on user's choice for step 3 (i.e.; time integration). A TVD-RK3 compressible flow solver would thus give only spatial reconstruction schemes as a choice for user.

A new solver rhoCentralFoamRK3 has been made to implement TVD-RK3 time scheme along with Kurganov-Tadmor flux scheme. Similarly, ausmPlusUpFoamRK3 has been made to implement AUSM$^+$-up flux scheme with TVD-RK3 time scheme. The use of AUSM$^+$-up flux scheme is inspired by hy2Foam [7] – a Hypersonic flow solver developed by a group of researchers lead by Dr. Vincent Casseau at the University of Strathclyde. ausmPlusUpFoamRK3 utilizes the flux calculation part of hy2Foam along with corrections and additions to implement TVD-RK3 time scheme. The performance of these solvers on various test cases is shown and discussed in Chapter 4. On a side note, a technical detail regarding implementing RK3 scheme in OpenFOAM is discussed in Appendix section B.6.

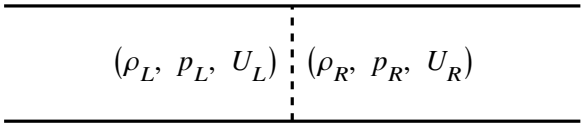
[2] In OpenFOAM all time integration schemes are made available as classes which contain functions required for time stepping.

Chapter 4

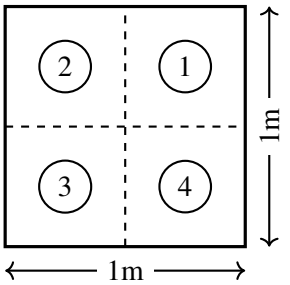
Benchmarking Results

Benchmarking refers to a process of comparing results of a new solver with analytical solution (if exists) or with well established and widely accepted numerical solution. For Gas Dynamics solvers, 1D Riemann Problems serve as good test cases since the analytical solution for these problems is available. Riemann Problems are the class of problems with discontinuity in initial conditions. A generic 1D problem's initial condition is shown in Figure 4.1a. The interface is suddenly removed at $t = 0$ seconds.

Similar to 1D, 2D Riemann Problems have also been set up (Figure 4.1b). The problem would involve 4 quadrants of a rectangle with different initial conditions, evolving in time. Apart from 2D Riemann problems, there are other 2D test cases which simulate more realistic flows.



(a) 1D Riemann Problem.



(b) 2D Riemann Problem.

Figure 4.1: Schematics of 1D and 2D Riemann problems.

4.1 1D tests

There are five 1D tests suggested by Toro [1]. Apart from these, three other test cases were considered. The left and right states are as shown in Table 4.1. In OpenFOAM, every field has dimension associated with it. The dimensions of the data in Table 4.1 are in S.I. units. Although the numerical values of data are not physically plausible, they are used to test the robustness of the solver.

Test 6 is the *Sod Shock Tube problem* and Test 1 is the modified Sod problem with non-zero velocity on the left. Test 2 (also called the *123 problem*) contains two strong expansion waves moving away from each other with very low pressure in between. This test is used for testing positivity preserving property of a scheme. Test 3 and Test 7 contain very strong shock and expansion waves moving away from each other. Test 4 contains two strong shocks moving towards each other. Test 5 is like Test 3 but with a background velocity.

All the numerical results being shown here were generated considering initial discontinuity at $x = 0$

and using 100 cells (except for Test 8, where 500 cells were used) with a maximum Courant number (refer Appendix section B.5) of 0.3 except for Test 2, which required it to be 0.1. vanLeer interpolation scheme was used for all test cases except Test 2, after comparing results of various interpolation schemes such as Minmod, vanAlbada, OSPRE and SUPERBEE. Test 2 required upwind scheme to be used. Also, the values of AUSM$^+$-up constants chosen are shown in Table 4.2. The analytical solution is represented by a solid line and numerical solutions are represented by symbols.

Table 4.1: Initial data for 1D tests. All values are in S.I. units. Here, KT stands for Kurganov-Tadmor and APUP stands for AUSM$^+$-up.

Test #	ρ_L	ρ_R	p_L	p_R	U_L	U_R	time	Execution time	
								KT	APUP
1	1.0	0.125	1.0	0.1	0.75	0.0	0.2	0.4	3.4
2	1.0	1.0	0.4	0.4	-2.0	2.0	0.15	0.6	1.6
3	1.0	1.0	1000.0	0.01	0.0	0.0	0.012	0.5	3.2
4	5.99924	5.99242	460.894	46.095	19.5975	-6.19633	0.035	0.6	1.9
5	1.0	1.0	1000.0	0.01	-19.59745	-19.59745	0.012	0.5	3.0
6	1.0	0.125	1.0	0.1	0.0	0.0	0.21	0.3	0.7
7	1.0	1.0	0.01	100.0	0.0	0.0	0.035	0.3	3.3
8	0.12619	6.59149	732.9289	3.15448	8.90475	2.26542	0.039	12.4	161

Table 4.2: Default AUSM$^+$-up constants. $f_a = 1 \implies \alpha = 0.1875$.

f_a	σ	K_u	K_p	β
1.0	1.0	0.75	0.25	0.125

Both the solvers crashed for Test 2 with vanLeer interpolation scheme and maximum Courant number of 0.3. However, they ran fine on reducing the Courant number to 0.1 and using upwind interpolation scheme. It is worth noting that AUSM$^+$-up flux scheme slightly under-predicts p and ρ compared to their respective exact values. Both the solvers give reasonable and very similar results for all cases except for Test 5 which contains a stationary contact discontinuity at $x = 0$. Referring to Figure 4.6, the Kurganov-Tadmor flux scheme seems to have problem in resolving the stationary contact wave. This is evident in couple of 2D test cases too (see Section 4.2). Both the schemes produce oscillations in cases involving very strong contact discontinuities (see Figures 4.4, 4.8 and 4.9). Although the mesh for these mentioned cases is relatively coarse to make such comments, the oscillations produced by both schemes in Test 8, near the strong contact wave, add more weight to the above argument.

4.2 2D Riemann Problems

The six 2D Riemann test cases suggested by Liska and Wendroff [8] are considered. The initial conditions are shown in Table 4.3. vanLeer interpolation scheme was used for all cases with a maximum Courant number of 0.3. Initially, a fine grid was used to simulate these 2D cases on 4 core parallel processor. But a problem (described in Appendix section B.6) had been encountered in doing so and hence, all these cases have been run on uniform 100×100 grid on a single core. All the results show 24 density contours on pressure colour map. The aim of performing a 2D Riemann problem simulation is to look at the resolution of discontinuities and expansion waves qualitatively. Considering the coarse grid chosen, it is only possible to

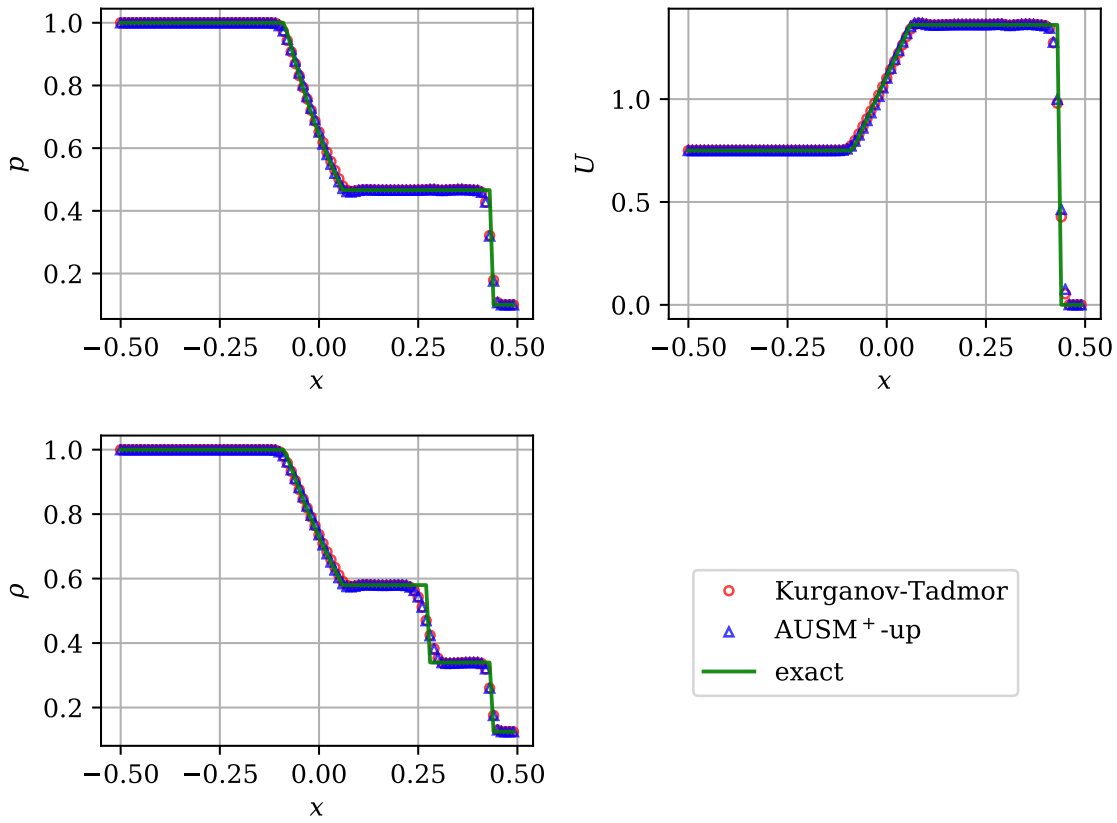


Figure 4.2: Results of Test 1. Initial discontinuity at $x = 0$, compared with analytical solution at $t = 0.2$s.

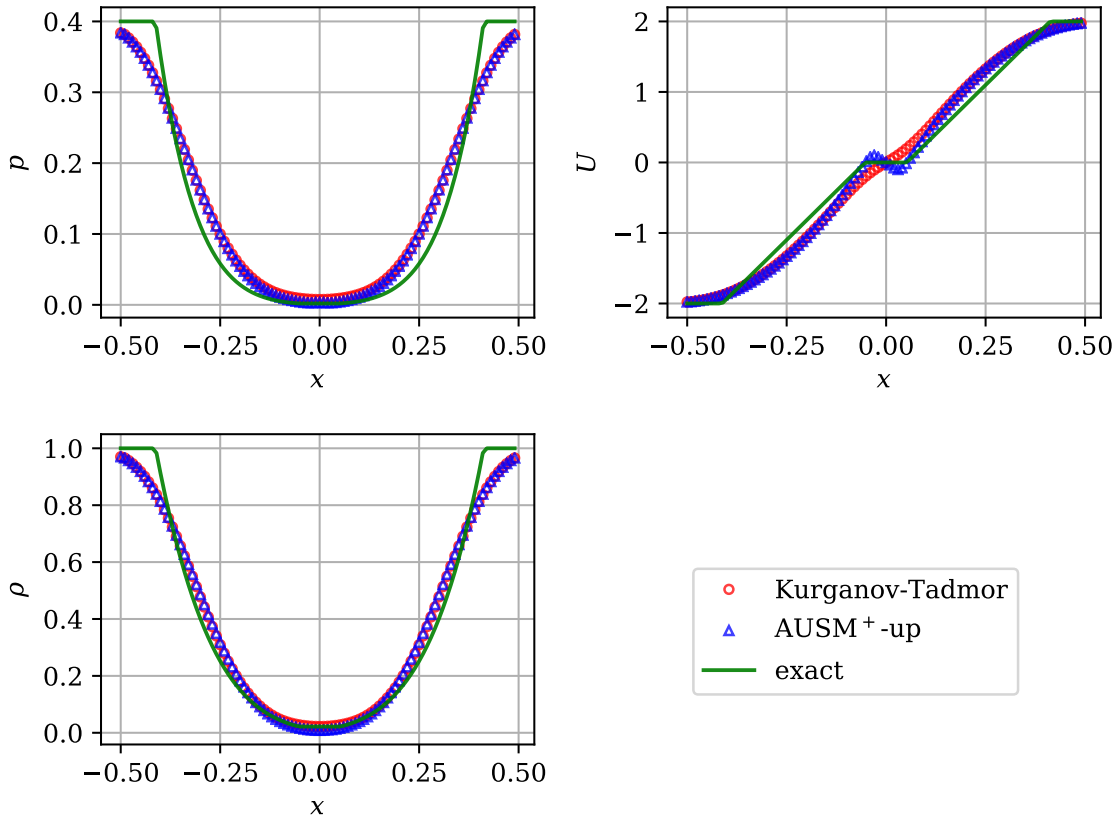


Figure 4.3: Results of Test 2. Initial discontinuity at $x = 0$, compared with analytical solution at $t = 0.15$s. Both solvers were run using maximum Courant number of 0.1 with upwind interpolation scheme.

