Thermodynamically Compatible Finite Volume Schemes

Thesis submitted by

Ravi Pushkar

2018MT60790

under the guidance of

Prof. Harish Kumar

in partial fulfillment of the requirements

for the award of the degree of

Dual Degree(B.Tech and M.Tech) in Mathematics and Computing

Department Of Mathematics

INDIAN INSTITUTE OF TECHNOLOGY DELHI

July 2023

THESIS CERTIFICATE

This is to certify that the thesis titled **Thermodynamically Compatible Finite Volume Schemes**, submitted by **Ravi Pushkar**, 2018MT60790, to the **Department of Mathematics**, **Indian Institute of Technology Delhi**, for partial fulfillment of requirements for the award of the degree of **Dual Degree(B.Tech and M.Tech)** in **Mathematics and Computing**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Harish Kumar
Department of Mathematics
Indian Institute of Technology Delhi

ACKNOWLEDGEMENTS

I heartily express my deepest gratitude to my respected teacher and supervisor of my thesis project, Prof. Harish Kumar, Department of Mathematics, Indian Institute of Technology Delhi, for his valuable guidance, suggestions, and motivation throughout the present work. I am very much indebted to him for his constant encouragement and inspiration. I am grateful to him for introducing me to this area of Finite Volume Schemes and giving me sufficient background on the same. I would also like to thank him for giving me the freedom to perform this research and explore it, for providing novel ideas with access to related literature, and for helping me whenever I was stuck. I would also like to thank Prof. Ananta K. Majee and Prof. Kamana Porwal, Department of Mathematics, Indian Institute of Technology Delhi, for their useful inputs and insights about my project.

Ravi Pushkar
2018MT60790
Department of Mathematics
Indian Institute of Technology Delhi

ABSTRACT

We present a simple and general framework for developing thermodynamically compatible strategies for numerically solving overdetermined hyperbolic PDE systems that meet an additional conservation law. The fact that the entropy inequality is solved as a major evolution equation in this study rather than the conventional total energy conservation law is the primary contribution that these schemes provide to the field of research being discussed. Instead, complete energy conservation is achieved simply by discretizing all other thermodynamically consistent equations. We begin by constructing a discrete framework for the compressible Euler equations to accomplish this. This framework replicates, at a discrete level, the continuous framework presented in Godunov's fundamental study. Correction terms are also included to account for additional factors that may affect the system's entropy dynamics. We demonstrate the applicability of the derived equation in one-dimensional (1D) and can be used in multi-dimensional spaces as well with some modifications, thus encompassing a wide range of scenarios.

We develop numerical implementations for some 1D cases to validate the proposed framework. The codes are designed to solve the entropy equation, compute the flux, and quantify the dissipation and correction terms. Through extensive simulations, we compare the results obtained from our approach with the exact solution in various scenarios. The consistency between our model and the exact solution highlights the effectiveness and accuracy of the proposed entropy-based methodology.

Contents

List of Figures

List of Tables

List of Algorithms

NOTATION

ρ	Density
S	Entropy of the system
$\mathcal{E}$	Total energy potential
q	State vector
C	Courant number
Δt	Time step size
Δx	Length between mesh elements
T	Temperature
γ	Ratio of the specific heat at constant pressure(c_p) to the specific heat at constant volume(c_v)
E	Specific total energy
trA	Trace of matrix A

ABBREVIATIONS

IITD	Indian Institute of Technology Delhi
FVS	Finite Volume Scheme
MHD	Magnetodynamics
HTC	Hyperbolic and Thermodynamically Compatible
SHTC	Symmetric Hyperbolic and Thermodynamically Compatible
PDE	Partial Differential Equations
GPR	Godunov, Peshkov and Romenski
1D	1-Dimension
2D	2-Dimension
FV	Finite Volume
DG	Discontinuous Galerkin
CFD	Computational Fluid Dynamics
SLIC	Slope-Limiter Centred Schemes
PBCs	Periodic boundary conditions

Chapter 1

Introduction

A family of semidiscrete and fully discrete finite volume schemes for overdetermined, hyperbolic, and thermodynamically compatible PDE systems in various dimensions is studied. In particular, we consider the Euler equations of magnetodynamics (MHD), with suitable relaxation of some source terms. In 1961, Godunov published his groundbreaking paper “An interesting class of quasilinear systems,"[7] in which he discovered the connection between symmetric hyperbolicity in the sense of Friedrichs and thermodynamic compatibility (SHTC), ten years before Friedrichs and Lax[6] independently rediscovered the same connection again in 1971. In a subsequent series of papers, Godunov and Romenski carried out further research on this link between symmetric hyperbolicity and thermodynamic compatibility and generalized the seminal idea of Godunov to the more general SHTC framework of symmetric hyperbolic and thermodynamically compatible systems, which includes not only the compressible Euler equations of gas dynamics but also the magnetohydrodynamics (MHD) equations and the equations of nonlinear hyperelasticity[3]. The main aim is to mimic the SHTC (symmetric hyperbolic thermodynamic compatible) framework also on the discrete level by directly discretizing the entropy inequality instead of the total energy conservation law, while the total energy conservation is obtained via an appropriate linear combination as a consequence of the thermodynamically compatible discretization of all equations. SHTC systems can go beyond classical mechanics. Despite the mathematical beauty and rigor of the SHTC framework, up to now, has not been carried much over to the discrete level.

We give a simple and general way to come up with methods for numerically solving overdetermined hyperbolic PDE systems that meet an additional conservation law and are compatible with thermodynamics. The main thing that these plans add to the field of research being talked about is that the entropy inequality is solved as a major evolution equation instead of the traditional total energy conservation law. Instead, you can make sure that all energy is kept the same by making all other thermodynamically consistent equations discrete. This is called the discretization process. To do this, we start by building a discrete structure for the compressible Euler equations. This framework is a discrete version of the continuous framework that Godunov's basic study showed. There are also correction terms to account for other things that might change the system's entropy behavior. We show that the resulting equation can be used in one-dimensional (1D). Moreover, it can be carried over to multi-dimensional spaces as well.

