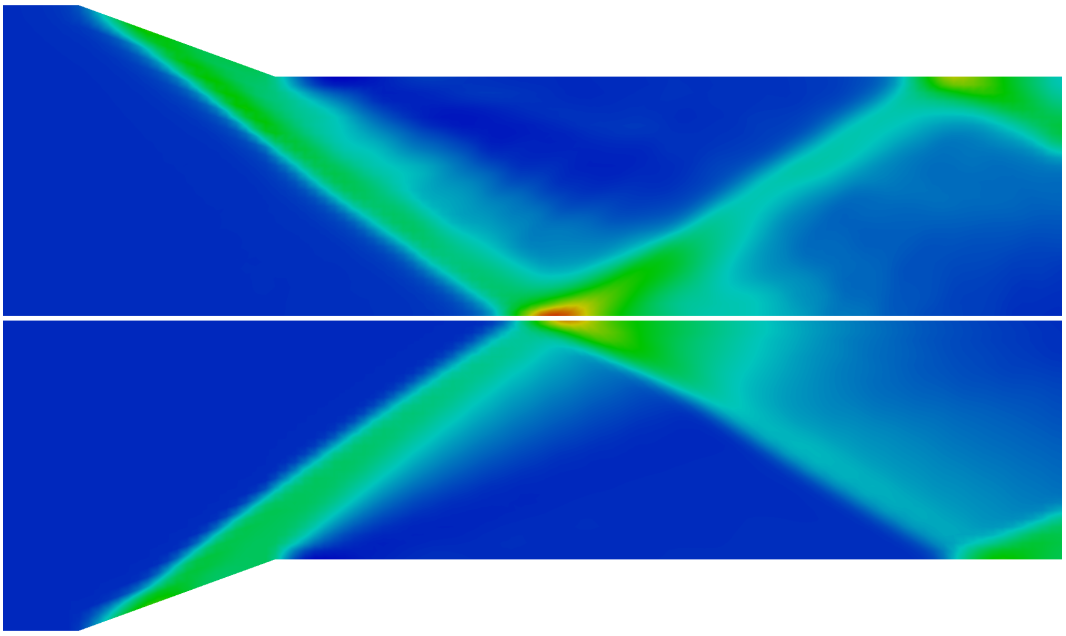
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Comparative study of density-based versus pressure-based solvers for supersonic flow



Bachelor Thesis

Sean Bone

Spring semester 2020

Supervisor: Prof. Dr. P. Jenny

Institute for Fluid Dynamics, ETH Zürich

Abstract

In this thesis a comparison is made between pressure- and density-based solvers, with the goal of determining which is best suited to simulations of the flight performance of a supersonic sounding rocket. Although there exists plenty of work on each of the two approaches to solving compressible flows, there appears to be little in the way of direct comparison between them.

In the following, the theory behind each approach is investigated, as well as its implementation in two solvers from the OpenFOAM v4.0 software package: rhoPimpleFoam and rhoCentralFoam. These two solvers are then also compared in terms of numerical results in two separate test cases: an oblique shock generated by an inclined wedge geometry, and Sod's shock tube.

Contents

Chapter 1

Introduction

Project EULER is an initiative by ARIS (Akademische Raumfahrt Initiative Schweiz), a student group at ETH Zürich, to design, build and launch a sounding rocket to an altitude of 30,000 feet (9,144 metres). Given the high target altitude, EULER will need to travel at supersonic speeds: this presents an entirely new set of challenges for the ARIS team.

From a simulations standpoint, one question immediately presents itself: which CFD solver is best suited to the task of simulating the rocket's flight? Of particular interest are the resulting values for drag coefficients as well as force and temperature distributions over the rocket's body during each flight phase.

The software of choice is OpenFOAM, a widely recognized and well-tested CFD toolbox suited to a large variety of applications. Its open-source nature means it is possible for us to investigate the implementation of its solvers, allowing us to accurately determine what algorithms they are using, even when the documentation is lacking.

The wide range of Mach numbers we are interested in studying (from 0.3 to 2) already tells us we are looking for a *compressible* solver. This already limits our choice to the following three solvers [7]:

- *rhoSimpleFoam* is a steady-state solver designed to resolve compressible flows. It implements the popular SIMPLE algorithm. It is a pressure-based solver.
- *rhoPimpleFoam* is a transient solver designed to resolve compressible flows. It implements the PIMPLE algorithm unique to OpenFOAM and presented above.

Like rhoSimpleFoam, it is a pressure-based solver.

- *rhoCentralFoam* is also a transient and compressible solver, but it is density-based in its approach. It implements the central-upwind schemes of Kurganov and Tadmor.

We are interested in both transient and steady-state phenomena, therefore we will be focusing our attention on the latter two solvers, since rhoSimpleFoam is only suitable for steady-state problems. In all comparisons performed in this work, version 4.0 of the OpenFOAM toolbox is used.

The two solvers are very different in nature: rhoPimpleFoam is a *pressure*-based solver, whereas rhoCentralFoam is a *density*-based solver. In this work we investigate the differences between these two approaches: first a theoretical overview of the algorithms driving each solver is given, and then this is compared to the actual implementation in OpenFOAM v4.0 code.

Following this, two test cases are proposed to compare the actual performance of each solver: an oblique shock produced by an inclined wedge, and Sod's shock tube. Both of these problems have well-known analytical solutions making them well suited to a comparison of numerical results.

Chapter 2

The pressure-based approach

2.1 The SIMPLE algorithm

The first pressure-based algorithm we will consider is SIMPLE (for Semi-Implicit Method for Pressure Linked Equations). Introduced by Caretto et al in 1972 [2], it is a steady-state solver for compressible or incompressible flows. It has seen many variants over the years, making it one of the most commonly studied algorithms in computational fluid dynamics.

This algorithm is implemented by the rhoSimpleFoam solver in OpenFOAM. Although this particular solver is not the main focus of this work, understanding the theory and implementation behind it will serve as a stepping stone towards rhoPimpleFoam.

The equations to be solved are expressed by the following system of differential equations, which express the physical laws of conservation of mass (continuity), conservation of momentum and conservation of energy:

$$\nabla \cdot (\rho U) = 0, \tag{2.1}$$

$$\nabla \cdot (\rho U \otimes U) + \nabla p - \nabla \cdot \sigma = S, \tag{2.2}$$

$$\nabla \cdot (UE + Up) - \nabla \cdot \dot{q} - \nabla \cdot (U\sigma) = Q, \tag{2.3}$$

where σ is the stress tensor which is related in some way to the velocity field and S, Q are source terms for momentum and energy respectively. Finally, $\dot{q}$ is the diffusive flux of energy, which is in some way related to the energy E.

Note that time-derivative terms have been dropped, since SIMPLE is a solver oriented at steady-state problems.

We can reformulate the momentum equation 2.2 as:

$$\underbrace{\nabla \cdot (\rho U \otimes U) - \nabla \cdot \sigma - S}_{\text{Discretised as } \sum_n A_n U_n - S} = -\nabla p$$

By introducing the operator $H(U) = -\sum_{n,n\neq p} A_n U_n - s$, where n indexes neighboring cells, we obtain a semi-discretised form of the momentum equation for a cell p:

$$A_p U_p = H(U) - \nabla p \tag{2.4}$$

$$U_p^* = A_p^{-1} H(U) - A_p^{-1} \nabla p. \tag{2.5}$$

This is still not enough to solve the system of equations 2.1–2.2 however, since we do not have an explicit formulation for p. On top of that, there is no guarantee that the velocity field U^* calculated in this manner actually conforms to the continuity equation 2.1. This highlights the need for a *corrector* step to adjust the velocity field in accordance with conservation of mass.

We multiply equation 2.5 by ρ and take its divergence:

$$\nabla \cdot (\rho U_p^*) = \nabla \cdot (\rho A_p^{-1} H(U)) - \nabla \cdot (\rho A_p^{-1} \nabla p)$$

By applying the continuity equation 2.1, we obtain the following *pressure equation*:

$$\nabla \cdot \left(\rho A_p^{-1} \nabla p' \right) = \nabla \cdot \left(\rho A_p^{-1} H(U^*) \right). \tag{2.6}$$

Here U^* is the initial guess for the velocity field, computed with equation 2.5. Solving equation 2.6 yields us the updated pressure field p', which we can now feed back into equation 2.5 to obtain a corrected velocity field:

$$U_p' = A_p^{-1} H(U^*) - A_p^{-1} \nabla p'. \tag{2.7}$$

It is worth noting that the pressure equation takes the form of a Poisson equation, for which there are many well-known numerical solvers.