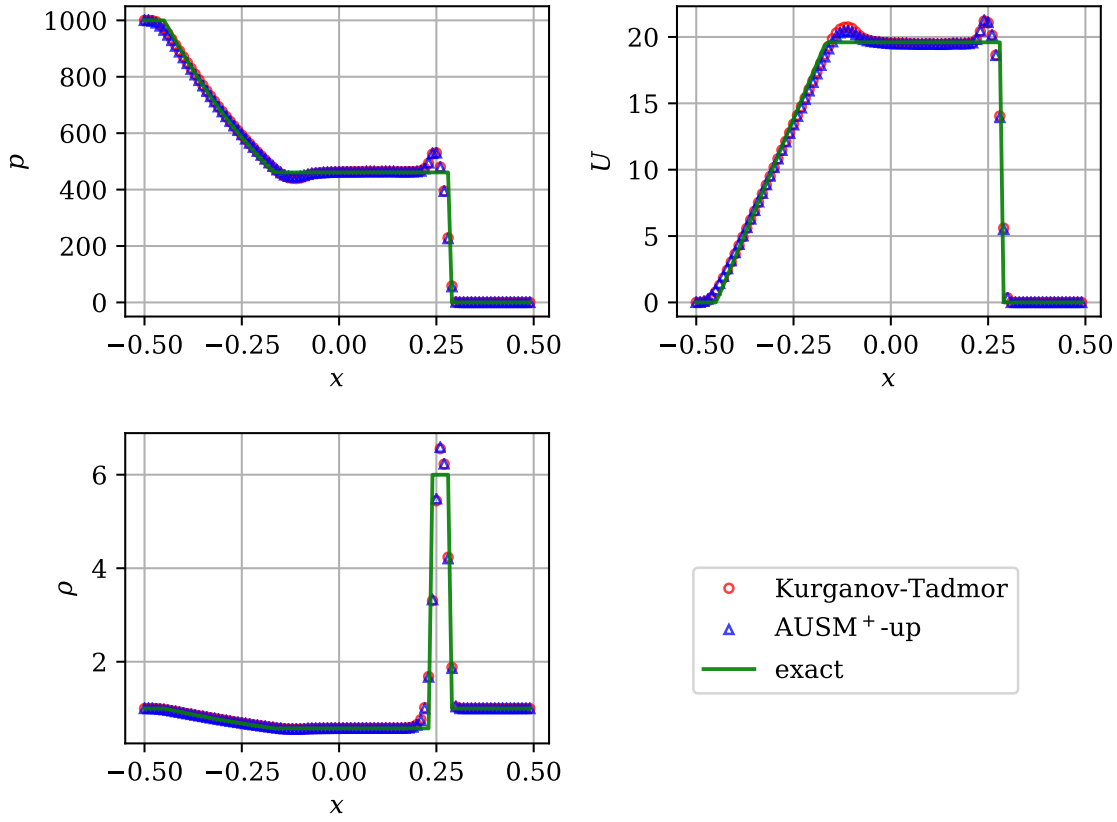


Figure 4.4: Results of Test 3. Initial discontinuity at $x = 0$, compared with analytical solution at $t = 0.012$s.

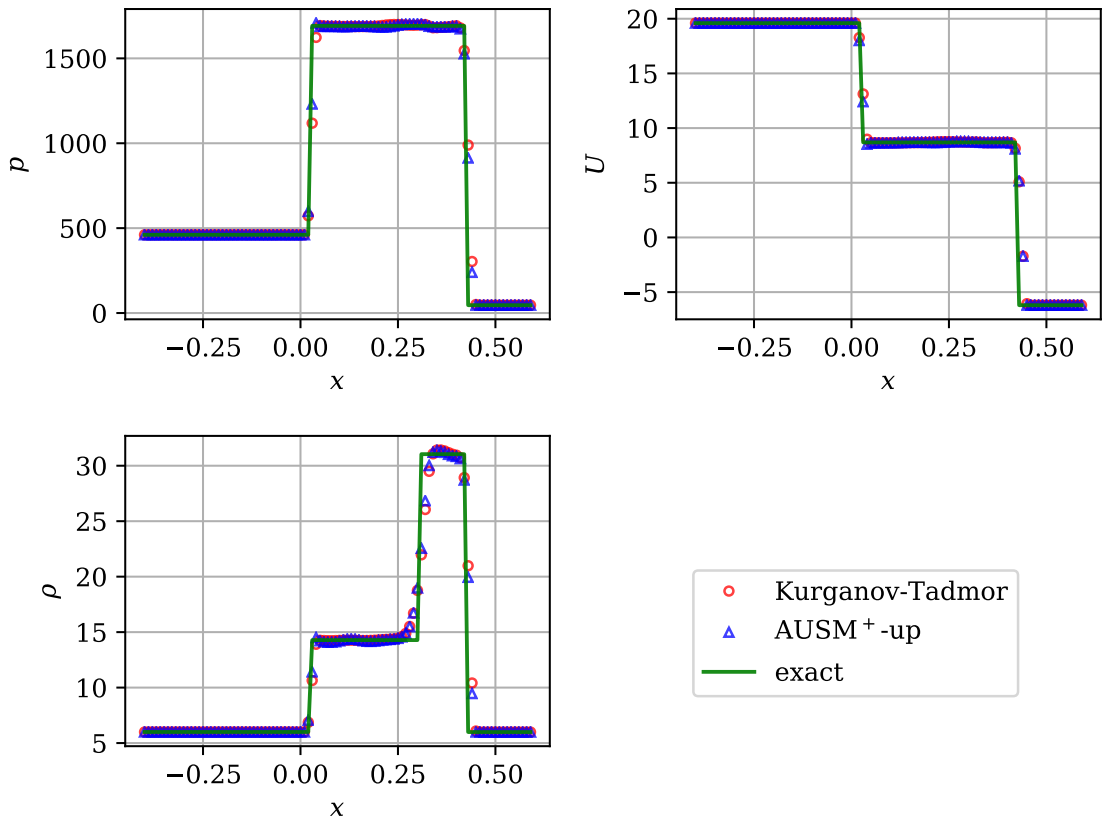


Figure 4.5: Results of Test 4. Initial discontinuity at $x = 0$, compared with analytical solution at $t = 0.035$s.

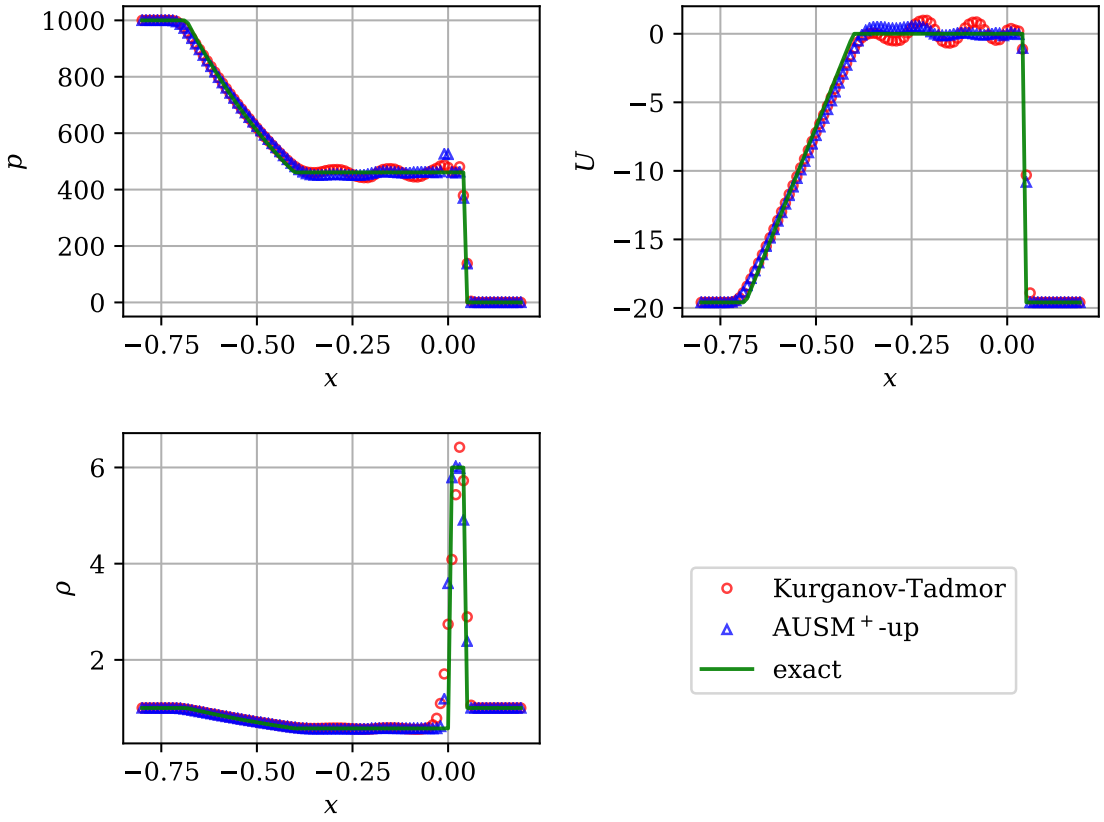


Figure 4.6: Results of Test 5. Initial discontinuity at $x = 0$, compared with analytical solution at $t = 0.012$s.