To prove that the proposed structure works, we make numerical implementations for the 1D cases. The codes are made to solve the equation for entropy, figure out the flow, and measure the loss and correction terms. We compare the results of our method to the exact solution in different situations by running a lot of simulations. The fact that our model and the real answer are the same shows that the proposed entropy-based method works and is accurate.

In chapter (2), we explored the existing research and developments in the field of computational fluid dynamics, specifically focusing on the Euler equations and the involved variables. In chapter (3), the governing equations are introduced along with the terms involved. Further, the exact formulae of these physical terms are derived. The algorithm is discussed in the chapter (4) and its results on various problems like the Sod problem, Lax problem, etc. Along with this, the numerical convergence and error

analysis is done. The conclusions are presented together with an outlook to future work in the chapter (5).

Chapter 2

Literature Review

2.1 Turbulent Shallow Water Flow Systems

The mathematical model of shear shallow water flows of constant density is studied[4]. A novel reformulation based on a decomposition of the Reynolds stress tensor $\mathbf{P}$ as $\mathbf{P} = \mathbf{Q}\mathbf{Q}^{\mathbf{T}}$. Further, viscous terms are introduced, and finally, we prove that the proposed viscous system is thermodynamically compatible with energy conservation and entropy inequality.

2.2 Governing Equations

Considering the over-determined hyperbolic model for turbulent shear shallow water flows in multiple space dimensions in gravity-free space and non-viscous system (i.e., $g = 0, b = 0, and, C_f = 0$), we have

$$\boxed{\partial_t h + \nabla \cdot (h\mathbf{v}) = 0} \tag{2.1}$$

$$\boxed{\partial_t (h\mathbf{v}) + \nabla \cdot (h\mathbf{v} \otimes \mathbf{v} + h\mathbf{P}) = 0} \tag{2.2}$$

$$\boxed{\partial_t \mathbf{P} + v \cdot \nabla \mathbf{P} + \nabla \mathbf{v} \mathbf{P} + \mathbf{P} \nabla \mathbf{v}^{\mathbf{T}} = 0} \tag{2.3}$$

So, through energy conservation law, we get the following equations:

$$\partial_t (hE) + \nabla \cdot (\mathbf{v}(hE) + hP\mathbf{v}) = 0 \tag{2.4}$$

Also, the total energy is defined as:

$$hE = \frac{1}{2}h||\mathbf{v}||+\frac{1}{2}h\mathbf{P} \tag{2.5}$$

2.3 Reformulaton of the Model

For hyperbolicity, this model requires $tr\mathbf{P} \geq 0$, where $tr\mathbf{P}$ is the trace of the matrix P. In order to guarantee this property also at the discrete level for all times, we will have reformulation of the system (2.1)-(2.3). We will first look at the following homogeneous equation for the symmetric tensor $\mathbf{P}$:

$$\dot{\mathbf{P}} + \mathbf{LP} + \mathbf{PL^T} = 0 \tag{2.6}$$

Here, for any f, $\dot{f}$ means the material time derivative: $\dot{f} = f_t + \mathbf{v} \cdot \nabla$

$$\mathbf{L} = \frac{\partial \mathbf{v}}{\partial \mathbf{x}} = \nabla \mathbf{v}. \tag{2.7}$$

Let us now replace $\mathbf{P}$ by $\mathbf{P} = \mathbf{Q}\dot{\mathbf{Q}}$, and then, the transformed equations with $\mathbf{P}$ replaced by $\mathbf{P} = \mathbf{Q}\dot{\mathbf{Q}}$ will be as follows:

$$(\dot{\mathbf{Q}} + \mathbf{LQ})\mathbf{Q^T} + \mathbf{Q}(\dot{\mathbf{Q}} + \mathbf{LQ}) = 0 \tag{2.8}$$

If $\dot{\mathbf{Q}} + \mathbf{LQ} = \mathbf{B}(\mathbf{Q^T})^{-1}$, with an anti-symmetric tensor $\mathbf{B} = -\mathbf{B^T}$, we can observe that the equations for $\mathbf{P}$ will be satisfied automatically.

2.3.1 What is the geometrical sense of decomposition $\mathbf{P = QQ^T}$?

Gram matrix with two vectors $\mathbf{w_i}$, $i = 1, 2$ is as follows:

$$\mathbf{G} = \begin{pmatrix} \mathbf{w_1 \cdot w_1} & \mathbf{w_1 \cdot w_2} \\ \mathbf{w_1 \cdot w_2} & \mathbf{w_2 \cdot w_2} \end{pmatrix} \tag{2.9}$$

Here, the dot defines the scalar product of vectors. So, it can be written as:

$$\mathbf{G = QQ^T} \tag{2.10}$$

We can see that in our case, $\mathbf{P}$ is the correlation tensor, expressed in terms of the velocity pulsations, as follows:

$$\mathbf{P} = \begin{pmatrix} \overline{v_1'^2} & \overline{v_1' v_2'} \\ \overline{v_1' v_2'} & \overline{v_2'^2} \end{pmatrix} \tag{2.11}$$

Here, the averaging is the depth averaging, which is denoted by a "bar."
Claim: $\mathbf{P}$ can be presented in the form of (2.9), i.e., we can have vectors $\mathbf{w_i}$ such that:

$$\mathbf{P} = \begin{pmatrix} \mathbf{w_1 \cdot w_1} & \mathbf{w_1 \cdot w_2} \\ \mathbf{w_1 \cdot w_2} & \mathbf{w_2 \cdot w_2} \end{pmatrix}$$