The SIMPLE algorithm in OpenFOAM can now be written as:

1. Set the boundary conditions.
2. Solve the momentum equation 2.2 to compute the intermediate velocity field.
3. Compute the mass fluxes ρU_f at the cell faces by interpolation.
4. Solve the pressure equation 2.6 and apply under-relaxation.
5. Correct the mass fluxes at the cell faces.
6. Correct the velocities based on the new pressure field by means of equation 2.7.
7. Update the boundary conditions.
8. Repeat until convergence.

2.2 Implementation of the rhoSimpleFoam solver

Listing 2.1 shows a simplified version of the main file in OpenFOAM's implementation of the compressible SIMPLE algorithm, rhoSimpleFoam. The SIMPLE loop matches the overall structure of the algorithm derived above: the file `UEqn.H` solves the momentum equation, and `pEqn.H` solves the pressure equation. Note that there is an alternative version of the pressure equation, `pcEqn.H`, which implements the SIMPLE-C (SIMPLE-Consistent) algorithm, a variant of SIMPLE.

```
while (simple.loop())
{
    // Pressure-velocity SIMPLE corrector
    #include "UEqn.H"
    #include "EEqn.H"

    if (simple.consistent())
        #include "pcEqn.H"
    else
        #include "pEqn.H"

    turbulence->correct();
}
```

Listing 2.1: The main loop of rhoSimpleFoam

2.2.1 The momentum equation

Listing 2.2 shows part of the contents of `UEqn.H`, where the momentum equation 2.2 is solved.

```
tmp<fvVectorMatrix> tUEqn
(
    fvm::div(phi, U)
    + MRF.DDt(rho, U)
    + turbulence->divDevRhoReff(U)
    ==
    fvOptions(rho, U)
);
fvVectorMatrix& UEqn = tUEqn.ref();

[...]

// Solve the Momentum equation
solve(UEqn == -fvc::grad(p));
```

Listing 2.2: The momentum equation in rhoSimpleFoam

The momentum equation can be reconstructed from the code in the following manner:

- `fvm::div(phi, U)` corresponds to $\nabla \cdot (\rho U \otimes U)$, since in OpenFOAM `phi` generally refers to face mass flux over cell faces, and is calculated as $\rho \cdot U$.

- `MRF.DDt(rho , U)` refers to a rate of change induced by a moving mesh (MRF stands for Multiple Reference Frame), and can be neglected if using a stationary mesh.
- `turbulence->divDevRhoReff(U)` is the turbulence term, and depends on what turbulence model is being used. It equates to the term $-\nabla \cdot \sigma$ in the original equation, and is typically approximated via the Reynolds-Averaged Navier-Stokes equations (RANS).
- `fvOptions(rho, U)` is used by OpenFOAM to add sources/sinks or impose constraints and numerical corrections.
- Finally, `-fvc::grad(p)` corresponds to the gradient of the pressure field: $-\nabla p$. It is calculated *explicitly* with Finite Volume Calculus, represented by the `fvc::` prefix, and is based on the pressure field at the beginning of the time step.

Hence the velocity equation used in the first step of the SIMPLE algorithm is:

$$\nabla \cdot (\rho U \otimes U) - \nabla \cdot \sigma = S - \nabla p$$

2.2.2 The pressure equation

OpenFOAM's implementation of the pressure equation essentially has two versions, depending on whether the "transonic" option is enabled (`simple.transonic()`). For simplicity in this section, we will focus on the case where this option is disabled.

The most important part we focus our attention on is the definition of the pressure equation:

```
volScalarField rAU(1.0/UEqn.A());
surfaceScalarField rhorAUf("rhorAUf", fvc::interpolate(rho*rAU));
volVectorField HbyA(constrainHbyA(rAU*UEqn.H(), U, p));

[...]

surfaceScalarField phiHbyA("phiHbyA", fvc::flux(rho*HbyA));

[...]
// --- Define the pressure equation
fvScalarMatrix pEqn
(
    fvc::div(phiHbyA)
  - fvm::laplacian(rhorAUf, p)
  ==
    fvOptions(psi, p, rho.name())
);
```

Listing 2.3: The pressure equation in rhoSimpleFoam

- The term `fvc::div(phiHbyA)` corresponds to the term $\sum_f S_f \cdot \left(\rho A_p^{-1} H(U)\right)_f$. `phiHbyA` is calculated as `fvc::flux(rho*HbyA)`, where `fvc::flux` calculates the face-flux of its argument (namely $S_f \left(\cdot\right)_f$), and in turn `rho*HbyA` translates to $\rho A_p^{-1} H(U)$.
- The term `fvm::laplacian(rhorAUf, p)` corresponds to $\nabla \cdot \left(\rho A_p^{-1} \nabla p\right)$. The variable `rhorAUf` equates to ρA_p^{-1} and is computed by interpolation on the cell faces, since p is stored on a staggered grid.
- `fvOptions(psi, p, rho.name())` adds sources and sinks, as in the momentum equation.

It becomes clear that this equates to solving the pressure equation 2.6 derived above:

$$\nabla \cdot \left(\rho A_p^{-1} \nabla p\right) - \sum_f \vec{S}_f \cdot \left(\rho A_p^{-1} H(U)\right)_f = 0.$$

Finally, having solved the above equation, the velocity field is updated according to the discretised momentum equation 2.5:

```
// --- Correct the velocity field
U = HbyA - rAU*fvc::grad(p);
```

2.3 The PISO algorithm

The PISO (for Pressure Implicit with Splitting of Operators) algorithm, first introduced by R. I. Issa in 1982 [4], is a non-iterative method for solving the coupled equations arising from implicitly discretising the time-dependent fluid flow equations. It can be applied to both compressible and incompressible versions of the transport equations, however OpenFOAM's implementation `pisoFoam` only tackles incompressible cases. A compressible variant of the algorithm is implemented in the `rhoPimpleFoam` solver, which implements the PIMPLE algorithm discussed later in this chapter.

We are now considering the more complete time-dependent form of the laws of conservation of mass (continuity), conservation of momentum and conservation of energy:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) = 0, \tag{2.8}$$

$$\frac{\partial}{\partial t}(\rho U) + \nabla \cdot (\rho U \otimes U) + \nabla p - \nabla \cdot \sigma = S, \tag{2.9}$$

$$\frac{\partial}{\partial t} E + \nabla \cdot (UE + Up) - \nabla \cdot \dot{q} - \nabla \cdot (U\sigma) = Q. \tag{2.10}$$

The algorithm itself is rather similar to the SIMPLE algorithm presented above. In fact it could be viewed as a transient variant of SIMPLE, with the important modification that the momentum corrector step is performed more than once at each time step.

We derive the momentum predictor equation in the same manner as for the SIMPLE algorithm:

$$U_p^* = A_p^{-1} H(U^n) - A_p^{-1} \nabla p^n. \tag{2.11}$$

Solving this equation will yield an initial prediction U^* for the velocity field at time t^{n+1} based on the velocity and pressure fields at time t^n. It should be noted that although the PISO algorithm is transient, the additional terms owed to the time-dependence of the problem only affect the A and H operators; the overall structure of the algorithm is unchanged.

We now use the same Poisson equation as SIMPLE to compute the pressure field p^*:

$$\nabla \cdot \left(A_p^{-1} \nabla p^* \right) = \nabla \cdot \left(A_p^{-1} H(U^*) \right). \tag{2.12}$$

As before, we can now feed this pressure field back into the momentum equation 2.11 to obtain a corrected velocity field:

$$U_p^{**} = A_p^{-1} H(U^*) - A_p^{-1} \nabla p^*. \tag{2.13}$$

Where SIMPLE would now proceed to the next iteration, PISO instead repeats these last two steps:

$$\nabla \cdot \left(A_p^{-1} \nabla p^{**} \right) = \nabla \cdot \left(A_p^{-1} H(U^{**}) \right), \tag{2.14}$$

$$U_p^{***} = A_p^{-1} H(U^{**}) - A_p^{-1} \nabla p^{**}. \tag{2.15}$$

This pressure corrector loop is repeated a set number of times before continuing to the next time-step. Issa [4] states that if a second order accurate time stepping scheme is used, then three corrector steps should be performed in order to reduce the discretization error due to PISO to second order.