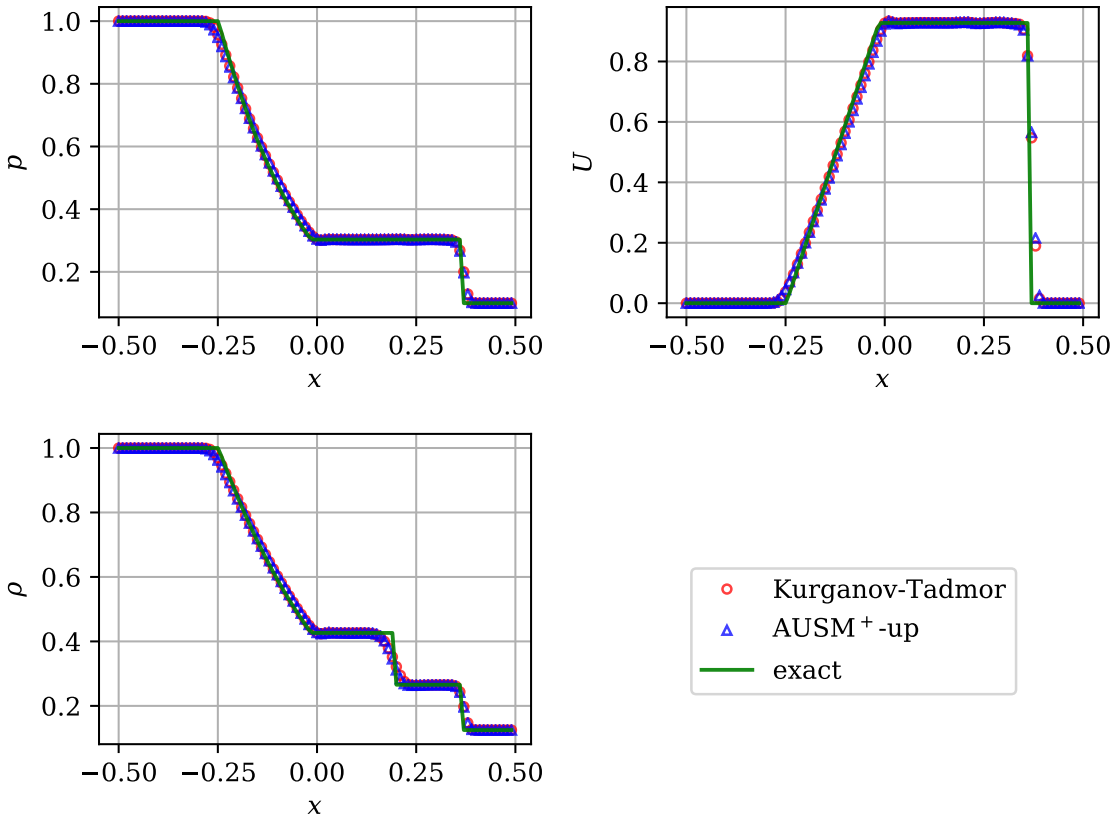


Figure 4.7: Results of Test 6. Initial discontinuity at $x = 0$, compared with analytical solution at $t = 0.012$s.

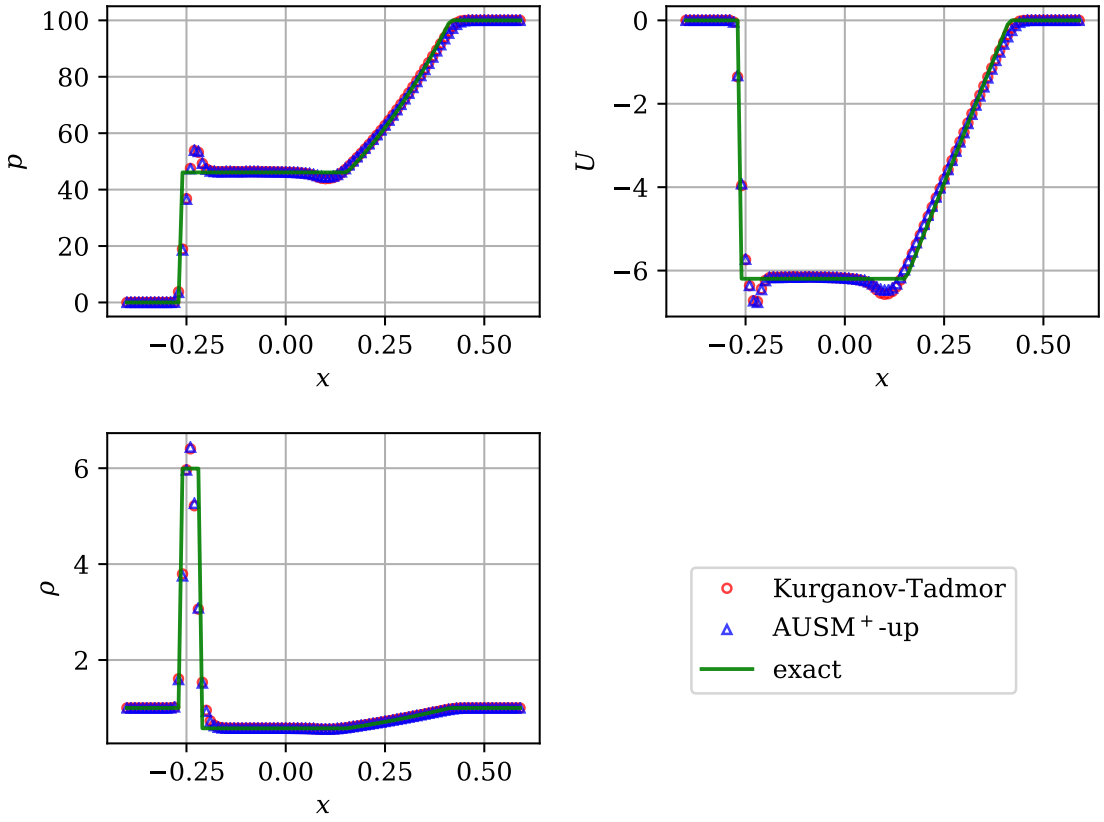


Figure 4.8: Results of Test 7. Initial discontinuity at $x = 0$, compared with analytical solution at $t = 0.035$s.

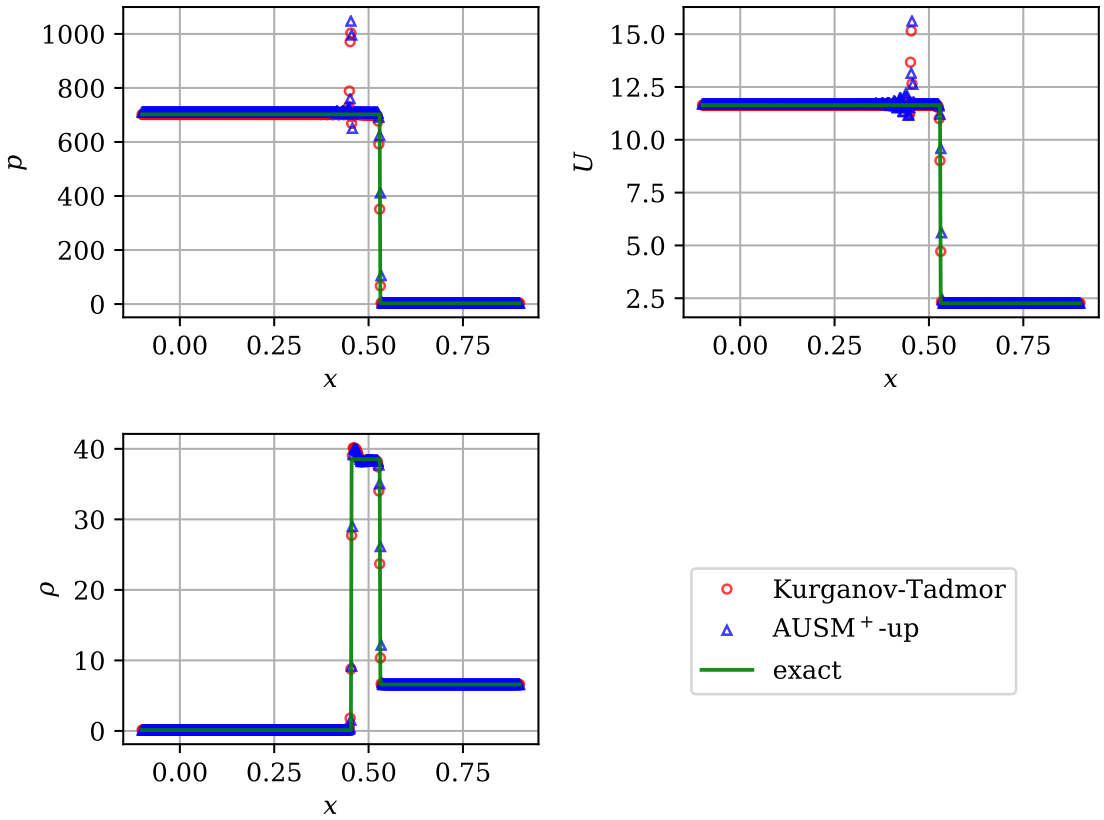


Figure 4.9: Results of Test 8. Initial discontinuity at $x = 0$, compared with analytical solution at $t = 0.039$s.

provide comparison between the flux schemes rather than providing assessment of performance of the flux schemes individually.

Test 9 (Figure 4.10) consists 4 initial shocks which result in formation other discontinuities (in the central region), after interacting with each other, which are not well captured since the grid is very coarse. The resolution of primary shocks is similar for both schemes with AUSM$^+$-up being more oscillatory in the top right region which is a zero velocity region.

Test 10 (Figure 4.11) consists 4 straight shocks initially which produce 2 additions curved shocks that decelerate the uniform velocity flow in the bottom left region to zero velocity in top right region. Both the schemes perform almost similarly for this case. A point to note is that pressure and density should be symmetric about the symmetry-axis of the curved shocks. Slight un-symmetry can be noticed in the result with AUSM$^+$-up.

Test 11 (Figure 4.12) consists 4 initial slip lines[1] which later get curved and also produce expansion waves. Performance of both the schemes is similar for this case, with the slip lines not being captured well.

Test 12 (Figure 4.13) contains 2 shocks and 2 stationary slip lines initially which produce 2 additional curved shocks later. AUSM$^+$-up scheme captures the stationary slip lines very well compared to Kurganov-Tadmor scheme which produces too many density oscillations. The whole solution must be symmetric about the 45° diagonal. Both the schemes retain this symmetry.

Test 13 (Figure 4.14) initially consists 2 slip lines a shock, and an additional discontinuity. Both the slip lines move slowly and the additional discontinuity transforms into an expansion wave. An additional curved contact discontinuity is produced in the middle region. Both the schemes give similar results, with AUSM$^+$-up being marginally better at resolving the contact discontinuity and the slip lines.

Test 14 (Figure 4.15) initially consists 2 stationary slip lines, a shock, and a discontinuity. The slip lines get curved and an expansion wave is formed due to the discontinuity. Again the ability of AUSM$^+$-up in capturing the stationary slip lines well compared to Kurganov-Tadmor surfaces here. Due to this very reason, the low pressure region generated due to the interaction of the slip lines and the shock gets captured by AUSM$^+$-up.

4.3 Further 2D test cases

Three other 2D test cases were considered apart from the six 2D Riemann problems.

The "square–shock interaction" (Test 15) case consists a shock of Mach number 2 moving into a medium with state variables $p = 1$, $\rho = 1$ and $\mathbf{U} = (0, 0)$ containing a square prism of side 0.1m. The geometry of this case is described in Figure 4.16. Post shock conditions are obtained using the Rankine-Hugoniot shock equations. The boundary conditions corresponding to wall are no-slip for velocity (i.e.; zero normal velocity) and zero normal gradient condition for pressure and temperature. A uniform mesh with $\Delta x = \Delta y = 1/100$ was chosen.

The "cylinder–shock interaction" (Test 16) case is similar to square–shock case except that the square is now replaced with a cylinder of diameter 0.2m and, the shock Mach number is increased to 3. The mesh for this case consists 30 triangular cells to capture the cylinder arc. Along the symmetry edge, triangular cell geometry was retained for 15 cells, followed by 30 rectangular cells on both sides of cylinder. 50 cells were used along the vertical left and right edges. Rectangular cells were graded slightly in both directions to get a

[1] A slip line is like a contact discontinuity in 2D across which, normal velocity in continuous and tangential velocity is discontinuous.