Clearly, we take $\mathbf{w_1} = \sqrt{\overline{v_1'^2}}(Cos\theta_1, Sin\theta_1)^T, \mathbf{w_2} = \sqrt{\overline{v_2'^2}}(Cos\theta_2, Sin\theta_2)^T$ with $Cos(\theta_1 - \theta_2) = \frac{\overline{\mathbf{v_1' v_2'}}}{\sqrt{\overline{\mathbf{v_1'^2 v_2'^2}}}}$ for this, and this satisfies our claim. The last relation is well-defined because of Cauchy-Schwartz's inequality.
Now, with $P_{ik} = Q_{im}Q_{km}$, the equations (2.1)-(2.3) is modified as follows:

$$\partial_t h + \partial_m (h v_m) = 0 \tag{2.12}$$

$$\partial_t(hv_i) + \partial_k(hv_iv_k + hP_{ik}) = 0 \tag{2.13}$$

$$\partial_t Q_{ik} + v_m\partial_m Q_{ik} + (\partial_m v_i)Q_{mk} = 0 \tag{2.14}$$

It can be seen that this new system has an extra energy conservation law, which can be obtained from (2.12) - (2.14)

$$\partial_t(hE) + \partial_i((hE)v_i + hQ_{im}Q_{km}v_k) = 0 \tag{2.15}$$

The time derivative of the determinant of $\mathbf{Q}$ can be obtained via the Jacobi formula, using which we can express the derivative of the determinant of a matrix in terms of the inverse of the matrix and the derivatives of the matrix itself:

$$\partial_t|Q| = |Q|Q^{-1}_{ki}\partial_t Q_{ik}, \quad \partial_m|Q| = |Q|Q^{-1}_{ki}\partial_m Q_{ik} \tag{2.16}$$

where Q^{-1}_{ki} is a compact notation for $(Q^{-1})_{ki}$.
Using (2.16) in (2.14), we get

$$\partial_t|Q| + |Q|Q^{-1}_{ki}v_m\partial_m Q_{ik} + |Q|Q^{-1}_{ki}(\partial_m v_i)Q_{mk} = 0 \tag{2.17}$$

$$\implies \partial_t|Q| + v_m\partial_m|Q| + |Q|(\partial_m v_i)\partial_{mi} = 0 \tag{2.18}$$

$$\implies \partial_t|Q| + \partial_m(v_m|Q|) = 0 \tag{2.19}$$

The equation (2.19) is the same as the mass conservation equation (2.12);

$$Taking \quad \psi = \frac{|\mathbf{P}|}{h^2} = \frac{|\mathbf{Q}\mathbf{Q}^\mathbf{T}|}{h^2}, \quad \partial_t(h\psi) + \partial_m(v_m h\psi) = 0 \tag{2.20}$$

2.4 Godunov Form of Nonlinear Systems of Hyperbolic Conservation laws

Firstly, we consider only hyperbolic systems of conservation laws in two space dimensions of the type:

$$\mathbf{q_t} + \partial_k \mathbf{f_k} = 0 \tag{2.21}$$

along with flux tensor $\mathbf{F} = \mathbf{(f_1, f_2)}$, and after the parametrization according to the Godunov, we get

$$(L_p)_t + \partial_k((v_k L)_p) = 0 \tag{2.22}$$

along with the conservation law of the form:

$$\mathcal{E}_t + \partial_k \mathbf{F_k} = 0 \tag{2.23}$$

where $\mathbf{F_k}$ is the total energy flux in the $k-th$ coordinate direction. Equation (2.22)-(2.23) is called *Godunov form of conservation law* and gives an overdetermined system of PDE. This system is thermodynamically compatible if the following conditions hold:

$$\mathbf{q} = L_p, \quad \mathbf{p} = \mathcal{E}_q, \quad \mathbf{f}_k = (v_k L)_p, \quad F_k = \mathbf{p} \cdot \mathbf{f}_k - v_k L \tag{2.24}$$

Here, L is called *generating potential*, and $\mathcal{E}$ is the total energy density. E and L are Legendre transform of each other and thus satisfy the following criteria:

$$L = \mathbf{p} \cdot \mathbf{q} - \mathcal{E}, \quad \mathcal{E} = \mathbf{p} \cdot \mathbf{q} - L \tag{2.25}$$

L and $\mathcal{E}$ are assumed to be strictly convex functions of their arguments,

and thus the transformation matrices between $\mathbf{p}$ and $\mathbf{q}$, which are Hessian matrices of L and $\mathcal{E}$, respectively satisfy:

$$\frac{\partial \mathbf{p}}{\partial \mathbf{q}} = \mathcal{E}_{qq} > 0, \quad \frac{\partial \mathbf{q}}{\partial \mathbf{p}} = L_{pp} > 0, \quad L_{pp} = (\mathcal{E}_{qq})^{-1} \tag{2.26}$$

$$L_{pp} = L^T_{\mathbf{pp}}, \quad \mathcal{E}_{qq} = \mathcal{E}^T_{\mathbf{qq}} \tag{2.27}$$

Claim: Equation (2.23) is a consequence of Equation (2.22).