2.4 The PIMPLE algorithm in OpenFOAM

The pisoFoam solver in OpenFOAM implements the PISO algorithm; however, this solver is incompressible only. The compressible variant of the algorithm is not directly present in OpenFOAM, which offers the rhoPimpleFoam solver, which implements the PIMPLE algorithm [7].

PIMPLE is a combination of the PISO and SIMPLE algorithms. It can be seen as applying the SIMPLE algorithm at each time step, iterating until either convergence is reached or a maximum number of iterations is reached. Within each iteration, the pressure corrector is applied iteratively as well, much like in the PISO algorithm. The advantage of PIMPLE over PISO is increased stability, even with Courant numbers above 1.

We can write up the PIMPLE algorithm as:

1. Repeat for each time step:
    a) Repeat `nOuterCorrections` times, or until convergence:
        i. Solve the momentum equation 2.11.
        ii. Repeat `nInnerCorrections` times:
            A. Solve equation 2.12 to obtain the pressure field.
            B. Solve equation 2.13 to obtain the corrected velocity field.

Listing 2.4 shows a simplified version of OpenFOAM's implementation of the PIMPLE algorithm, rhoPimpleFoam. It is easy to see how the overall structure matches that which is presented above.

```
while (runTime.run())
{
    // --- Pressure-velocity PIMPLE corrector loop
    while (pimple.loop())
    {
        #include "UEqn.H"
        #include "EEqn.H"

        // --- Pressure corrector loop
        while (pimple.correct())
            #include "pEqn.H"
    }
}
```

Listing 2.4: The time loop of rhoPimpleFoam

The contents of both `UEqn.H` and `pEqn.H` will not be presented here, as they are extremely similar to those used in the rhoSimpleFoam solver and presented earlier; the only addition being the time-dependent terms that were added in equations 2.8 and 2.9.

Chapter 3

The density-based approach

A density-based approach tackles the same problem of compressible flows from a rather different angle. For simplicity, let us consider the case of the one-dimensional Euler equations:

$$\frac{\partial}{\partial t}\rho + \frac{\partial}{\partial x}(\rho u) = 0, \tag{3.1}$$

$$\frac{\partial}{\partial t}(\rho u) + \frac{\partial}{\partial x}\left(\rho u^2 + p\right) = 0, \tag{3.2}$$

$$\frac{\partial}{\partial t}E + \frac{\partial}{\partial x}(u(E+p)) = 0. \tag{3.3}$$

Both solvers obtain the velocity field by solving the momentum equation (3.2). The difference arises in how the two methods determine the pressure field: where a pressure-based solver sets up a pressure (or pressure-correction) equation, which is derived by manipulating the continuity (3.1) and momentum (3.2) equations, a density-based solver first obtains the density field directly from the continuity equation and then derives the pressure field from that using an equation of state.

In this chapter we will look at how a particular class of finite volume schemes, called central schemes, solves equations 3.1-3.3. Then we will delve into the code for the rhoCentralFoam solver, which implements one such method.

3.1 Introduction: Finite Volumes for scalar conservation laws

First consider the generalized problem of a scalar conservation law [6], formulated as

$$\frac{\partial}{\partial t}U + \frac{\partial}{\partial x}f(U) = 0. \tag{3.4}$$

Here U is the quantity being conserved, and f is the flux function.

We will constrain ourselves to one spatial dimension and regular grids, but the method can be generalized to several dimensions and irregular grids.

In a Finite Volumes method, we work with *cell averages:*

$$U_j^n \approx \frac{1}{\Delta x} \int_{x_{j-\frac{1}{2}}}^{x^{j+\frac{1}{2}}} U(x, t^n) dx.$$

This is in stark contrast to Finite Difference methods, where point-values are stored – the advantage being that the cell average is well-defined even for discontinuous functions. This is a desirable property when dealing with supersonic flows, since shock waves are indeed discontinuities in the fluid quantities.

We can now rewrite our conservation law 3.4 in *integral form*, by integrating over the space-time domain $[x_{j-\frac{1}{2}}; x_{j+\frac{1}{2}}] \times [t^n; t^{n+1}]$:

$$\int_{t^n}^{t^{n+1}} \int_{x_{j-\frac{1}{2}}}^{x^{j+\frac{1}{2}}} \frac{\partial}{\partial t} U \, dxdt + \int_{t^n}^{t^{n+1}} \int_{x_{j-\frac{1}{2}}}^{x^{j+\frac{1}{2}}} \frac{\partial}{\partial x} f(U) \, dxdt = 0.$$

By using the fundamental theorem of calculus we can manipulate the above into a simpler form:

$$U_j^{n+1} = U_j^n - \frac{\Delta t}{\Delta x} \left(\bar{F}_{j+\frac{1}{2}}^n - \bar{F}_{j-\frac{1}{2}}^n \right), \tag{3.5}$$

where U_j^n is the cell-average of U in cell j at time t^n, and $\bar{F}_{j+\frac{1}{2}}$ is the flux over the cell face at $x_{j+\frac{1}{2}}$, averaged over the time step:

$$\bar{F}_{j+\frac{1}{2}}^n = \frac{1}{\Delta t} \int_{t^n}^{t^{n+1}} f(U(x_{j+\frac{1}{2}}, t)) \, dt. \tag{3.6}$$

Note that the relation 3.5 is not explicit yet, since we need a priori knowledge of the solution to compute $\bar{F}_{j+\frac{1}{2}}^n$. However, S. K. Godunov showed in 1959 [3] that the approximate flux in 3.6 is actually *constant in time*, allowing us to calculate it explicitly based on the conditions at time t^n:

$$\bar{F}_{j+\frac{1}{2}}^n = \frac{1}{\Delta t} \int_{t^n}^{t^{n+1}} f(U(x_{j+\frac{1}{2}}, t)) \, dt = F_{j+\frac{1}{2}}^n.$$

This is thanks to the realization that at time t^n, since Finite Volumes effectively approximates the solution U^n as a piecewise constant function comprised of the cell averages in each cell, we effectively have a Riemann problem at each cell interface $x_{j+\frac{1}{2}}$ where there is a jump in the value of U. This means that, if we can solve that Riemann problem at time t^n explicitly on a *local* scale, we can solve the scalar conservation problem on a *global* scale!

This leads us to the general Finite Volumes scheme for conservation laws:

$$U_j^{n+1} = U_j^n - \frac{\Delta t}{\Delta x}\left(F_{j+\frac{1}{2}}^n - F_{j-\frac{1}{2}}^n\right). \tag{3.7}$$

Although the Riemann problem arising at cell interfaces can sometimes be solved analytically, this is not always possible or practical. Therefore, most Finite Volume schemes use *approximate* Riemann solvers to evaluate the fluxes $F_{j+\frac{1}{2}}^n$: how exactly this is done is the main ingredient to any FVM scheme.

3.1.1 Central schemes

One such class of approximate Riemann solvers are called *central schemes*. These are based on a simple idea: the solution of a Riemann problem is replaced by one consisting of two waves traveling in opposite directions with speeds $s_{j+\frac{1}{2}}^l$ and $s_{j+\frac{1}{2}}^r$. It can be shown that the numerical flux then takes the form [6]:

$$F_{j+\frac{1}{2}}^n = \frac{s_{j+\frac{1}{2}}^r f(U_j^n) - s_{j+\frac{1}{2}}^l f(U_{j+1}^n) + s_{j+\frac{1}{2}}^r s_{j+\frac{1}{2}}^l (U_{j+1}^n - U_j^n)}{s_{j+\frac{1}{2}}^r - s_{j+\frac{1}{2}}^l}. \tag{3.8}$$

The most prominent example of a central scheme is the Lax-Friedrichs scheme, which assumes that both waves travel at the maximum speed possible, such that they don't interfere with the Riemann problems at neighboring boundaries. This speed is given by the grid spacing Δx and time step Δt:

$$s_{j+\frac{1}{2}}^l = -\frac{\Delta x}{\Delta t},\ s_{j+\frac{1}{2}}^r = \frac{\Delta x}{\Delta t}.$$

Substituting into 3.8 yields

$$F_{j+\frac{1}{2}}^n = \frac{f(U_j^n) + f(U_j^n + 1)}{2} - \frac{\Delta x}{2\Delta t}(U_{j+1}^n - U_j^n),$$

and further using 3.7 we obtain the fully discrete formulation of the Lax-Friedrichs scheme:

$$U^{n+1} = \frac{U_{j+1}^n - U_{j-1}^n}{2} - \frac{\Delta t}{\Delta x}(f(U_{j+1}^n) - f(U_{j-1}^n)). \tag{3.9}$$

It can be shown that this scheme is first-order accurate and stable. The main strength of this scheme, and in fact all central schemes, is its simplicity – it can be applied as a "black box" solver for general (systems of) conservation laws 3.4, since it does not involve any costly characteristic decomposition of the flux function f. However there is no free lunch, as they say, and the Lax-Friedrichs scheme comes with one major

disadvantage: it suffers from large numerical dissipation, which smears over shocks and the extrema of rarefaction waves.