Table 4.3: Initial conditions for 2D Riemann problems. All values are in S.I. units. Here, KT stands for Kurganov-Tadmor and APUP stands for AUSM$^+$-up.

Test #		ρ_L	ρ_R	p_L	p_R	$\mathbf{U}_L$	$\mathbf{U}_R$	time	Execution time	
									KT	APUP
9	Upper	0.5323	1.5	0.3	1.5	(1.206, 0)	(0, 0)	0.3	17	6381
	Lower	0.138	0.5323	0.029	0.3	(1.206, 1.206)	(0, 1.206)			
10	Upper	0.5065	1.1	0.35	1.1	(0.8939, 0)	(0, 0)	0.25	14	7667
	Lower	1.1	0.5065	1.1	0.35	(0.8939, 0.8939)	(0, 0.8939)			
11	Upper	2.0	1.0	1.0	1.0	(0.75, 0.5)	(0.75, −0.5)	0.3	15	7242
	Lower	1.0	3.0	1.0	1.0	(−0.75, 0.5)	(−0.75, −0.5)			
12	Upper	1.0	0.5313	1.0	0.4	(0.7276, 0)	(0, 0)	0.25	13	7275
	Lower	0.8	1.0	1.0	1.0	(0, 0)	(0, 0.7276)			
13	Upper	0.5197	1.0	0.4	1.0	(−0.6259, −0.3)	(0.1, −0.3)	0.2	10	5636
	Lower	0.8	0.5313	0.4	0.4	(0.1, −0.3)	(0.1, 0.4276)			
14	Upper	2.0	1.0	1.0	1.0	(0, −0.3)	(0, −0.4)	0.3	15	7169
	Lower	1.0625	0.5197	0.4	0.4	(0, 0.2145)	(0, −1.1259)			

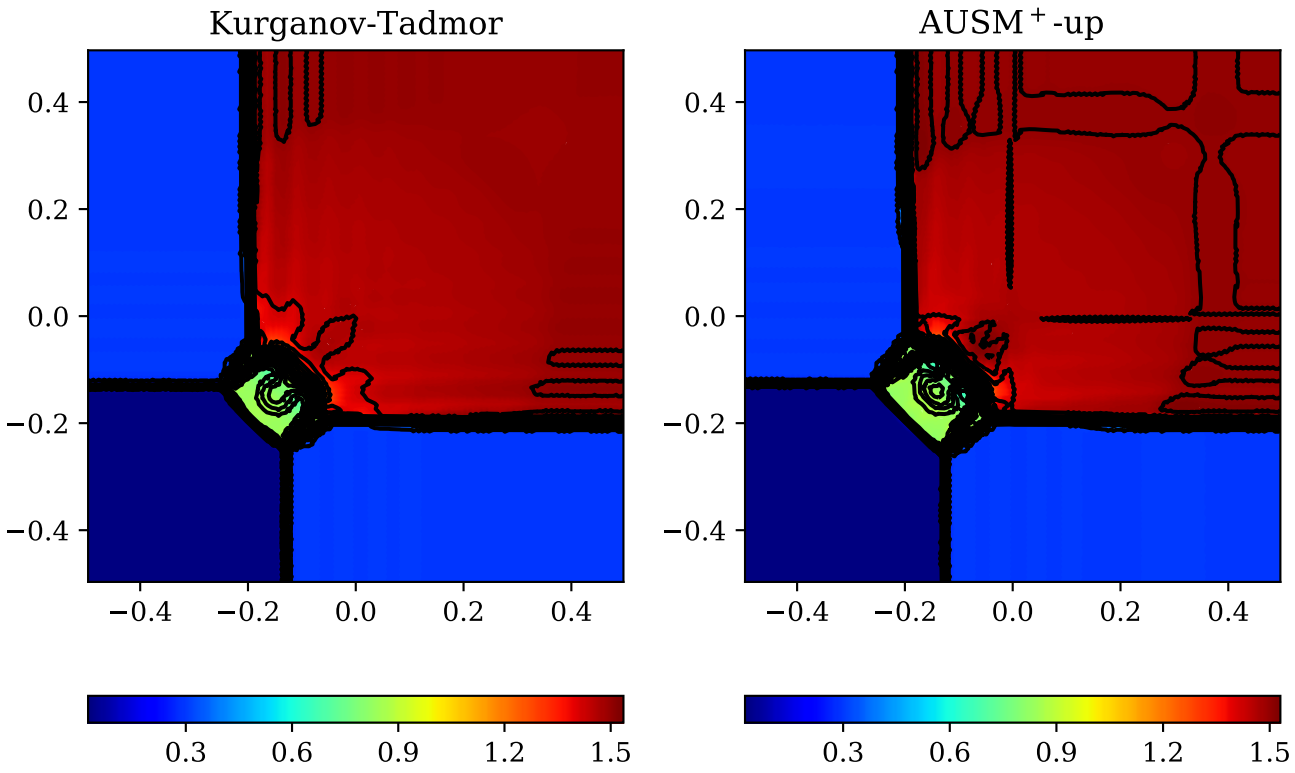


Figure 4.10: Results of Test 9 at t = 0.3s.

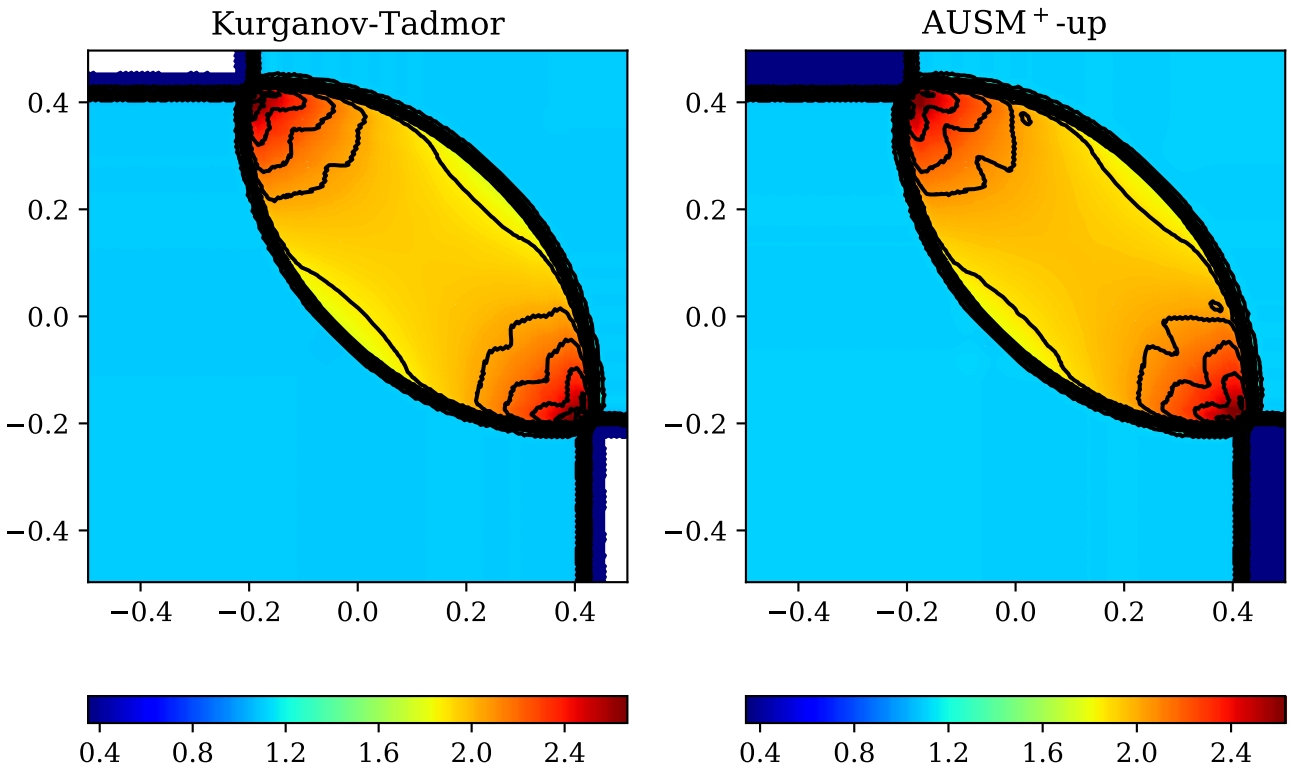


Figure 4.11: Results of Test 10 at $t = 0.25$s. The white patches have shown up in contour for Kurganov-Tadmor scheme due to some unrecognized error in plotting, not due to numerical error.

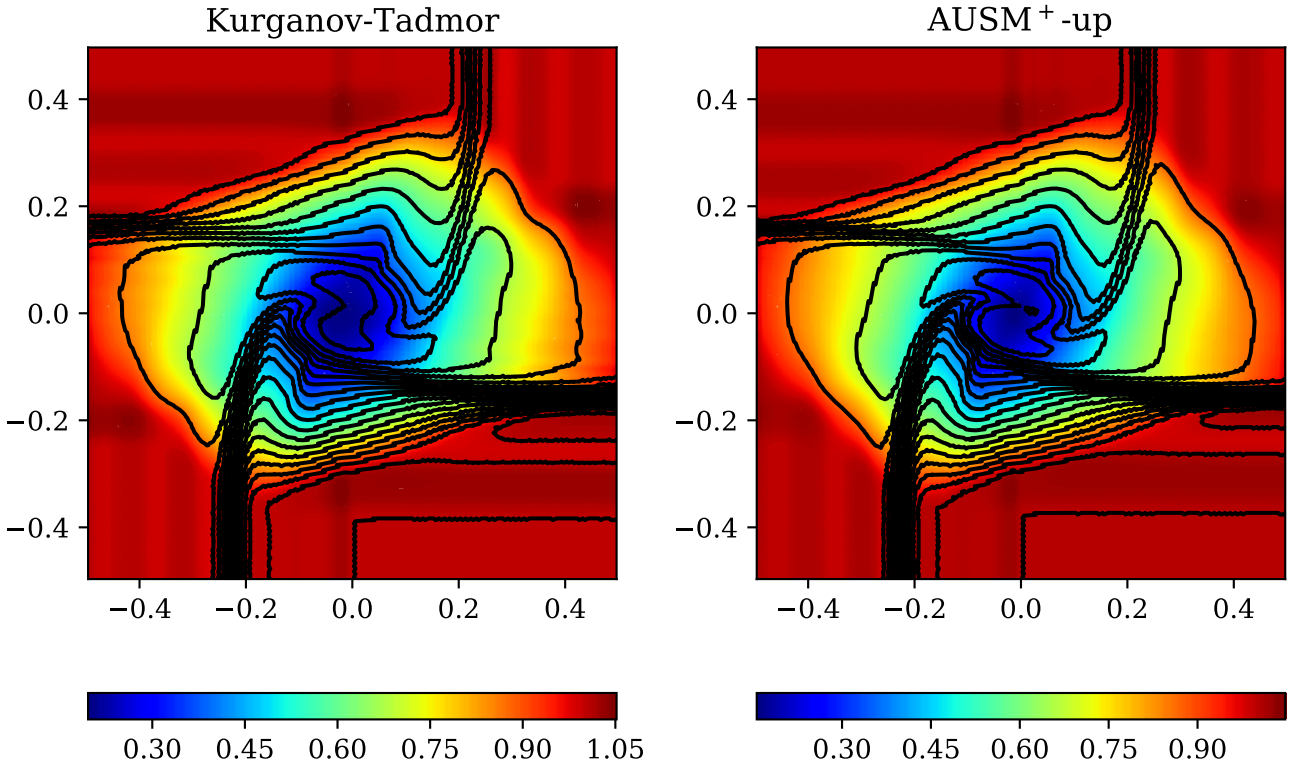


Figure 4.12: Results of Test 11 at $t = 0.3$s.