$$(L_p)_t + \partial_k((v_k L)_p) = 0$$

$$\implies \mathbf{p} \cdot (\mathbf{L_p})_\mathbf{t} + \mathbf{p} \cdot \partial_k \mathbf{f}_k = 0$$

$$\implies (\mathbf{p}L_p)_t - \mathbf{p}_t L_p + \partial_k(\mathbf{p} \cdot \mathbf{f}_k) - (\partial_k \mathbf{p}) \cdot \mathbf{f_k} = 0$$

$$\implies \frac{\partial}{\partial_t}(\mathbf{p}L_p) - L_t + \partial_k(\mathbf{p} \cdot \mathbf{f}_k) - \partial_k \mathbf{p} \cdot (v_k L)_p = 0$$

$$\implies \frac{\partial}{\partial_p}(\mathbf{p}L_p)\mathbf{p}_t - L_t + \partial_k F_k = 0$$

$$\implies (\mathbf{p}L_{pp} + L_p)\mathbf{p}_t - L_t + \partial_k F_k = 0$$

$$\implies \left(\mathbf{p}\frac{\partial \mathbf{q}}{\partial \mathbf{p}} + \mathbf{q}\right)\mathbf{p}_t - L_t + \partial_k F_k = 0$$

$$\implies \left(\frac{\partial}{\partial \mathbf{p}}(\mathbf{p} \cdot \mathbf{q})\right)\mathbf{p}_t - L_t + \partial_k F_k = 0$$

$$\implies \left(\frac{\partial}{\partial \mathbf{p}}(\mathbf{p} \cdot \mathbf{q})\right)\mathbf{p}_t - L_t + \partial_k F_k = 0$$

$$\implies \frac{\partial}{\partial p}(\mathcal{E} + L) - L_t + \partial_k F_k = 0$$

$$\implies \mathcal{E}_t + \partial_k F_k = 0 \tag{2.28}$$

Equivalently, the shallow water subsystem for the flat bottom is as follows:

$$\partial_t h + \partial_k(h v_k) = 0 \tag{2.29}$$

$$\partial_t(hv_i) + \partial_k(hv_iv_k) = 0 \tag{2.30}$$

$$\partial_t\mathcal{E} + \partial_k(\mathcal{E}v_k) = 0 \tag{2.31}$$

2.5 Thermodynamically Compatible vanishing Viscosity Limit

Here, we will assume flat bottom, b = 0, along with $\alpha = C_f = 0$, but in order to guarantee the exact total energy conservation, a non-negative term T_{ik} is added to the governing PDE for $\mathbf{Q}$:

$$\partial_t h + \partial_m(hv_m) = \partial_m\epsilon\partial_m h \tag{2.32}$$

$$\partial_t(hv_i) + \partial_k(hv_iv_k + hP_{ik}) = \partial_m\epsilon\partial_m(hv_i) \tag{2.33}$$

$$\partial_t Q_{ik} + v_m\partial_m Q_{ik} + (\partial_m v_i)Q_{mk} = \partial_m\epsilon\partial_m Q_{ik} + T_{ik} \tag{2.34}$$

$$\partial_t\mathcal{E} + \partial_i((\mathcal{E}_1 + \mathcal{E}_2)v_i + (hP_{ik})v_k) = \partial_m\epsilon\partial_m\mathcal{E} \tag{2.35}$$

where $\epsilon > 0$ is the dissipation coefficient, associated with the small parabolic dissipation term, is added to the equations.

Now, the total energy $\mathcal{E} = hE = \mathcal{E}_1 + \mathcal{E}_2$ can be divided into $\mathcal{E}_1 = hE_1 = \frac{1}{2}hv_iv_i$ and $\mathcal{E}_2 = hE_2 = \frac{1}{2}hQ_{ik}Q_{ik}$, and thus $\mathcal{E}_1$ is the total potential energy associated with the shallow water subsystem (2.32)-(2.35), while $\mathcal{E}_2$ is the total energy associated with the object Q_{ik}.

Further, we will now denote the inviscid part of the total energy flux through the following equation:

$$\mathbf{F} = (\mathcal{E}_1 + \mathcal{E}_2)v_i + hP_{ik}v_k = \mathbf{F_G} + \mathcal{E}_2v_i + hP_{ik}v_k \tag{2.36}$$

where,

$$\mathbf{F_G} = \mathcal{E}_1 v_i \tag{2.37}$$

whose usage we will see later on.

The production term, T_{ik}, which is needed for the consistency of the total energy conservation, is formulated as follows:

$$T_{ik} = \epsilon \frac{Q_{ik}}{htr\mathbf{P}} \partial_m q_i (\mathcal{E}_{q_i q_j}) \partial_m q_j \tag{2.38}$$

Theorem 1. *(Energy Conservation) The energy conservation law (*(2.35) *is a consequence of the equations* (2.32) - (2.34).

Proof. The black terms in equations (2.32) - (2.35) can be directly deduced from the general class of PDE (2.22) - (2.23) by Godunov. So, the compatibility of the shallow water system with the energy conservation law with energy potential $\mathcal{E}$ is trivial. So, now, we have to consider only the remaining terms associated with the quantity Q_{ik}, which are colored red, and the viscous terms, which are colored blue.

Now, we will show the compatibility of the red terms. Since, $(\mathcal{E}_2)_h = E_2 = \frac{1}{2} Q_{ik} Q_{ik} = \frac{1}{2} tr\mathbf{P}, \mathcal{E}_{hv_i} = v_i, \mathcal{E}_{Q_{ik}} = (\mathcal{E}_2)_{Q_{ik}} = h(E_2)_{Q_{ik}} = hQ_{ik}$, now the summation of (2.32) - (2.34) with the thermodynamic dual variables and considering only the new contributions that are not yet contained in the Godunov-form. □

Theorem 2. *(Entropy-type Inequality) A direct consequence of the PDE* (2.34) *without the parabolic dissipative term, i.e., of the equation*

$$\partial_t Q_{ik} + v_m \partial_m Q_{ik} + (\partial_m v_i) Q_{mk} = T_{ik} \tag{2.39}$$

is an entropy-type inequality

$$\partial_t|Q|+\partial_m v_m|Q|) = \epsilon\frac{|Q|\partial_{kk}}{htr\boldsymbol{P}}\partial_m q_i(E_{q_iq_j})\partial_m q_j \geq 0, with\{x,y\} \in \{1,2,3\}. \tag{2.40}$$