There have been many proposed improvements to the Lax-Friedrichs scheme. In particular, rhoCentralFoam uses one such scheme proposed by Kurganov and Tadmor in 1999 [5]. Delving into the details of how this is derived and implemented is beyond the scope of this work, however it is worthwhile to mention the two principal improvements made over the Lax-Friedrichs scheme and the implications these have:

- The piecewise constant reconstruction of the solution U^n by cell averages, which produces first-order accurate schemes in space, is replaced by a piecewise *linear* reconstruction, subject to the condition that the cell average be the same. This brings the spatial discretization up to second order accuracy. This greatly decreases the numerical dissipation present in the Lax-Friedrichs scheme.
- The simplistic assumption made by the Lax-Friedrichs scheme about the wave propagation speeds is refined to make a more accurate estimate of the local speed of propagation. This also aids in counteracting the excessive numerical dissipation.

It should also be noted that these schemes can be recast into semi-discrete form, meaning that it is straightforward to increase the order of accuracy in time by applying one of the many well-known high-order time integration schemes such as the Crank-Nicholson scheme which is often used by OpenFOAM applications.

3.2 Implementation in rhoCentralFoam

It is straightforward to see how the one-dimensional Euler equations 3.1-3.3 can be recast into a form matching that of the generalized conservation law 3.4:

$$\frac{\partial}{\partial t}\begin{bmatrix}\rho\\ \rho u\\ E\end{bmatrix} + \frac{\partial}{\partial x}\begin{bmatrix}\rho u\\ \rho u^2 + p\\ u(E+p)\end{bmatrix} = 0. \tag{3.10}$$

The conserved quantity U is now a vector, however the scalar schemes described above can still be applied in a straightforward manner to each of the three equations separately [5]; furthermore they can be generalized to three spatial dimensions, but that process will not be covered in detail here.

Looking at the source code for OpenFOAM's rhoCentralFoam, we find the implementation of (the three-dimensional variant of) the Euler equations described above:

- First of all, the continuity equation 3.1. Note that the variable `phi` is used in OpenFOAM to represent the mass flux over face areas, and equates to $\rho \cdot u$ in our equations.

```
1 // --- Solve density (continuity equation)
2 solve(fvm::ddt(rho) + fvc::div(phi));
```

- Secondly, the momentum equation 3.2. Here `rhoU` is $\rho \cdot u$, and `phiUp` equates to $(\rho \cdot u) \cdot u + p$.

```
// --- Solve momentum
solve(fvm::ddt(rhoU) + fvc::div(phiUp));
```

- Next, we have the energy equation 3.3. Similar to above, `rhoE` is $\rho \cdot E$ and `phiEp` is $\rho u(E + p)$. Also note we have an additional term for viscosity, and that the implementation scales this equation by ρ.

```
// --- Solve energy
solve
 (
     fvm::ddt(rhoE)
   + fvc::div(phiEp)
   - fvc::div(sigmaDotU)
 );
```

- Finally, the pressure field is updated using the compressibility $\psi = \frac{\rho}{p}$:

```
/// --- Solve pressure
p.ref() = rho() / psi();
```

  How ψ is evaluated depends on the thermophysical configuration of OpenFOAM – specifically, which equation of state is being used. In the most common case (and the one used in this work) of the ideal gas law we have:

$$pV = nRT \Rightarrow p = \frac{nM}{V}\frac{RT}{M} = \rho\frac{RT}{M} \Rightarrow \psi = \frac{\rho}{p} = \frac{M}{RT},$$

  where M is the molar mass of the simulated fluid.

For the sake of brevity we will not cover in-depth the manner in which the fields `rhoU`, `phiUp`, etc. are assembled using the schemes by Kurganov and Tadmor. This process is complicated somewhat by the three-dimensional nature of the solver, and the fact that OpenFOAM uses generalized irregular polyhedral meshes.

Chapter 4

Comparison: supersonic flow past wedge

The first test case taken into consideration was that of supersonic airflow past a wedge. This textbook steady-state example of compressible flow has an analytical solution, making it an ideal starting point for a comparison between solvers.

4.1 Introduction

As illustrated in Figure 4.1, a supersonic flow from the left impinges on an inclined surface (the wedge), generating a discontinuity in the flow called an *oblique shock*. Given the inlet Mach number M_1 and the angle θ of the wedge's surface, the shock's angle β is given by the $\theta - \beta - M$ equation:

$$\tan\theta = 2\frac{M_1^2 \sin^2\beta - 1}{M_1^2(\gamma + \cos 2\beta) + 2}\cot\beta$$

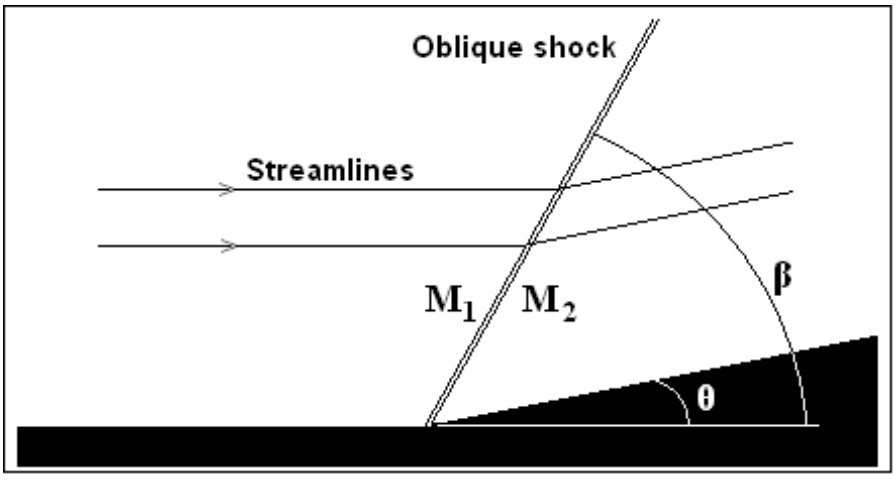
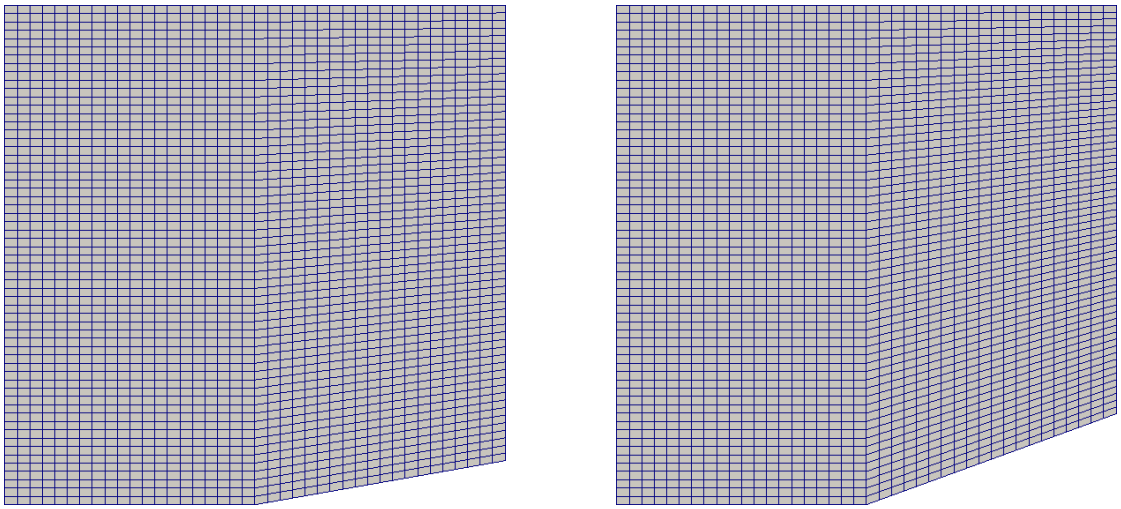


Figure 4.1: An illustration of the wedge problem (Source: Wikimedia Commons).