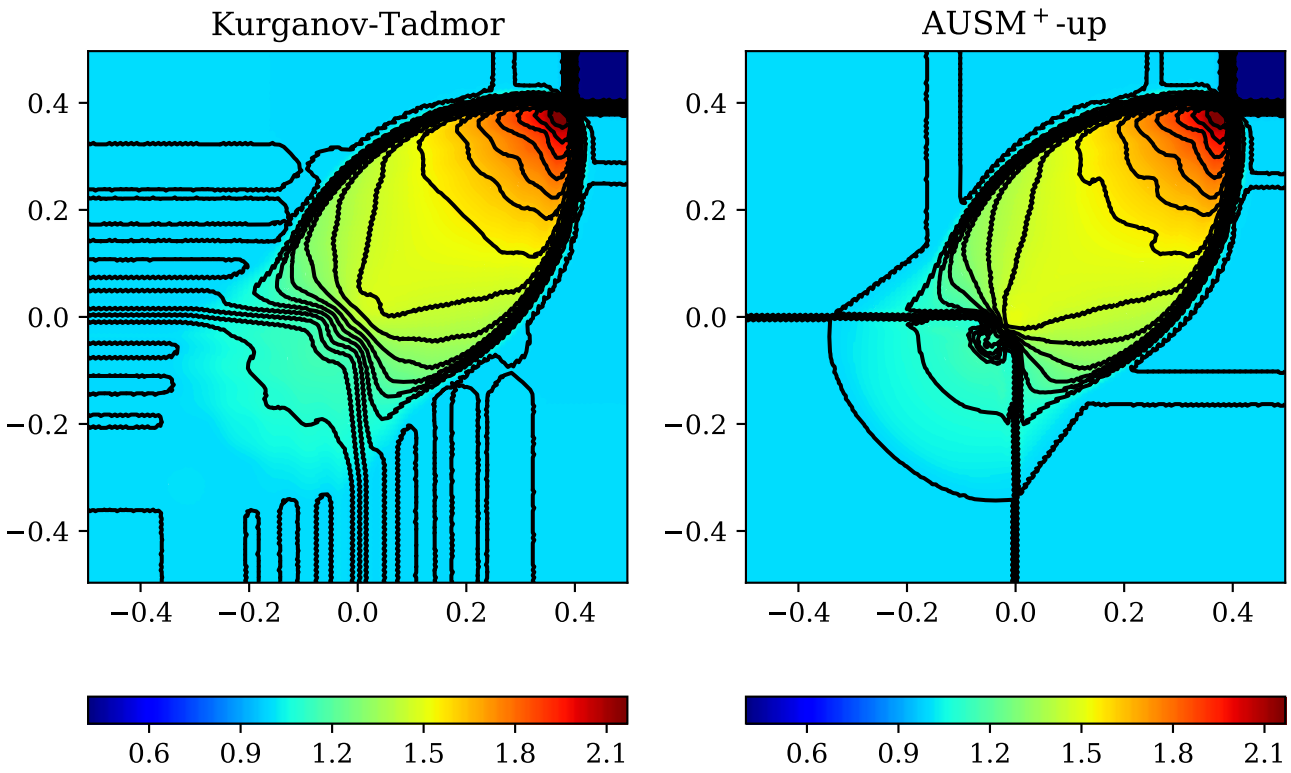


Figure 4.13: Results of Test 12 at $t = 0.25$s.

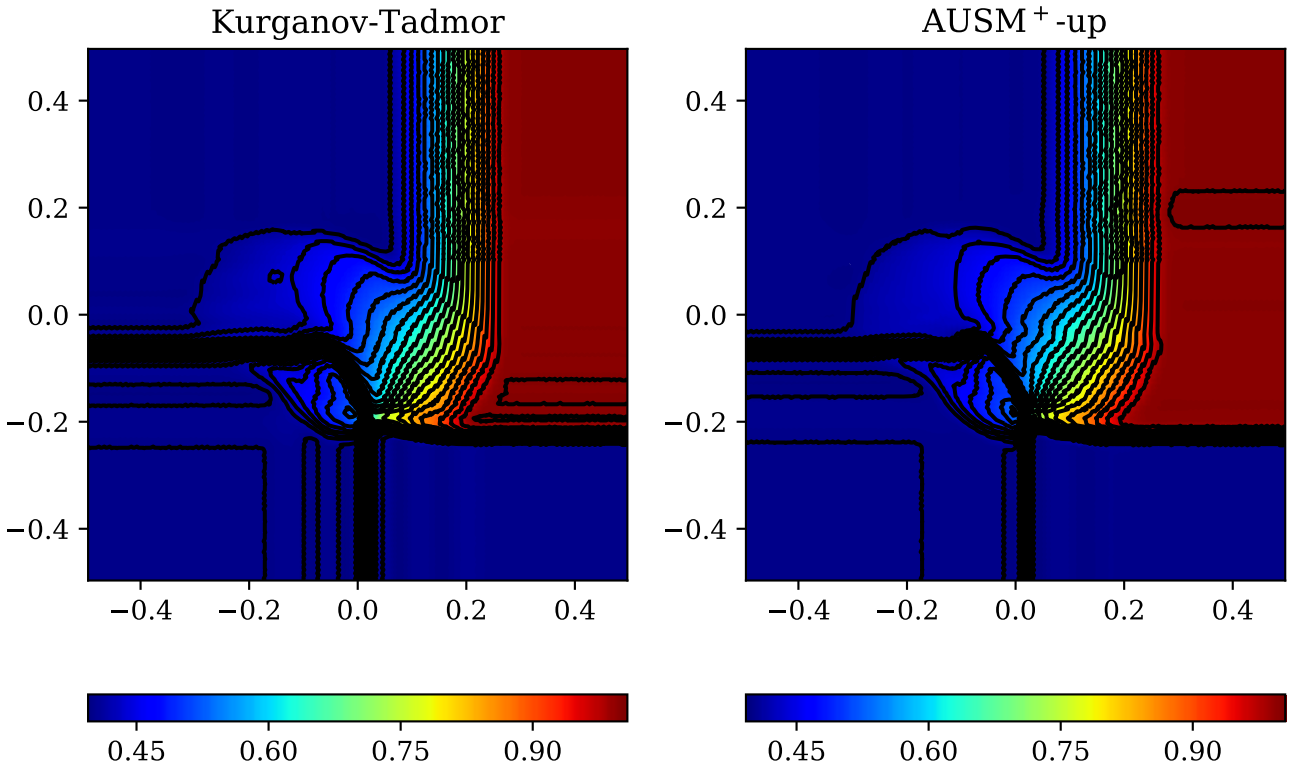


Figure 4.14: Results of Test 13 at $t = 0.2$s.

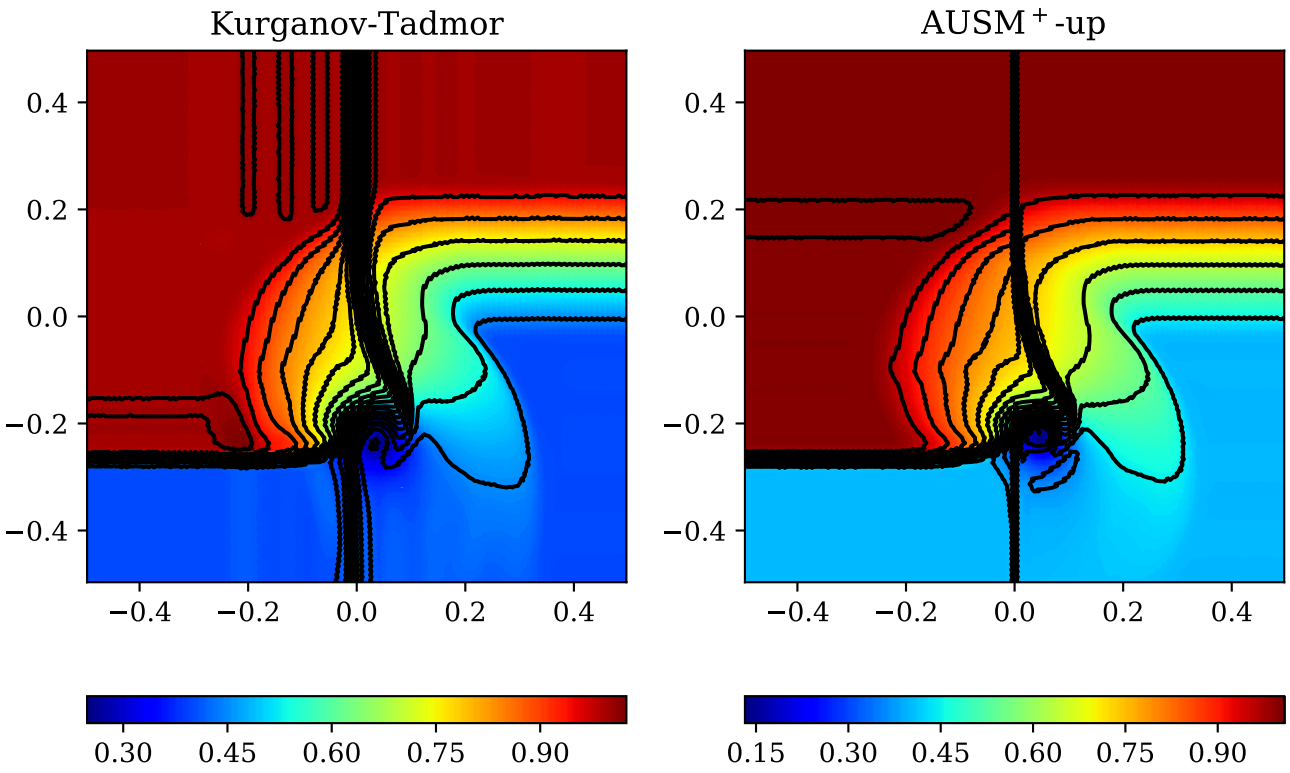


Figure 4.15: Results of Test 14 at $t = 0.3$s.

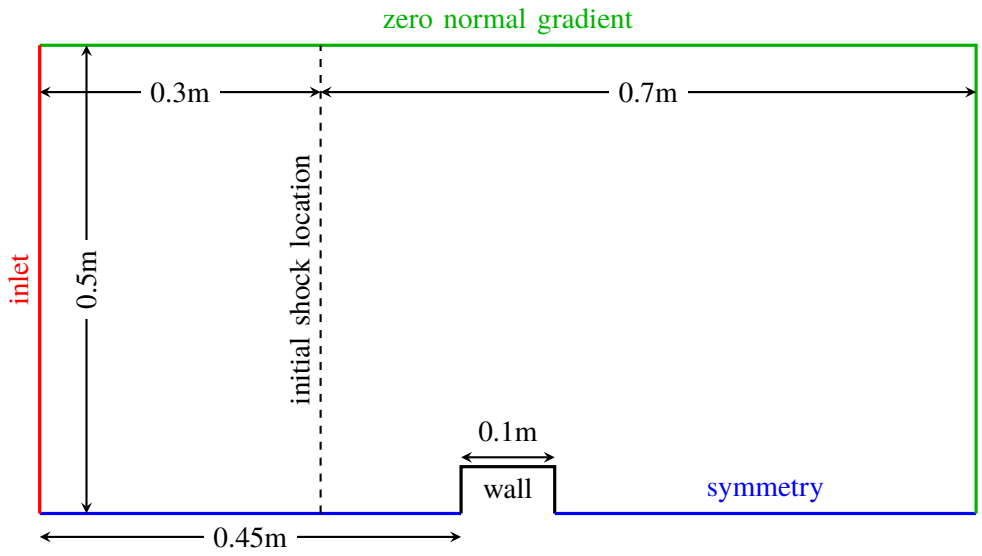


Figure 4.16: Schematic of initial and boundary conditions for square–shock interaction case (Test 15).

finer mesh near cylinder.