Proof. Applying Jacobi identity (2.16) to equation (2.39), we will get

$$\partial_t|Q|+|Q|Q^{-1}_{ki}v_m\partial_m Q_{ik} + |Q|Q^{-1}_{ki}(\partial_m v_i)Q_{mk} = |Q|Q^{-1}_{ki}T_{ik} \tag{2.41}$$

$$\implies \partial_t|Q|+\partial_m(v_m|Q|) = |Q|Q^{-1}_{ki}T_{ik} \tag{2.42}$$

$$|Q|Q^{-1}_{ki}T_{ik} = \epsilon\frac{|Q|Q^{-1}_{ki}Q_{ki}}{htr\mathbf{P}}\partial_m q_i\mathcal{E}_{q_iq_j}\partial_m q_j = \epsilon\frac{|Q|\partial_{kk}}{htr\mathbf{P}}\partial_m q_i\mathcal{E}_{q_iq_j}\partial_m q_j \geq 0 \tag{2.43}$$

Thus, one obtains the following entropy-type inequality associated with the system (2.32) - (2.35):

$$\partial_t|Q|+\partial_m v_m|Q|) = \epsilon\frac{|Q|\partial_{kk}}{htr\mathbf{P}}\partial_m q_i(E_{q_iq_j})\partial_m q_j \geq 0 \tag{2.44}$$

□

Chapter 3

Problem Statement and Proposed Method

3.1 Problem Statement

The main aim is to mimic the SHTC (symmetric hyperbolic thermodynamic compatible) framework also on the discrete level by directly discretizing the entropy inequality instead of the total energy conservation law, while the total energy conservation is obtained via an appropriate linear combination as a consequence of the thermodynamically compatible discretization of all equations. Firstly, the system is studied in 1D in a gravity-free space with zero viscosity and zero additional fluxes. And then, various terms like correction terms and viscosity terms are added to the system.

3.2 Key Terms

Let's now discuss some of the key physical and mathematical terms used here.

3.2.1 Godunov form of hyperbolic equations

Following the classical Finite-volume method (A.1) framework, we seek to track a finite set of discrete unknowns, $Q_i^n = \frac{1}{\Delta x}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} q(t^n, x)dx$ where $x_{i-\frac{1}{2}} = x_{low} + (i - \frac{1}{2})\Delta x$ and, $t^n = n\Delta t$ form a discrete set of points for the hyperbolic problem: $q_t + (f(q))_x = 0$, where indices t and x indicate

the derivations in time and space, respectively. If we integrate a hyperbolic problem over a control volume, we obtain a method of lines(MOL) formulation for the spatial cell averages. Godunov's method replaces the time integral of each $\int_{t^n}^{t^n+1} f\left(q(t, x_{i-\frac{1}{2}})\right) dt$, using forward Euler method.

3.2.2 Forward Euler Method

Let's denote the time at the n^{th} time-step by t_n and the computed solution at the nth time-step by y_n, i.e., $y_n \equiv y(t = t_n)$. The step size h (assumed to be constant for the sake of simplicity) is then given by $h = t_n - t_{n-1}$. Given (t_n, y_n), the forward Euler method (FE) computes y_{n+1} as $y_{n+1} = y_n + hf(x_n, y_n)$[20].

3.2.3 SHTC

Symmetric and Hyperbolic Thermodynamically Compatible systems are a class of systems of quasi-linear conservation laws that every closed system owns the additional conservation law (the analog of the first law of thermodynamics) and can be reduced to the symmetric hyperbolic form.

3.2.4 Numerical Quadrature Rule

Quadrature refers to any method for numerically approximating the value of a definite $\int_b^a f(x)dx$. The goal is to attain a given level of precision with the fewest possible function evaluations. Ex- Trapezoidal rule, Simpson rule, etc. [11]

3.2.5 Primitive and Conservative Forms

In terms of programming convenience, the non-conservation form is the better of the two. It is easier to program because the equations are solved using the primitive variables (ρ, v, T), and their values can be obtained directly.

However, in the conservation form, the equations are expressed in flux variables that need to be modified in terms of solution vectors. The equations are solved for the solution vectors, and then the primitive variables are determined from them. This adds a fair amount of complexity while solving the governing equations in conservation form. [10]

3.2.6 Riemann Problems

A Riemann problem in the theory of hyperbolic equations is a problem in which the initial state of the system is defined as:

$$q(x, t = 0) = \begin{cases} q_L & \text{for } x \leq 0 \\ q_R & \text{for } x > 0 \end{cases}$$

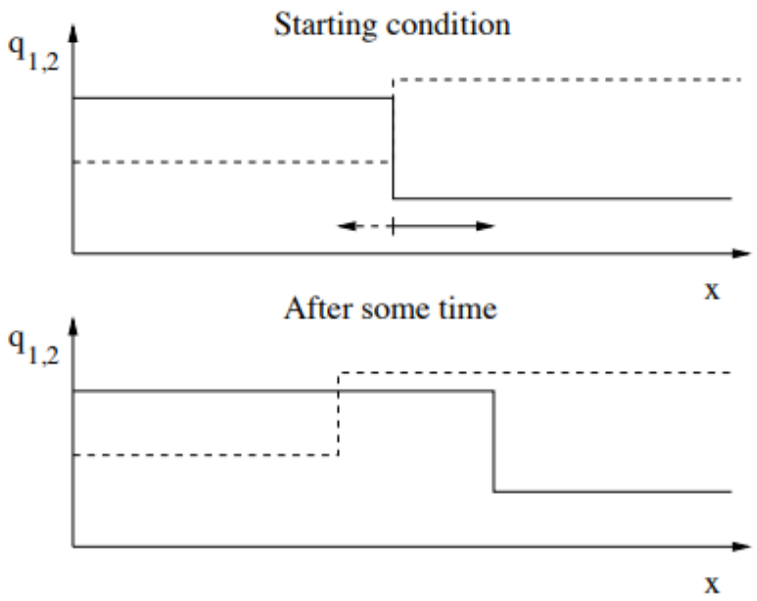

Figure 3.1: Starting Condition and Condition after sometime

Hydrodynamic problems such as *shock tube tests* are used to test the performance of numerical hydrodynamics algorithms.