(a) 10 degree wedge.

(b) 20 degree wedge.

Figure 4.2: The two computational meshes used.

Here γ is the *adiabatic index*, and depends on the physical properties of the modelled fluid. For all calculations presented in this work, γ is taken to be 1.4 (the value for dry air).

The Mach number M_2 after the shock is derived from the relations for *normal* shocks, by first calculating the Mach numbers in perpendicular direction relative to the shock front:

$$M_{n1} = M_1 \sin\beta$$
$$M_{n2}^2 = 1 - \frac{M_{n1}^2 - 1}{1 + \frac{2\gamma}{\gamma+1}(M_{n1}^2 - 1)}$$
$$M_2 = \frac{M_{n2}}{\sin(\beta - \theta)}$$

We use the same approach to compute the pressure jump $\frac{p_2}{p_1}$ across the shock:

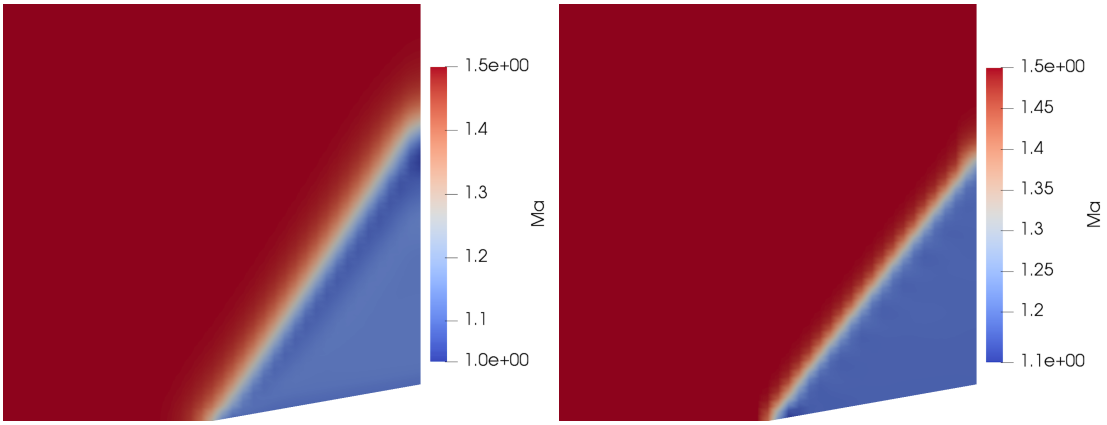
$$\frac{p_2}{p_1} = \frac{2\gamma}{\gamma+1} M_{n1}^2 - \frac{\gamma - 1}{\gamma + 1}$$

Two geometries were used for the comparison: one with a 10 degree wedge and a second with a 20 degree angle. Each of these was run with two inlet Mach numbers, making four comparison cases. The computational meshes used are depicted in Figure 4.2.

Table 4.1 shows the exact values of the observed quantities calculated for each of the test cases, for use as reference for the numerical experiments.

Table 4.1: Exact solutions of the oblique shock problems considered

θ	M_1	β	M_2	p_2/p_1
10	**1.5**	56.6786767	1.11438369	1.66619322
	2	39.3139318	1.64052221	1.7065786
20	**2**	53.4229405	1.21021838	2.8428627
	3	37.7636341	1.99413166	3.77125746



(a) rhoPimpleFoam's solution.

(b) rhoCentralFoam's solution.

Figure 4.3: Side-by-side comparison of the solutions to the first test case with $M_1 = 1.5$ and $\theta = 10°$.

4.2 Numerical experiments

In this section we will investigate how well rhoPimpleFoam and rhoCentralFoam (implemented in OpenFOAM v4.0) fare at reproducing the analytical results. Each solver was run with all four test cases presented above. The Van Leer scheme was used in all cases for spatial discretization, and implicit Euler for time discretization – although it is less relevant, since the problem being treated is steady-state.

We expect a density-based solver to resolve the shocks more clearly than its pressure-based counterpart. This is because, since it explicitly solves Riemann problems at each cell interface, the discontinuities that arise in compressible problems should be transported more accurately. In particular, pressure-based solvers tend to diffuse shocks gradually as they travel through the fluid – a phenomenon that a density-based solver should not suffer from.

Figure 4.3 shows the Mach number values calculated by rhoPimpleFoam and rhoCentralFoam in the case where $\theta = 10$ and $M_1 = 1.5$. It is apparent that the density-based approach resolves shocks more clearly than the pressure-based: despite the use of Van Leer discretization, which generally produces rather sharp shocks, rhoPimpleFoam

visibly diffuses the discontinuity much more than rhoCentralFoam – in fact the latter resolves the shock to a length comparable to the cell size used. On top of that, rhoPimpleFoam diffuses the shock more and more the further away from the source of the shock we move (the wedge) – an unphysical behaviour typical of pressure-based solvers and completely remedied by rhoCentralFoam's approach. Finally, rhoPimpleFoam presents overshooting after the shock, something which is almost completely absent from rhoCentralFoam's solution.

The more attentive reader will have noticed another difference between the solutions: the two shocks propagate at different slopes – meaning that at least one of the solvers presents some inaccuracy when it comes to the calculated slope β of the shock. This brings us to the next part of our analysis: a comparison of the slope β of the shock, as well as values for M_2 and $\frac{p_2}{p_1}$ across the shock, to the analytical solutions calculated above.

Tables 4.2 and 4.3 show the results for these values calculated by the rhoPimpleFoam and rhoCentralFoam solvers respectively, including relative errors of each figure with respect to the reference (analytical) solution.

Table 4.2: Results from the rhoPimpleFoam solver

θ	M_1	β	Rel. error	M_2	Rel. error	p_2/p_1	Rel. error
10	**1.5**	54	-4.73%	1.115618327	0.11%	1.666173008	0.00%
	2	39.5	0.47%	1.641311905	0.05%	1.707707792	0.07%
20	**2**	52	-2.66%	1.212560159	0.19%	2.842354736	-0.02%
	3	39	3.27%	1.988981188	-0.26%	3.798757552	0.73%

Before we analyze these numbers, a note on method: values for M_2 and p_2 were obtained by averaging over the range below the shock wave, at the right boundary of the computational domain. Values for β were measured by hand on a contour plot of the solutions and, as such, are subject to measurement error estimated to be within $\pm 3\%$ relative to the values considered.

It is immediately apparent that rhoPimpleFoam is extremely adept at predicting the values of M_2 and p_2 across the shock – in fact, all of its predictions lie within 1% of the analytical values. rhoCentralFoam on the other hand seems less reliable, with errors

Table 4.3: Results from the rhoCentralFoam solver

θ	M_1	β	Rel. error	M_2	Rel. error	p_2/p_1	Rel. error
10	**1.5**	52	-8.25%	1.180810638	5.96%	1.576529862	-5.38%
	2	38.5	-2.07%	1.690511905	3.05%	1.658893471	-2.79%
20	**2**	49.5	-7.34%	1.331521595	10.02%	2.641398152	-7.09%
	3	37.5	-0.70%	2.127863366	6.71%	3.58902854	-4.83%

of up to 10% on some values.

When it comes to the angle of the shock β, once again rhoPimpleFoam seems to outperform rhoCentralFoam in terms of relative accuracy in most cases, even accounting for measurement errors. The two solvers are comparable for shallow angles, however it appears that rhoCentralFoam suffers particularly when dealing with steeper values of β. Whether this is due purely to happenstance, or whether there exists a real correlation remains unclear; one might speculate that the alignment of the mesh cell faces could relate to this issue – however further investigation would be required to make any certain claims.

4.3 Conclusions

As is often the case, there is no clear answer as to which solver is most accurate for this simple steady-state case study. Rather, the two solvers excel at different tasks: the density-based rhoCentralFoam is adept at resolving sharp shocks and propagating them accurately in space; however its pressure-based counterpart rhoPimpleFoam outperforms it when it comes to predicting the change in physical properties across the discontinuity. Depending on the application, these differences may make one or the other solver more desirable; however for the purpose of calculating the flight performance of a rocket rhoPimpleFoam seems to be the clear winner.