The final test case is a Mach number 3 flow over a forward facing step (Test 17). All the details about geometry, initial and boundary conditions are described by Woodward and Colella [9]. The mesh in this case consists of rectangular cells graded to give finer mesh at the step edge. The vertical edge of step and the remaining length in the vertical direction were meshed using 15 and 60 cells respectively. The horizontal dimensions left and right of the step were meshed using 45 and 150 cells respectively.

All the results show 24 density contours on pressure colour map. The run times for these three cases are shown in table Table 4.4.

A point to note at the outset regarding Test 15 is that results shown in Figure 4.17 are with Kurganov-Tadmor and AUSM$^+$-up flux schemes using different interpolation schemes. The AUSM$^+$-up flux scheme didn't work with the preferred vanLeer interpolation scheme. The major challenge in this test is to capture the diffraction of the incident and reflected shocks over sharp square edges. This statement is justified by the observation that AUSM$^+$-up scheme using vanLeer scheme crashed when either reflected shock or the incident shock start diffracting. AUSM$^+$-up scheme was also found to be more sensitive to the diffraction strength, meaning; with vanLeer interpolation scheme, the solver gave expected results for a lower incident shock Mach number of 1.5, while the same combination failed when the incident shock Mach number value was 2. Reducing the Courant number only had the limited effect of slightly increasing the time of solver crash. Kurganov-Tadmor scheme doesn't seem to have these issues.

Results of Test 16 (Figure 4.18) provide more light on the discussion of previous paragraph. For this test, consisting an incident shock of Mach number 2, AUSM$^+$-up flux scheme managed to produce expected results with vanLeer interpolation scheme, probably because the diffraction of both the incident and reflected shocks is over a gradually changing cylinder profile (as opposed to a sharp 90° diffraction in Test 15). As may be expected, Kurganov-Tadmor scheme doesn't have any problem for this case. The results from both the flux schemes are very similar in this case.

Lastly, to strengthen the argument proposed in the previous paragraphs, AUSM$^+$-up scheme wasn't able to simulate Test 17 which involves an even stronger expansion wave, at the start of simulation. Decreasing the Courant number and changing the interpolation scheme were un-fruitful in this case. As expected, the crash point was always the time when expansion wave started forming (~ 0.03s, to be specific). The Kurganov-Tadmor scheme however, produced expected results with vanLeer interpolation scheme and a Courant number of 0.3. It is very clear from Figures 4.19 and 4.20 that the contact discontinuity along the upper wall (which moves very slowly since the flow starts becoming steady) is not captured well enough.

Table 4.4: Execution times (in seconds) for the remaining 2D test cases. The execution time for Test 17 reported here is for an end time of 4 seconds.

Test #	Kurganov-Tadmor	AUSM$^+$-up
15	7	2166
16	17	3964
17	458	—

4.4 A note on execution times

The most noticeable feature from from Tables 4.1, 4.3 and 4.4, is that for 2D test cases, Kurganov-Tadmor scheme is about two orders of magnitude faster than AUSM$^+$-up scheme, although, there is not as much

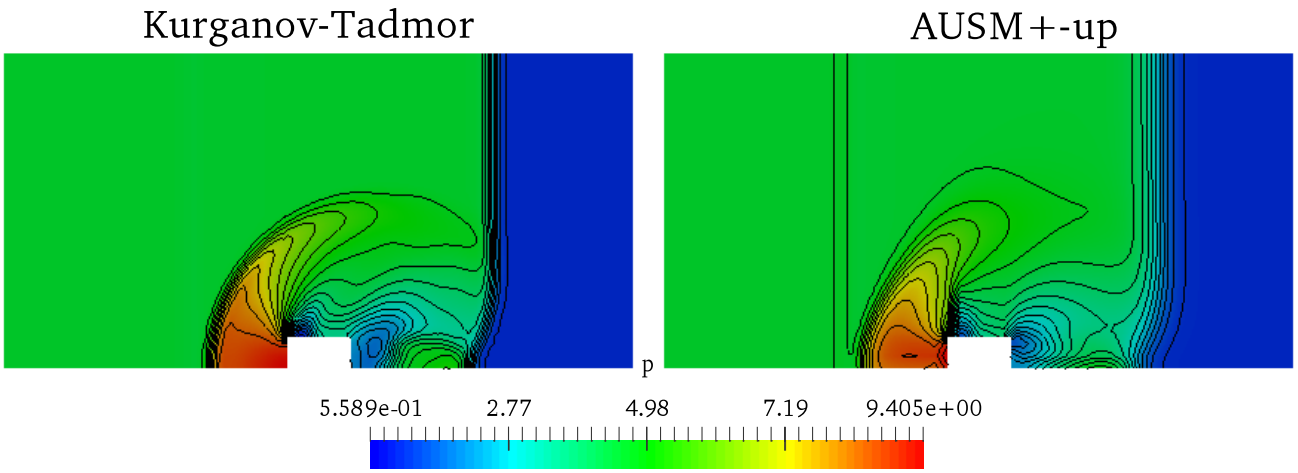


Figure 4.17: Results of Test 15 with Kurganov-Tadmor and AUSM$^+$-up flux schemes using vanLeer and upwind inteprolation schemes respectively, at $t = 0.2$s. Maximum Counant number of 0.3 was used for both schemes.

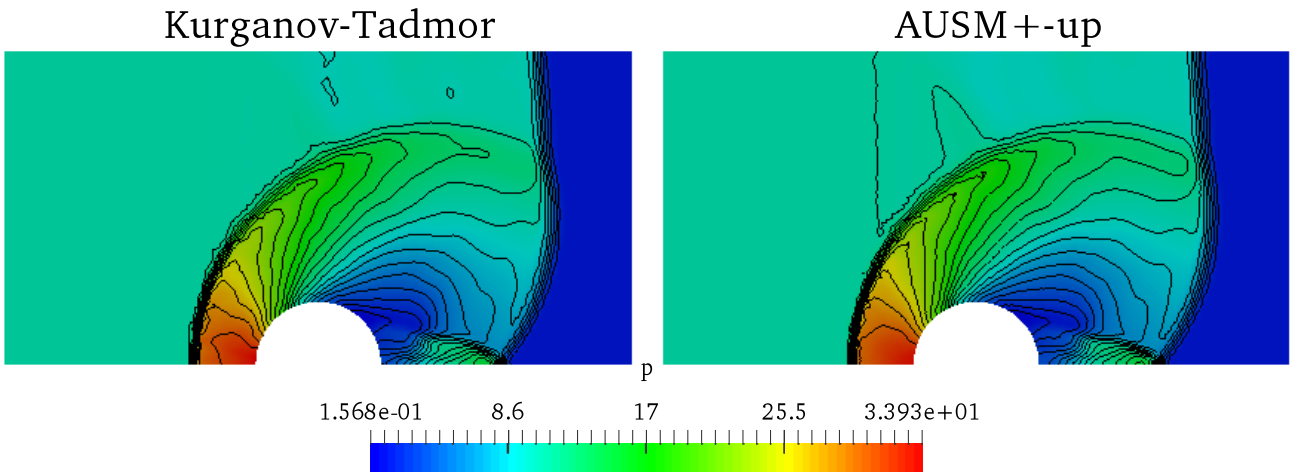
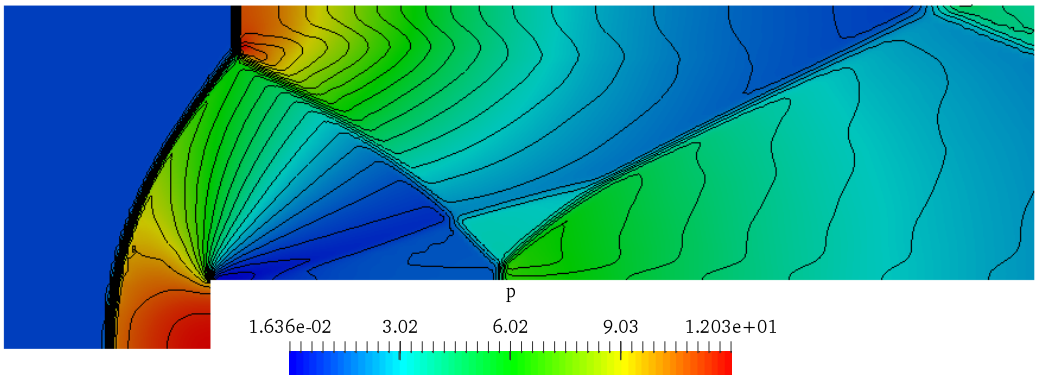
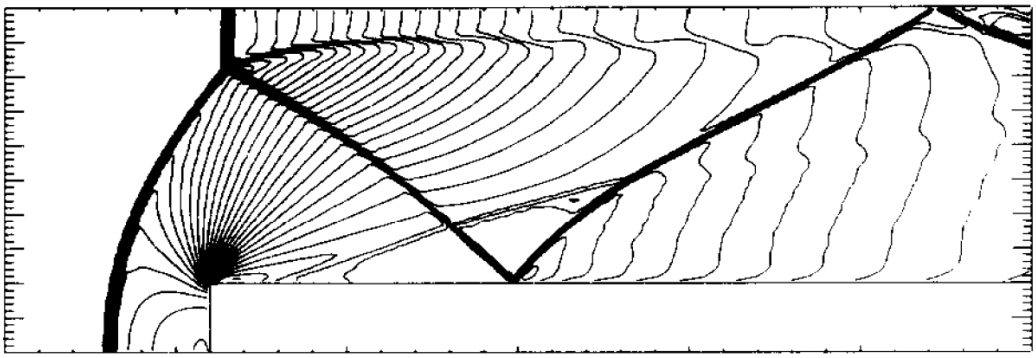


Figure 4.18: Results of Test 16 at $t = 0.17$s. vanLeer interpolation scheme with maximum Courant number of 0.3 was used for both cases.

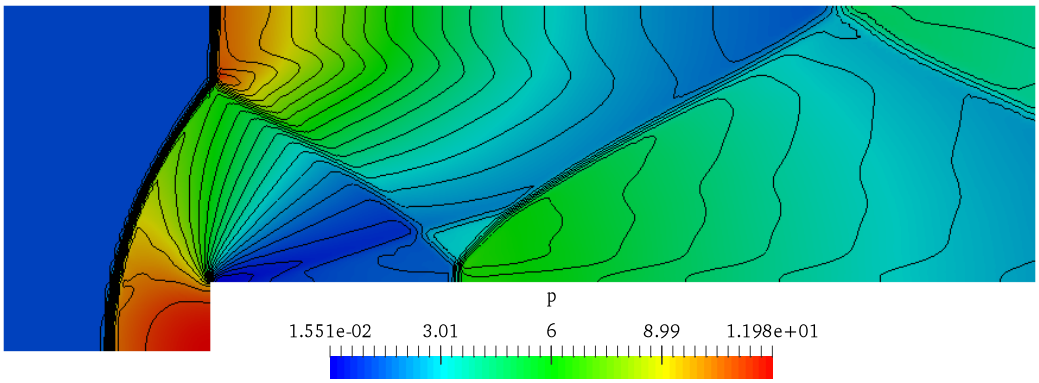


(a) Results of Test 17 with Kurganov-Tadmor scheme at $t = 3$s. vanLeer interpolation scheme with maximum Courant number of 0.3 was used.