3.2.7 Periodic Boundary Conditions

Periodic boundary conditions (PBCs) are a set of boundary conditions often chosen for approximating an extensive (infinite) system using a small part called a unit cell. PBCs are often used in computer simulations and mathematical models. An object which has passed through one face of the simulation box should re-enter through the opposite face—or its image should do it.

3.2.8 Courant Number

The Courant number is a dimensionless value representing the time a particle stays in one cell of the mesh. The derivation of the CFL condition leads to the formula for the Courant number and is given by:

$$C = u\frac{\Delta t}{\Delta x}$$

where C is the Courant number, u is velocity magnitude, Δt is time step size and Δx is the length between mesh elements. It must be below 1 and ideally should be 0.7-0.8.

3.3 Governing Equations

The governing PDE system in one dimension is as follows:

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

$$\frac{\partial \rho v}{\partial t} + \frac{\partial(\rho v^2 + p)}{\partial x} = 0 \tag{3.2}$$

$$\frac{\partial \rho S}{\partial t} + \frac{\partial \rho S v}{\partial x} = 0 \tag{3.3}$$

$$\frac{\partial \mathcal{E}}{\partial t} + \frac{\partial(\mathcal{E} v + p v)}{\partial x} = 0 \tag{3.4}$$

The system described here represents the Euler subsystem in one dimension in the gravity-free, non-viscous system with no additional fluxes. Firstly, we will study this system in detail. Afterward, we will include a dissipative term and a correction term to make the scheme more practical. Let's discuss some of the important terms and their thermodynamic formulae.

3.4 Some terms related to the above system of PDEs

Density is represented by ρ while S is the entropy involved in the system, and E is the specific total energy. Here, $\mathcal{E} = \rho E = \mathcal{E}_1 + \mathcal{E}_2$ is the total energy potential with $\mathcal{E}_i = \rho E_i$.
The two contributors to the total energy density are

$$\mathcal{E}_1 = \frac{\rho^\gamma}{\gamma - 1} e^{S/c_v}, \quad \mathcal{E}_2 = \frac{1}{2}\rho v^2 \tag{3.5}$$

We will use the notations $\partial_p = \frac{\partial}{\partial p}$ and $\partial^2_{pq} = \frac{\partial^2}{\partial p \partial q}$ for the first and second partial derivatives w.r.t. generic coordinates or quantities p and q, which may also be vectors or components of a vector. We use lower case subscripts, i, j, k, for tensor indices, while lower case superscripts, ℓ, refer to the spatial discretization index. We denote the spatial control volumes in 1D by $\Omega^\ell = [x^{\ell-\frac{1}{2}}, x^{\ell+\frac{1}{2}}]$ and $\Delta x = x^{\ell+\frac{1}{2}} - x^{\ell-\frac{1}{2}}$ is the uniform mesh spacing.
$\mathbf{q} = \{q_i\} = (\rho, \rho v, \rho S)^T$ denotes the state vector. The thermodynamic dual variable $\mathbf{p} = \mathcal{E}_\mathbf{q} = (r, v, T)^T$ along with

$$r = \partial_\rho \mathcal{E}, \quad v = \partial_{\rho v} \mathcal{E}, \quad T = \partial_{\rho S} \mathcal{E} \tag{3.6}$$

Where T is the temperature.
Gamma(γ) is the ratio of the specific heat at constant pressure(c_p) to the specific heat at constant volume(c_v). The purely hydrodynamic pressure is defined as $p = \rho^2 \frac{\partial E}{\partial \rho}$, where E is the specific total energy. Also,

$$S = \ln\left(\frac{p}{\rho^\gamma}\right), \quad \mathcal{E} = \frac{p}{\gamma - 1} + \frac{1}{2}\rho v^2 \tag{3.7}$$

3.5 Energy Equation as a Consequence of the above three equations

The system is over-determined. We will see that equation (3.4) is a consequence of equations (3.1) - (3.3). Firstly, using (3.2), we get

$$\frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho v^2 + p)}{\partial x} = 0$$

$$\implies v\frac{\partial \rho}{\partial t} + v\frac{\partial(\rho v)}{\partial x} + \rho\frac{\partial v}{\partial t} + \rho v\frac{\partial(v)}{\partial x} + \frac{\partial p}{\partial x} = 0$$

$$\implies \rho\frac{\partial v}{\partial t} + \rho v\frac{\partial(v)}{\partial x} + \frac{\partial p}{\partial x} = 0 \qquad \text{[Using(3.1)]} \tag{3.8}$$

From equation (3.7), we get

$$S = \ln(p) - \gamma\ln(\rho)$$

$$\frac{\partial S}{\partial t} = \frac{\frac{\partial p}{\partial t}}{p} - \frac{\gamma}{\rho}\frac{\partial \rho}{\partial t}, \quad \frac{\partial S}{\partial x} = \frac{\frac{\partial p}{\partial x}}{p} - \frac{\gamma}{\rho}\frac{\partial \rho}{\partial x}$$

Using equation (3.3),

$$S\frac{\partial \rho}{\partial t} + S\frac{\partial \rho v}{\partial x} + \rho\frac{\partial S}{\partial t} + \rho v\frac{\partial S}{\partial x} = 0$$

$$\frac{1}{p}\left(\frac{\partial p}{\partial t} + v\frac{\partial \rho}{\partial x}\right) - \frac{\gamma}{\rho}\left(\frac{\partial \rho}{\partial t} + v\frac{\partial \rho}{\partial x}\right) = 0$$