Chapter 5

Comparison: Sod's shock tube

Fist investigated in-depth by Gary A. Sod in 1978 [8], Sod's shock tube has become a standard test for the accuracy of numerical solvers in the field of computational fluid dynamics. Its simple nature means that an analytical solution can be computed, despite being a time-dependent problem unlike the previous test case. In the following, both of the OpenFOAM solvers being considered are run on Sod's shock tube, and the results compared to the analytical solution to the problem. This was generated using a freely available analytical solver implemented in Python and maintained by Dr. I. Backus [1].

5.1 Introduction

We have a 1-dimensional geometry, with $x \in [0;1]$. Here a Riemann problem is set up as initial conditions: we therefore have two regions of fluid separated by a diaphragm at $x = 0.5$. The left and right regions of the initial conditions have different physical properties:

$$\begin{pmatrix} p_L \\ U_L \\ \rho_L \end{pmatrix} = \begin{pmatrix} 1.0 \\ 0.0 \\ 1.0 \end{pmatrix}; \quad \begin{pmatrix} p_R \\ U_R \\ \rho_R \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0.0 \\ 0.125 \end{pmatrix} \tag{5.1}$$

Upon removing the diaphragm at $t = 0$, the discontinuity evolves into several distinct regions of fluid with varying pressure, density and temperature. Solving the Euler equations for the given problem yields three distinct features in the fluid flow, namely a rarefaction wave, a contact discontinuity and a shock wave. These can be seen in a plot of the density after some time (see figure 5.1).

We observe the density profile at a time $t > 0$ before any wave has reached the boundary, such as in figure 5.1. Points x_1 and x_2 represent the extremes of the rarefaction wave which characterizes region II. Here the solution is continuous, however the derivatives of some fluid quantities may not be [8]. Point x_3 is the position that an element of fluid originally at $x_0 = 0.5$ has reached by time t. At this position there is

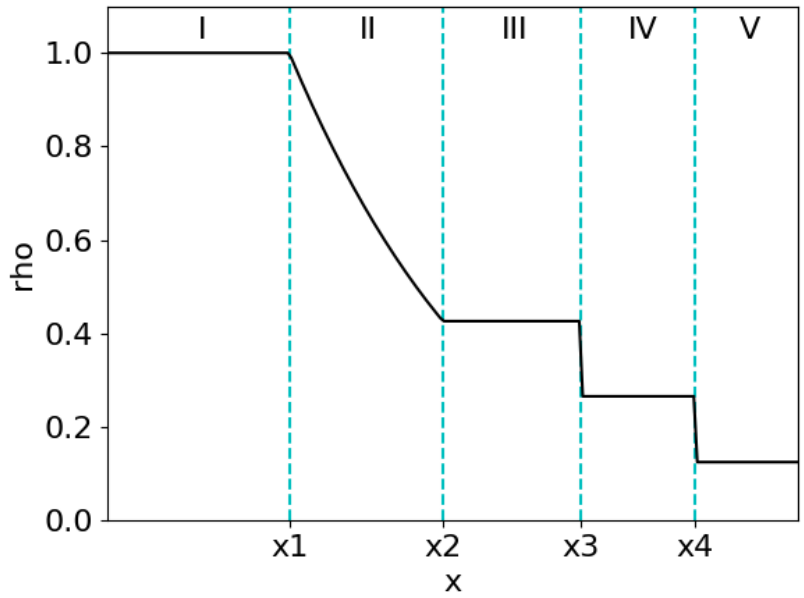


Figure 5.1: The exact solution of density at $t = 0.2$, with the ensuing fluid regions.

a contact discontinuity: the pressure and velocity are continuous, but there is a jump in the density and temperature which travels at the velocity of the fluid in this region. x_4 marks the location of he shock wave moving to the right: all fluid quantities will in general be discontinuous across such a shock. Regions I and V are still unaffected and present the initial values 5.1 of the left and right sides of the Riemann problem.

The nature of this problem is such that we see all three types of discontinuities that occur in compressible fluid flows: rarefaction waves, contact discontinuities and shock waves. This property makes it an ideal test case to benchmark the performance of compressible and transient numerical solvers.

5.2 Numerical experiments

5.2.1 Initial conditions in OpenFOAM

We would like to subject the two solvers rhoPimpleFoam and rhoCentralFoam to Sod's shock tube test, in order to compare their performance. However, OpenFOAM does not allow us to explicitly define the density ρ of the fluid in the initial conditions; instead, we get to define the temperature T. We must therefore calculate the temperatures which will give us the correct initial densities ρ_L and ρ_R from 5.1 for both sides of the Riemann problem. This can be done easily by leveraging the ideal gas relation:

$$pV = nRT, \tag{5.2}$$

where V is the volume of fluid, n is the amount of substance in moles and $R = 8.314\,463\,\mathrm{m^3\,Pa\,K^{-1}\,mol^{-1}}$ is the ideal gas constant. We relate the density ρ to the volume V and the amount of substance n via the molar mass of the fluid M, which we defined in the OpenFOAM configuration as $M = 0.028\,97\,\mathrm{kg\,mol^{-1}}$ (corresponding to dry air):

$$\rho = \frac{nM}{V} \Rightarrow \frac{M}{\rho} = \frac{V}{n} \tag{5.3}$$

Combining 5.2 and 5.3 we can calculate the temperatures we need:

$$T = \frac{pM}{\rho R} \tag{5.4}$$

$$\begin{pmatrix} p_L \\ U_L \\ T_L \end{pmatrix} = \begin{pmatrix} 1.0 \\ 0.0 \\ 3.484290 \times 10^{-3} \end{pmatrix}; \quad \begin{pmatrix} p_R \\ U_R \\ T_R \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0.0 \\ 2.787432 \times 10^{-3} \end{pmatrix} \tag{5.5}$$

5.2.2 Results

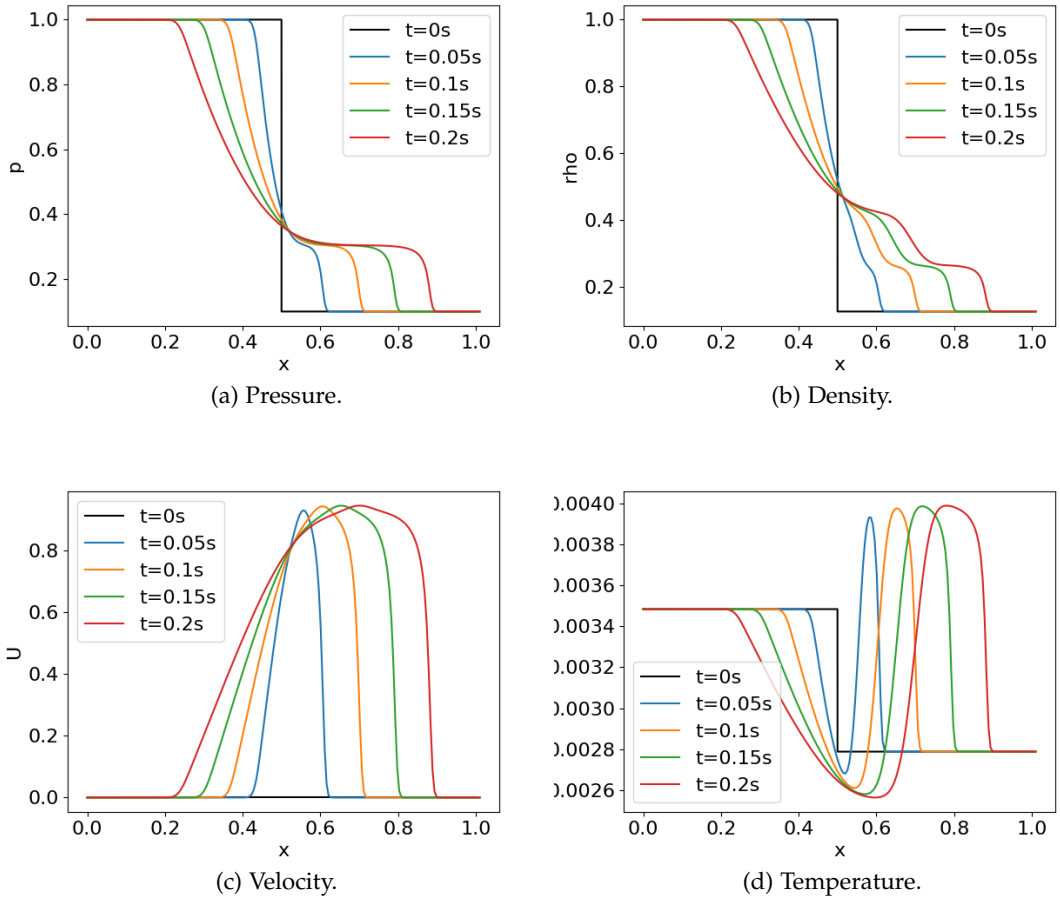
All simulations run in this section were performed with 200 cells, upwind spatial discretization and implicit Euler time discretization. The time step was chosen to be $\Delta t = 1 \times 10^{-4}$s.