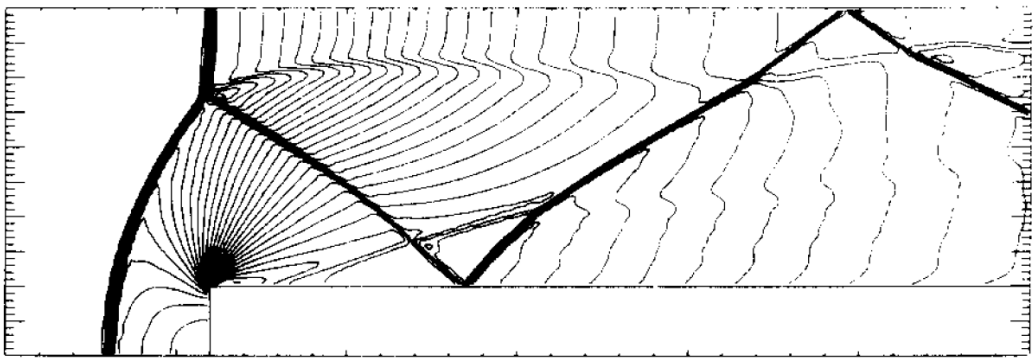


(b) Density contours of Test 17 at $t = 3$s from Woodward and Colella [9].

Figure 4.19: Comparison of results at $t = 3$s with those given by Woodward and Colella [9].



(a) Results of Test 17 with Kurganov-Tadmor scheme at $t = 4$s. vanLeer interpolation scheme with maximum Courant number of 0.3 was used.



(b) Density contours of Test 17 at $t = 4$s from Woodward and Colella [9].

Figure 4.20: Comparison of results at $t = 4$s with those given by Woodward and Colella [9].

disparity in execution time for 1D test cases. However, it will probably be incorrect to conclude that Kurganov-Tadmor scheme actually is $O(100)$ times faster than AUSM$^+$-up scheme purely based on these results. Two main points are to be noted.

Firstly, it's true that AUSM$^+$-up scheme involves many more computations compared to Kurganov-Tadmor scheme. It is due to this very reason that the computational load (and consequently thermal load) on a processor is much more for the former. As thermal loading approaches the threshold, processors tend to slow down. This might be an additional factor to account for when comparing execution times.

Secondly, Kurganov-Tadmor scheme is implemented in a built-in solver of OpenFOAM, while, AUSM$^+$-up was implemented independently, as a user. There could be opportunity for optimizing the code of ausmPlusUpFoamRK3 to achieve higher computational speed. Such work requires better understanding of the way OpenFOAM works, which is out of the scope of this project, and hence has not been considered. This could be another factor to be taken into account before comparison of computational speeds.

Chapter 5

Conclusions and Future Work

With the aim of implementing high resolution schemes in OpenFOAM, 2 new solvers – rhoCentralFoamRK3 and ausmPlusUpFoamRK3 – were made, and tested with several 1D and 2D test cases.

5.1 Remarks on Kurganov-Tadmor and AUSM$^+$-up flux schemes

The most significant shortcoming of the Kurganov-Tadmor flux scheme is it's apparent inability in resolving stationary contact discontinuities and slip lines. This has been observed in one of the 1D test case and in a couple of 2D test cases. AUSM$^+$-up flux scheme on the other hand, performed well compared to Kurganov-Tadmor in resolving such stationary waves. Liou [10] mentions that the common speed of sound employed in Equation (3.11) to calculate left and right Mach numbers serves this very purpose.

The major problem observed with AUSM$^+$-up flux scheme, which was discussed in detail in Section 4.3, is its tendency to crash while simulating strong expansion waves. This behaviour could have been expected considering the fact that AUSM$^+$-up flux scheme slightly under-predicts p and ρ in the low pressure region of Test 2 (Figure 4.3). Liou reports in [5] that AUSM and AUSM$^+$ flux schemes produce pressure oscillations in low Mach number regimes. The correction terms for $M_{1/2}$ and $p_{1/2}$ in Equations 3.17 and 3.12 were proposed to eliminate this behaviour. However, this correction doesn't seem to work for Tests 15 and 17. Kurganov-Tadmor flux scheme was able to resolve the very strong expansion wave that occurred Test 17 (Figures 4.19 and 4.20a) and hence doesn't have such issues.

Apart from these, it is evident from tests 3, 5, 7, 8 and 11 that both the solvers don't capture strong contact discontinuities and slip lines well. It is to an extent uncertain whether such behaviour is inherent to the flux schemes themselves, or whether these emerge when the flux schemes are used along with RK3 time integration.

The computational speed of rhoCentralFoamRK3 is much higher than ausmPlusUpFoamRK3 and thus, it is much more convenient to use rhoCentralFoamRK3 if it is expected before hand – based on the analysis done so far – that rhoCentralFoamRK3 will give reasonable results for the problem considered.

5.2 Future work

The scope of this project has been limited to implementing RK3 time integration in OpenFOAM. There is clearly a chance now, to compare the results of RK3 solvers with the results from conventional solvers of OpenFOAM. One can see how much improvement does RK3 scheme provide compared to the standard

explicit Euler time integration scheme. Once such a comparison is done, the independent behaviour of flux scheme and time integration scheme can be understood.

Considering results of all the test cases together, a different flux scheme which will fill the deficiencies in the Kurganov-Tadmor and AUSM$^+$-up flux schemes is much desirable. It would be preferable to use a flux scheme similar to the Kurganov-Tadmor scheme for this purpose, since, it is very fast and simple with the only shortcoming being inability to capture stationary slip lines well. Addressing the problem of capturing strong expansion by AUSM$^+$-up might require using a different flux scheme from the large family of AUSM schemes.

Since the framework for implementing RK3 in OpenFOAM is now available, it is also possible to implement flux schemes which are entirely different than those considered for this project. An example could be the HLL family of flux schemes.

Bibliography

[1] Eleuterio F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer-Verlag Berlin Heidelberg, 3rd edition, 2009.

[2] Doyle D. Knight. *Elements of Numerical Methods for Compressible Flows*. Cambridge University Press, 2006.

[3] Christopher J. Greenshields. *OpenFOAM User Guide*. OpenFOAM Foundation Ltd, 4.0 edition. `https://openfoam.org/`.

[4] Alexander Kurganov and Eitan Tadmor. New High-Resolution Central Schemes for Nonlinear Conservation Laws and Convection–Diffusion Equations. *Journal of Computational Physics*, 160(1):241–282, May 2000.

[5] Meng-Sing Liou. A sequel to AUSM part II: AUSM$^+$-up for all speeds. *Journal of Computational Physics*, 214(1):137–170, February 2006.

[6] Sigal Gottlieb and Chi-wang Shu. Total Variation Diminishing Runge-Kutta schemes. Technical Report 221, Mathematics of computation of the American Mathematical Society, January 1998.

[7] `https://github.com/vincentcasseau/hyStrath/wiki`. Accessed in September 2017.

[8] Richard Liska and Burton Wendroff. Comparison of several difference schemes on 1D and 2D test problems for the Euler equations. *SIAM Journal on Scientific Computing*, 25(3):995–1017, 2003.

[9] Paul Woodward and Phillip Colella. The numerical simulation of two-dimensional fluid flow with strong shocks. *Journal of computational physics*, 54(1):115–173, 1984.

[10] Meng-Sing Liou. A sequel to AUSM: AUSM$^+$. *Journal of Computational Physics*, 129(2):364–382, July 1996.

[11] Christopher J. Greenshields. *OpenFOAM Programmer's Guide*. OpenFOAM Foundation Ltd, 3.0.1 edition. `https://openfoam.org/`.

Appendix A

Theory

A.1 Total Variation Diminishing schemes

The main desired attribute of any numerical scheme is that it should *converge* to exact solution as grid is refined. To put it in mathematical terms, if q_i^n is defined to be $q\left(x_i, t^n\right)$, then it is desired that

$$\left\|\mathcal{E}^n\right\| \to 0 \text{ as } \Delta x \to 0 \tag{A.1}$$

where, $\mathcal{E}^n$ is the vector of differences between Q_i^n and q_i^n over all points i and $\|\bullet\|$ indicates a norm (could be 1-norm, 2-norm, etc.).

For non-linear systems, to garuntee convergence, schemes have to be conservative, *consistent* and *stable*. Ensuring consistency of a scheme requires that the numerical flux F approaches analytical flux f as the data to left and right of an interface tends to constant value. To ensure stability, the concept of *Total Variation* (TV) is used.

Total Variation of a set of data Q^n is defined as follows.

$$\mathrm{TV}\left(Q^n\right) = \sum_i \left|Q_{i+1}^n - Q_i^n\right| \tag{A.2}$$

A scheme is said to be *Total Variation Bounded* (TVB) if

$$\mathrm{TV}\left(Q^n\right) \leq B \quad \forall \quad n \tag{A.3}$$

where B is some finite number bounding the total variation of data. Furthermore, a scheme is said to be *Total Variation Diminishing* (TVD) if the following relation holds.

$$\mathrm{TV}\left(Q^{n+1}\right) \leq \mathrm{TV}\left(Q^n\right) \quad \forall \quad n \tag{A.4}$$

The TVD property is a property of exact solution to Euler quations. There are theorems which prove that if a conservative and consistent scheme is TVD, it will converge to the exact solution under suitable restriction on time step [1].

Appendix B

OpenFOAM related

B.1 Input-Output(I/O)

I/O in OpenFOAM is mainly through *lists* and *dictionaries*. Listing B.1 shows a snippet of *p* file in the time directory *0* of a case.

Listing B.1: *list* and *dictionary* in OpenFOAM

```
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       volScalarField;
13     location    "0";
14     object      p;
15 }
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18 dimensions      [1 -1 -2 0 0 0 0];
19
20 internalField   nonuniform List<scalar>
21 100
22 (
23 50000
24 50000
25 50000
120 10000
121 10000
122 10000
123 )
124 ;
```

The entire file is a dictionary which in itself contains another dictionary `FoamFile`. In OpenFOAM, a dictionary is a collection of *keywords* and their *values*. In this snippet, there are two keywords when the file is considered as a dictionary: `dimensions` and `internalField`. The value of the keyword `dimensions` is the *dimensionSet* `[1 -1 -2 0 0 0 0]`. These are the powers of the seven S.I. units for mass, length, time, temperature, quantity, current and luminous intensity respectively [3]. These dimensions correspond to the unit of pressure: kg/m-s^2.

The value of keyword `internalField` is a list. Lines 20 and 21 of Listing B.1 mean that the `internalField` is a non-uniform list of size 100, of a scalar. The i^{th} value of the list is the value of pressure in cell i. Lines 8-15 in the listing together are referred to as *header* of the file (which is actually a dictionary) in OpenFOAM terminology. They mean that this file contains data of a *volScalarField* named `p` at the location `0` in the case directory. The Listing B.1 shows only few elements of the list of `internalField`.