Using equation (3.7), we get the following

$$\frac{\partial p}{\partial t} + v\frac{\partial p}{\partial x} + \gamma p\frac{\partial v}{\partial x} = 0 \tag{3.9}$$

We have,

$$\mathcal{E} + p = \frac{\gamma p}{\gamma - 1} + \frac{1}{2}\rho v^2$$

Then,

$$\frac{\partial \mathcal{E}}{\partial t} + \frac{\partial \left(\mathcal{E}v + pv\right)}{\partial x}$$

$$= \frac{\partial \left(\frac{p}{\rho^{\gamma-1}} + \frac{1}{2}\rho v^2\right)}{\partial t} + \frac{\partial \left(\frac{\gamma p v}{\rho^{\gamma-1}} + \frac{1}{2}\rho v^3\right)}{\partial x}$$

$$= \frac{1}{\rho^{\gamma-1}}\frac{\partial p}{\partial t} + \frac{\gamma p}{\gamma-1}\frac{\partial v}{\partial x} + \frac{\gamma v}{\gamma-1}\frac{\partial p}{\partial x} + \rho v\frac{\partial v}{\partial t} + \frac{v^2}{2}\left(\frac{\partial \rho}{\partial t} + v\frac{\partial \rho}{\partial x} + \rho\frac{\partial v}{\partial x}\right) + \rho v^2\frac{\partial v}{\partial x}$$

By using (3.1) and (3.8), we have the following:

$$= -\frac{v\frac{\partial p}{\partial x}}{\gamma-1} + \frac{v\gamma\frac{\partial p}{\partial x}}{\gamma-1} + \rho v\frac{\partial v}{\partial t} + \rho v^2\frac{\partial v}{\partial x}$$

Now, making use of the (3.8), we conclude

$$= v\left(\frac{\partial p}{\partial x} + \rho v\frac{\partial v}{\partial x} + \rho\frac{\partial v}{\partial t}\right) = 0$$

Thus, we can now say that the energy conservation equation (3.4) is a consequence of the other three equations (3.1) - (3.3).

3.6 Semi-discrete Godunov formalism for the Euler System

The generating potential, L, is the Legendre transform of the total energy density. It is only related to one of the Euler subsystems, i.e., $\mathcal{E} = \mathcal{E}_1 + \mathcal{E}_2$ and is defined as

$$L = \mathbf{p}\cdot\mathbf{q} - \mathcal{E} \tag{3.10}$$

$$\implies L = \rho r + \rho v^2 + \rho ST - \frac{p}{\gamma-1} - \frac{1}{2}\rho v^2 = \rho r + \frac{1}{2}\rho v^2 + \rho ST - \frac{p}{\gamma-1}$$

We have our state vector $\mathbf{q}$ as follows:

$$\boxed{\mathbf{q} = (\rho, \rho v, \rho S)^T}$$

$$\mathcal{E} = \frac{p}{\gamma - 1} + \frac{1}{2}\rho v^2$$

$$S = ln\left(\frac{p}{\rho^\gamma}\right)$$

Thus,

$$\mathcal{E} = \frac{\rho^\gamma}{\gamma - 1}e^S + \frac{1}{2}\rho v^2$$

3.6.1 Evaluation of p:

$$\mathbf{p} = \left(\frac{\partial \mathcal{E}}{\partial \rho}, \frac{\partial \mathcal{E}}{\partial \rho v}, \frac{\partial \mathcal{E}}{\partial \rho S}\right)^T$$

Now, let us take

$$\rho = m_1, \quad \rho v = m_2, \quad \rho S = m_3$$

Then,

$$\mathcal{E} = \frac{1}{\gamma - 1}m_1^\gamma e^{\frac{m_3}{m_1}} + \frac{1}{2}\frac{m_2^2}{m_1}$$

$$\frac{\partial \mathcal{E}}{\partial m_1} = \frac{\gamma}{\gamma - 1}m_1^{\gamma-1}e^{\frac{m_3}{m_1}} + \frac{1}{\gamma}m_1^\gamma e^{\frac{m_3}{m_1}}\left(\frac{-m_3}{m_1^2}\right) + \frac{1}{2}(-1)\frac{m_2^2}{m_1^2}$$

$$\frac{\partial \mathcal{E}}{\partial m_1} = \frac{\gamma}{\gamma - 1}\rho^{\gamma-1}e^S + \frac{1}{\gamma - 1}\rho^\gamma e^S\left(-\frac{S}{\rho} - \frac{1}{2}\frac{\rho^2 v^2}{\rho^2}\right)$$

$$= \frac{\rho^{\gamma-1}}{\gamma - 1}e^S(\gamma - S) - \frac{1}{2}v^2$$

$$\frac{\partial \mathcal{E}}{\partial m_2} = \frac{1}{2}2\frac{m_2}{m_1} = v$$

$$\frac{\partial \mathcal{E}}{\partial m_3} = \frac{m_1^\gamma}{\gamma - 1}e^{\frac{m_3}{m_1}}\frac{1}{m_3}$$

$$= \frac{\rho^{\gamma-1}}{\gamma - 1}e^S$$

So,

$$\boxed{\mathbf{p} = \left(\frac{\rho^{\gamma-1}}{\gamma-1}e^S(\gamma - S) - \frac{1}{2}v^2, v, \frac{\rho^{\gamma-1}}{\gamma-1}e^S\right)^T}$$