Figures 5.2 and 5.3 show the time evolution of the solutions from $t = 0$ to $t = 0.2$s calculated by rhoPimpleFoam and rhoCentralFoam respectively. Similar to the wedge test case, we see that rhoCentralFoam tends to resolve much sharper shocks than rhoPimpleFoam, despite the extremely dissipative upwind discretization. This can be seen in particular when comparing the pressure or velocity graphs for the two solvers. What's more, rhoPimpleFoam tends to diffuse discontinuities gradually over time as well: this is particularly egregious when looking at the evolution of the shock wave in the velocity profile over time. On the other hand, rhoCentralFoam maintains the shock wave unchanged as it travels along the x axis.

Figure 5.4 compares the numerical solutions from both solvers to the exact solution at $t = 0.2$. Once again, rhoCentralFoam presents much sharper discontinuities than its pressure-based counterpart; however, their positions and the jumps in the values of fluid quantities are not always as accurate. For instance, the rarefaction wave is visibly shifted in the temperature plot; and the temperature in region III (between the rarefaction wave and the contact discontinuity) appears overestimated with respect to the exact solution.

5.3 Conclusions

In addition to the qualitative considerations made above, we wish to make a more quantitative evaluation of the accuracy of two solvers. As a measure of the overall



(a) Pressure.

(b) Density.

(c) Velocity.

(d) Temperature.

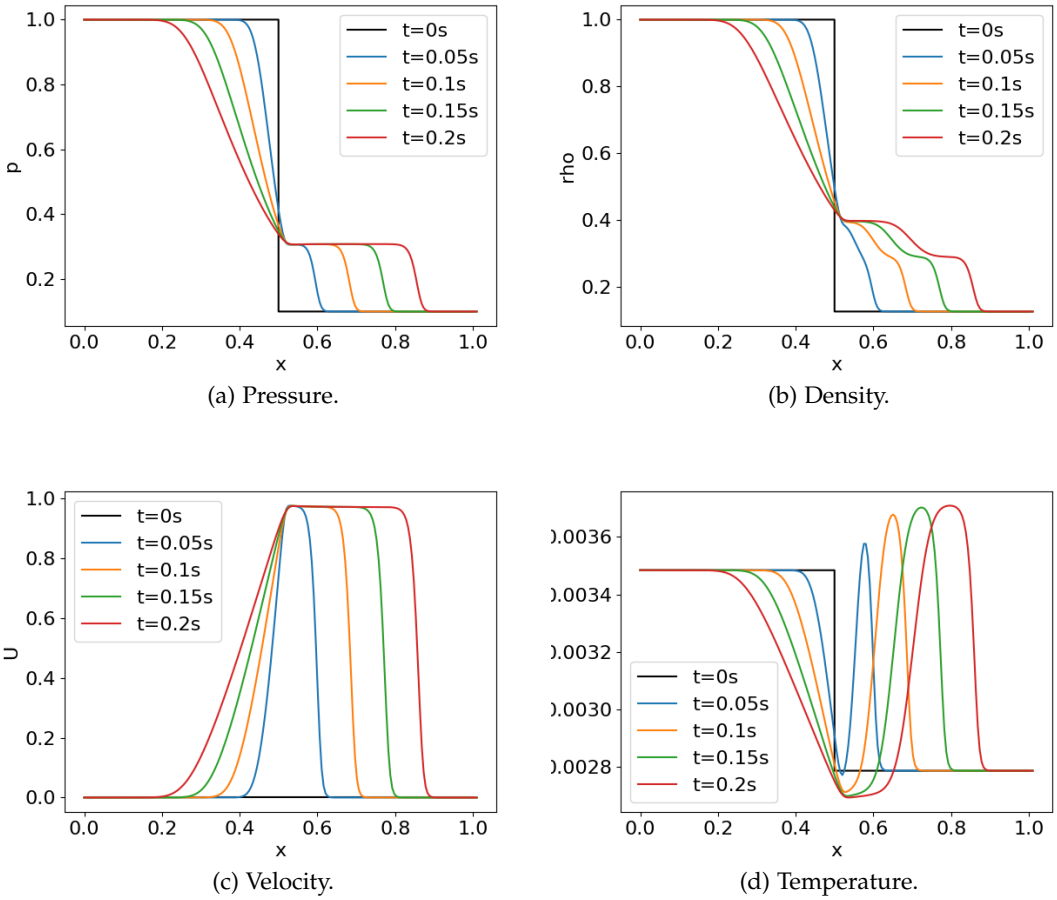
Figure 5.2: Time evolution of the rhoPimpleFoam solution from $t = 0$ to $t = 0.2$.

accuracy of each solver, we use the normalized L_1 error norm, which we evaluate as:

$$L_1(Q^n, Q^*) = \frac{1}{Q_{ref}} \sum_{i=0}^{N=200} |Q^*(i \cdot \frac{1}{N}, n \cdot \Delta t) - Q_i^n|, \tag{5.6}$$

where Q_i^n is the numerical solution for an arbitrary fluid quantity Q at time step n and cell i, and Q^* denotes the exact solution for that same quantity. Q_{ref} denotes the reference value for Q which was taken to be the initial value of the property in the left half of the domain, as defined in equations 5.1 and 5.5, with the exception of velocity where U_{ref} was taken to be unity.

The resulting values for the L_1 error norm for times $t = 0.05, 0.1, 0.15, 0.2$ are reported in table 5.1 both for rhoPimpleFoam and rhoCentralFoam.



(a) Pressure.

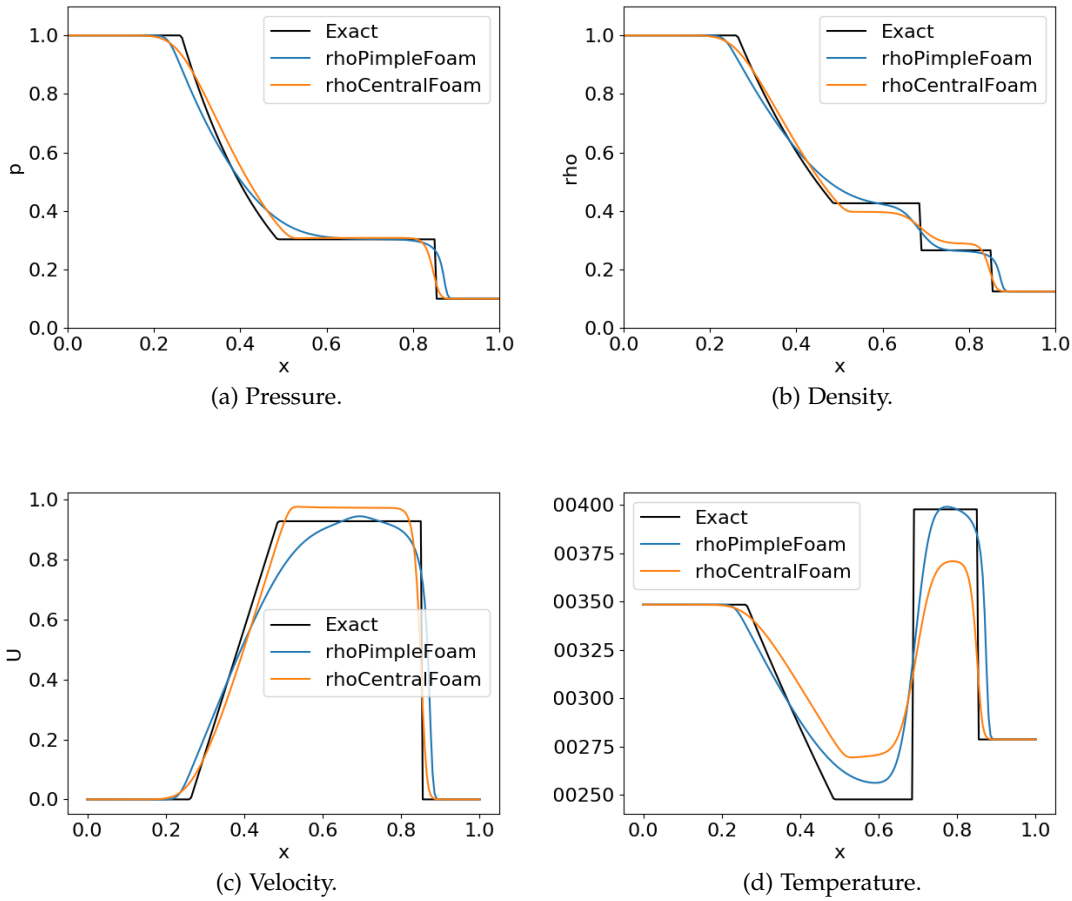
(b) Density.