B.2 Detailed description of *fvm* and *fvc*

fvm and *fvc* are abbrevations for *Finite Volume Method* and *Finite Volume Calculus* respectively [11]. The functions in *fvc* are meant to perform mathematical operations on existing discrete data. For example, if `T` is a `volScalarField`, then `fvc::grad(T)` would be a `volVectorField`. *fvc* thus performs explicit operations.

On the other hand, *fvm* performs implicit operations. The output of an *fvm* function is not a *field*, but an *fvMatrix*. The *fvMatrix* class stores the coefficient matrix ($\mathbf{A}$), unknown variable ($\boldsymbol{\psi}$) and the source ($\mathbf{b}$) of the set of linear algebraic equations $\mathbf{A}\boldsymbol{\psi} = \mathbf{b}$ produced by discrete implicit operations on a *field*. For example, `fvm::grad(T)` would not produce a `volVectorField` because the value of `T` at the next time step is not available. It produces an *fvMatrix* which contains information of $\mathbf{A}$, unknown $\boldsymbol{\psi}$ (new time step values of `T` in this example) and $\mathbf{b}$ corresponding to the equation $\nabla T = \mathbf{0}$.

B.3 Limited interpolation schemes

Interpolation is done to obtain values of a field on faces using field values at cell centers. Limited interpolation schemes are used as a means to circumvent diffusive nature of lower order schemes and oscillatory nature of higher order schemes. As described in Subsection 3.3.1, higher order schemes require data from atleast two cells. On un-structured grids, it is difficult to implement interpolation schemes requiring more than two cells since there is no sense of directionality. The only two cells, whose data can be used, are the cells which have the face in consideration as a common face i.e.; the owner and neighbour cells of a face.

OpenFOAM by default assumes any grid to be un-structured and three dimensional. Hence, interpolation schemes, at most, are of 2^{nd} order in OpenFOAM. Let ϕ represent a scalar field defined at cell centers. Let subscripts f, P and N represent values at the face, owner cell and neighbour cell respectively. A limited interpolation scheme can be written as

$$\phi_f = \phi_f^U + \beta(r)\left(\phi_f^H - \phi_f^U\right) \tag{B.1}$$

where, superscripts U and H represent interpolated values using upwind and higher order interpolation schemes respectively. β, the limiter, in a sense is a sensor which switches between upwind and higher order scheme depending on flow gradient at a face captured by r. As already mentioned in Subsection 3.3.1, H corresponds to linear interpolation in OpenFOAM.

In OpenFOAM, interpolated value is calculated using *weights*. Each face has an associated weight (w_f) and the interpolated value is computed using the formula given below.

$$\phi_f = w_f\phi_P + \left(1 - w_f\right)\phi_N \tag{B.2}$$

The weights w_f for a limited interpolation scheme are calculated using the values of ϕ at cell centers and flux $\mathcal{F}$ specified at faces to determine upwind direction. Let $\mathbf{C}_P$ and $\mathbf{C}_N$ represent co-ordinates of cell centers of owner and neighbour cells. Let $\mathbf{C}_f$ represent co-ordinate of face center. See Figure B.1 for illustration.

1. The gradient parameter at a face r_f is calculated as

$$r_f = 2\left(\frac{\Delta\phi_{uf}}{\Delta\phi_f}\right) - 1, \tag{B.3}$$

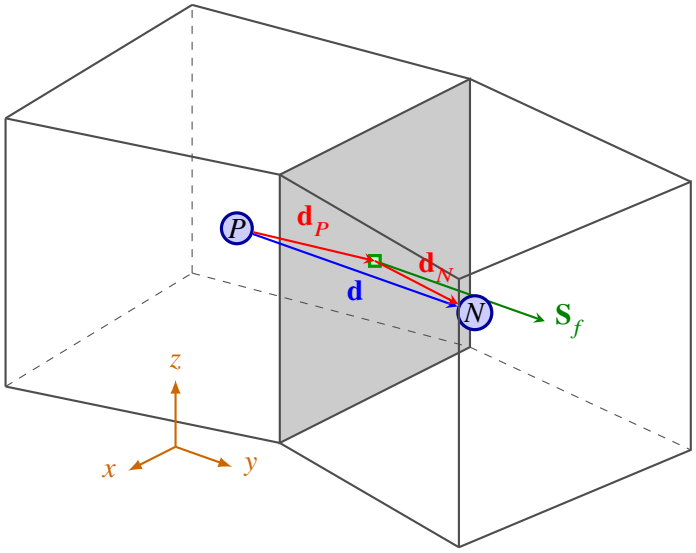


Figure B.1: Close up view of a face. P represents the owner cell and N represents the neighbour cell.

where,

$$\begin{aligned} \Delta\phi_f &= \phi_N - \phi_P \\ \Delta\phi_{uf} &= \begin{cases} \mathbf{d}\cdot\nabla\phi_P & \text{for } \mathcal{F} > 0 \\ \mathbf{d}\cdot\nabla\phi_N & \text{for } \mathcal{F} < 0 \end{cases} \\ \mathbf{d} &= \mathbf{C}_N - \mathbf{C}_P \end{aligned} \tag{B.4}$$

Here ∇ represents the gradient operator. By default, gradient is calculated by central differencing in OpenFOAM (refer Appendix section B.4 for more details).

2. The central difference (linear interpolation) weights (L_f) are calculated as follows.

$$L_f = \frac{\mathbf{S}_f \cdot \mathbf{d}_N}{\mathbf{S}_f \cdot \left(\mathbf{d}_N + \mathbf{d}_P\right)} \tag{B.5}$$

Where, $\mathbf{S}_f$ is the face area vector, $\mathbf{d}_N = \mathbf{C}_N - \mathbf{C}_f$ and $\mathbf{d}_P = \mathbf{C}_f - \mathbf{C}_P$. It is worthwhile noting that in OpenFOAM, $\mathbf{S}_f$ points from the owner cell to neighbour cell (as shown in Figure B.1). As a result, any flux is positive if the corresponding flux vector points from owner to neighbour and negative otherwise.

3. Value of limiter at a face (β_f) is calculated using any one of the limiter functions. For example, the vanLeer limiter function reads as follows.

$$\beta_f = \frac{r_f + \left|r_f\right|}{1 + \left|r_f\right|} \tag{B.6}$$

4. Limited interpolation weights (w_f) are calculated as follows.

$$w_f = \beta_f L_f + \left(1 - \beta_f\right)\text{pos}(\mathcal{F}) \tag{B.7}$$

Where,

$$\text{pos}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

It can be proved by simple algebra that using Equation (B.7), the expression for ϕ_f can be written as follows.

$$\phi_f = \begin{cases} \phi_P + \beta_f \left[L_f \phi_P + \left(1 - L_f\right) \phi_N - \phi_P \right] & \text{if } \mathcal{F} \geq 0 \\ \phi_N + \beta_f \left[L_f \phi_P + \left(1 - L_f\right) \phi_N - \phi_N \right] & \text{otherwise} \end{cases}$$

Notice the similarity of above equation with Equation (B.1). The first two terms within the square brackets represent linear interpolated face value (compare with Equation (B.2)). Also, $w_f = \text{pos}(\mathcal{F})$ is the weight for upwind interpolation, and Equation (B.7) follows from Equation (B.1) due to this fact.

B.4 Gradient calculation

Let ϕ be any volume field for which gradient is to be calculated. In OpenFOAM, gradient is calculated in two steps.

1. Interpolating ϕ to obtain it's values on faces
2. Using interpolated values to calculate gradient

Step 1 is done using linear interpolation (as described in Appendix section B.3) by default. For step two, there are choices available for the way gradient is estimated from the face interpolated values. The widely used method is the *Gauss* method, described below. Let ϕ_f be interpolated value of ϕ on a representative face f of a representative cell P. Gradient is approximated as

$$(\nabla \phi)_P = \frac{\sum\limits_{\text{own}} \phi_f \mathbf{S}_f - \sum\limits_{\text{nei}} \phi_f \mathbf{S}_f}{\Delta V_P}, \tag{B.8}$$

where, ΔV is the volume of the cell P, $\sum\limits_{\text{own}}$ represents summation over faces for which cell P is owner and $\sum\limits_{\text{nei}}$ represents summation over faces for which cell P is neighbour. Note that the numerator in Equation (B.8) is a vector sum.

B.5 Courant Number calculation

Courant Number is a number which indicates the extent of wave propagation at the interfaces of the domain. There are many interpretations of the physical meaning of this number. One such interpretation is that the value of Courant Number has to be below certain limit to ensure that Riemann solution at a given face is unaffected by Riemann solution of neighbouring faces.

There are many ways of calculating Courant Number in OpenFOAM. The commonly used method in compressible flow solvers is to calculate the *Central Courant Number*. Let ψ_f be the estimate of maximum mass flux per density at a representative face f of a representative cell P. Then the maximum and mean values of central Courant Number – C and $\overline{C}$ respectively – are calculated as follows.

$$C = \left(\frac{1}{2}\right) \max_P \left\{ \frac{\sum\limits_f \psi_f \left\| \mathbf{S}_f \right\|_2}{\Delta V_P} \right\} \cdot \Delta t, \qquad \overline{C} = \left(\frac{1}{2}\right) \frac{\sum\limits_P \left(\sum\limits_f \psi_f \left\| \mathbf{S}_f \right\|_2 \right)}{\sum\limits_P \Delta V_P} \cdot \Delta t \tag{B.9}$$

Here, $\mathbf{S}_f$ is the face area vector, $\|\bullet\|_2$ represents Euclidean norm and ΔV_P is the volume of cell P. Different flux schemes use different ways of defining ψ_f. For example, a reasonable choice might be to use the

maximum absolute value of the eigen values at a face.

$$\psi_f = \frac{\left|\mathbf{U}_f \cdot \mathbf{S}_f\right|}{\left\|\mathbf{S}_f\right\|_2} + \left|a_f\right|$$

Where, $\mathbf{U}_f$ is the velocity vector at a face and a_f is the speed of sound at a face. These values could be obtained by interpolation.

B.6 Few problems encountered

Firstly, the default way of time stepping in OpenFOAM is to use `solve`. For example,

Listing B.2: Usage of `solve`

```
1 solve(fvm::ddt(q) + fvc::div(f) = 0);
```

updates `q` based on time scheme chosen and $\nabla \cdot \mathbf{f}$. Initial approach to implement RK3 was to specify `Euler` time scheme and use the `solve` statement 3 times, similar to Listing B.2, by updating the flux $\mathbf{f}$ in between the 3 updations.

Listing B.3: Updating normally, without using `solve`

```
1 q = q - dt*fvc::div(f);
```

But, this approach failed to give results. Simulation using such solvers crashed immediately for 1D tests even with maximum Courant number as low as 0.01. This could probably be due to messing up of OpenFOAM's inbuilt storage style. OpenFOAM can store references to a field for previous and pre-previous times. Each `solve` updates these references according to time scheme chosen. As this whole mechanism was not properly understood, after the problem was encountered, update was done normally as shown in Listing B.3. This algorithm worked. The time step, `dt`, can be obtained from `runTime` object.

The second problem is related to parallel simulations in OpenFOAM. It has been realized while running the square–shock interaction case in parallel, that solutions had oscillations exactly at the locations where domain was split, even though ideally they shouldn't have occurred. The reason for this is unknown. Probably, there must be some additional accounting done in the solvers for running in parallel, or, this could be a problem with parallel simulation in OpenFOAM itself. To avoid this problem, all the cases had to be run using coarse mesh on a single core processor.