(c) Velocity.

(d) Temperature.

Figure 5.3: Time evolution of the rhoCentralFoam solution from $t = 0$ to $t = 0.2$.

The takeaway from this table is fairly unambiguous: rhoCentralFoam is overall the more accurate solver, managing to keep its L_1 error norm below or on par with its competitor. The one major exception here is the temperature profile, where rhoPimpleFoam outperforms rhoCentralFoam.



(a) Pressure.

(b) Density.

(c) Velocity.

(d) Temperature.

Figure 5.4: Comparison of the two solvers to the exact solution at $t = 0.2$.

Table 5.1: Normalized L_1 error norms of both numerical solutions

rhoPimpleFoam	**t = 0.05**	**t = 0.1**	**t = 0.15**	**t = 0.2**
p	1.9051	2.6689	3.3577	3.8180
ρ	1.7071	2.5219	3.2185	3.7374
U	4.7684	6.5525	8.3462	9.2785
T	2.7284	3.7757	4.8423	5.4644
rhoCentralFoam	**t = 0.05**	**t = 0.1**	**t = 0.15**	**t = 0.2**
p	1.7272	2.3440	2.8524	3.3806
ρ	1.6435	2.4459	3.1280	3.7782
U	3.7053	4.9110	6.0614	7.1656
T	3.4495	5.6248	7.6288	9.5964

Chapter 6

Conclusions and final remarks

As expected, the density-based solver rhoCentralFoam was able to capture shocks much more accurately than the pressure-based rhoPimpleFoam, which has a tendency to diffuse discontinuities. However, it was surprising to see rhoCentralFoam performing so badly when it came to predicting the jumps in fluid quantities across shocks and other discontinuity. This would seem to indicate that the schemes implemented in rhoCentralFoam from OpenFOAM v4.0 are not conservative, however more detailed investigation would be required in order to be certain.

6.1 A note on runtime performance

One further comparison metric which has not yet been mentioned is the runtime of each solver. Whilst not strictly related to accuracy, this is definitely a factor worth considering when selecting a solver, especially for larger cases. Table 6.1 reports the runtime in seconds for each solver in each test case, as well as the speedup $S = \frac{T_p}{T_s}$ of rhoCentralFoam over rhoPimpleFoam.

In all test cases, rhoCentralFoam is considerably faster than rhoPimpleFoam. This is to be expected – the pressure-based solver sets up a Poisson equation to correct the velocities, and does so several times per time step. This has an obvious impact on performance.

It is worth noting that the comparison made here is not completely fair: both solvers

Table 6.1: Solver runtime

Runtime [s]	**Oblique shock**				**Shock tube**
rhoPimpleFoam	645.49	634.95	636.11	631.61	32.07
rhoCentralFoam	353.67	351.44	346.42	344.58	8.74
Speedup	183%	181%	184%	183%	367%

were run with the same timestep and therefore the same number of time steps were taken by each solver. However, rhoPimpleFoam should, at least in theory, be able to handle larger timesteps owing to its exceptional stability, something which may compensate for the imbalance highlighted here.

6.2 Future work

Owing to the Coronavirus pandemic and the ensuing lockdown, it was unfortunately not possible to compare the two solvers on the actual geometry of the EULER rocket. Despite this, the results obtained here are of great value, and open the door for future work:

- Further investigation of the large errors produced by rhoCentralFoam, perhaps with a more recent version of OpenFOAM which may have made some improvements to the implementation.
- A comparison of the values for drag forces produced by the solvers, for instance using a diamond-shaped airfoil (which has an analytical solution for supersonic speeds) or one of the NACA airfoils, which have been extensively characterized in the past both numerically and experimentally.
- A more rigorous study of the runtime performance of the solvers, particularly with larger test cases requiring parallel execution.

Appendix A

Wedge with reflection

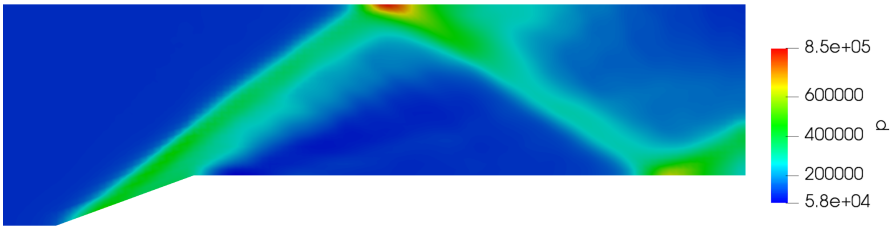


Figure A.1: Pressure field of the rhoPimpleFoam solution.

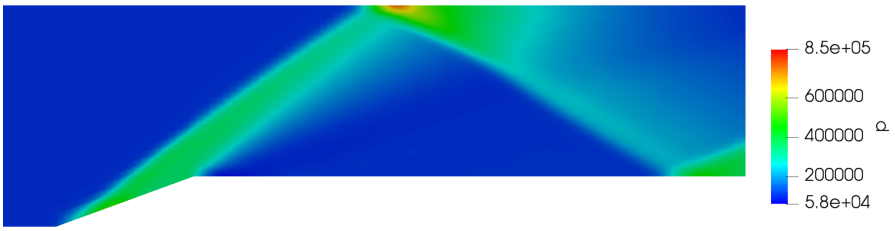


Figure A.2: Pressure field of the rhoCentralFoam solution.

Figures A.1 and A.2 illustrate a variant of the wedge geometry, where the right-hand side of the domain has been extended to allow for two reflections of the shock wave to occur. A symmetry condition was applied at the upper boundary of the domain. Van Leer discretization was used for both solvers; the Mach number at the inlet on the left is 3.

It is immediately apparent that rhoCentralFoam produces a much smoother solution after the shock; rhoPimpleFoam presents some oscillations which are advected with the fluid. The peak pressure at the sites of the reflection is also higher in the rhoPimple-

Foam solution, and the position of the two reflections is also affected by the difference in the solvers.

Bibliography

[1] I. Backus. *Sod shock tube calculator*. URL: https://github.com/ibackus/sod-shocktube (visited on 05/29/2020).

[2] L.S. Caretto et al. "Two calculation procedures for steady, three-dimensional flows with recirculation". In: *Cabannes H., Temam R. (eds) Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics. Lecture Notes in Physics* 19 (1973). DOI: https://doi.org/10.1007/BFb0112677.

[3] S. K. Godunov. "A Difference Scheme for Numerical Solution of Discontinuous Solution of Hydrodynamic Equations". In: *Math. Sbornik* (1959).

[4] R.I. Issa. "Solution of the implicitly discretised fluid flow equations by operator-splitting". In: *Journal of Computational Physics* 62.1 (1986), pp. 40–65. DOI: https://doi.org/10.1016/0021-9991(86)90099-9.

[5] A. Kurganov and E. Tadmor. "New High-Resolution Central Schemes for Nonlinear Conservation Laws and Convection–Diffusion Equations". In: *Journal of Computational Physics* 160.1 (2000), pp. 241–282. ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.2000.6459.

[6] S. Mishra, U. S. Fjordholm, and R. Abgrall. *Advanced numerical methods for CSE (2019 lecture script)*. 2019.

[7] *OpenFOAM v1912 User Guide*. URL: https://www.openfoam.com/documentation/guides/latest/doc/guide-applications-solvers-compressible.html (visited on 05/29/2020).

[8] G. A. Sod. "A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws". In: *Journal of Computational Physics* 27.1 (1978), pp. 1–31. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(78)90023-2.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Comparative study of density-based versus pressure-based solvers for supersonic flow

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Bone	Sean

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 29.05.2020	Sean Bone

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.