3.6.2 Evaluation of L:

We know that,

$$L = \mathbf{p}\cdot\mathbf{q} - \mathcal{E}$$

$$= \frac{\rho^\gamma}{\gamma-1}e^S(\gamma - S) - \frac{1}{2}\rho v^2 + \rho v^2 + \frac{\rho^\gamma}{\gamma-1}Se^S - \frac{\rho^\gamma e^S}{\gamma-1}\gamma - \frac{\rho^\gamma}{\gamma-1}e^S$$

$$\implies \boxed{L = \rho^\gamma e^S}$$

Thus, we have

$$\boxed{vL = v\rho^\gamma e^S}$$

3.6.3 Evaluation of f:

$$\mathbf{f} = \partial_{\mathbf{p}}(v\rho^\gamma e^S)$$

Let us consider the following three variables m_1, m_2, m_3:

$$m_1 = \frac{\rho^{\gamma-1}}{\gamma-1}e^S(\gamma - S) - \frac{1}{2}v^2, \quad m_2 = v, \quad m_3 = \frac{\rho^{\gamma-1}}{\gamma-1}e^S$$

Clearly,

$$m_1 = \frac{\rho^{\gamma-1}}{\gamma-1}e^S(\gamma - S) - \frac{1}{2}m_2^2$$

$$\implies m_1 + \frac{1}{2}m_2^2 = m_3(\gamma - S)$$

$$\gamma - S = \frac{m_1}{m_3} + \frac{1}{2}\frac{m_2^2}{m_3}$$

$$S = \gamma - \frac{m_1}{m_3} - \frac{1}{2}\frac{m_2^2}{m_3}$$

$$m_3 = \frac{\rho^{\gamma-1}}{\gamma-1} e^{\gamma - \frac{m_1}{m_3} - \frac{1}{2}\frac{m_2^2}{m_3}}$$

$$\implies \rho^{\gamma-1} = (\gamma-1) m_3 e^{\frac{m_1}{m_3} + \frac{1}{2}\frac{m_2^2}{m_3} - \gamma}$$

$$\implies \rho = ((\gamma-1)m_3)^{\frac{1}{\gamma-1}} e^{\frac{m_1}{(\gamma-1)m_3} + \frac{m_2^2}{2m_3(\gamma-1)} - \frac{\gamma}{\gamma-1}}$$

Now, we have the following term

$$v\rho^\gamma S = m_2((\gamma-1)m_3)^{\frac{\gamma}{\gamma-1}} e^{\frac{\gamma m_1}{(\gamma-1)m_3} + \frac{\gamma m_2^2}{2m_3(\gamma-1)} - \frac{\gamma^2}{\gamma-1}} e^{\gamma - \frac{m_1}{m_3} - \frac{1}{2}\frac{m_2^2}{m_3}}$$

$$= m_2((\gamma-1)m_3)^{\frac{\gamma}{\gamma-1}} e^{\frac{m_1}{(\gamma-1)m_3} + \frac{m_2^2}{2m_3(\gamma-1)} - \frac{\gamma}{\gamma-1}}$$

Differentiating it with respect to the first term, i.e., m_1, we get

$$\frac{\partial(v\rho^\gamma e^S)}{\partial m_1} = \frac{1}{(\gamma-1)m_3} m_2((\gamma-1)m_3)^{\frac{\gamma}{\gamma-1}} e^{\frac{m_1}{(\gamma-1)m_3} + \frac{m_2^2}{2m_3(\gamma-1)} - \frac{\gamma}{\gamma-1}}$$

$$= v(\rho^{\gamma-1}e^S)^{\frac{\gamma}{\gamma-1}} e^{\frac{-S}{\gamma-1}} \frac{1}{\rho^{\gamma-1}e^S}$$

$$= v\frac{\rho^\gamma}{\rho^{\gamma-1}} = \rho v$$

Differentiating it with respect to the second term, i.e., m_2, we get

$$\frac{\partial(v\rho^\gamma e^S)}{\partial m_2}$$

$$= ((\gamma-1)m_3)^{\frac{\gamma}{\gamma-1}} e^{\frac{m_1}{(\gamma-1)m_3} + \frac{m_2^2}{2m_3(\gamma-1)} - \frac{\gamma}{\gamma-1}} + m_2((\gamma-1)m_3)^{\frac{\gamma}{\gamma-1}} e^{\frac{m_1}{(\gamma-1)m_3} + \frac{m_2^2}{2m_3(\gamma-1)} - \frac{\gamma}{\gamma-1}}$$

$$\frac{m_2}{(\gamma-1)m_3}(\rho^{\gamma-1}e^S)^{\frac{\gamma}{\gamma-1}} e^{\frac{-S}{\gamma-1}} + v((\gamma-1)m_3)^{\frac{\gamma}{\gamma-1}} e^{\frac{m_1}{(\gamma-1)m_3} + \frac{m_2^2}{2m_3(\gamma-1)} - \frac{\gamma}{\gamma-1}} \frac{v}{\rho^{\gamma-1}e^S}$$

$$= \rho^\gamma e^S + v\frac{\rho^\gamma}{\rho^{\gamma-1}} v$$

$$= \rho^\gamma e^S + \rho v^2$$

$$= p + \rho v^2$$

Differentiating with respect to the third variable, i.e., m_3, we get

$$\frac{\partial(v\rho^\gamma e^S)}{\partial m_3}$$

$$= \gamma m_2 \left((\gamma-1)m_3\right)^{\frac{\gamma}{\gamma-1}} e^{\frac{m_1}{(\gamma-1)m_3}+\frac{m_2^2}{2m_3(\gamma-1)}-\frac{\gamma}{\gamma-1}} + \frac{1}{\gamma-1}\left(\frac{-m_1}{m_3^2}-\frac{1}{2}\frac{m_2^2}{m_3^2}\right) m_2$$

$$\left((\gamma-1)m_3\right)^{\frac{\gamma}{\gamma-1}} e^{\frac{m_1}{(\gamma-1)m_3}+\frac{m_2^2}{2m_3(\gamma-1)}-\frac{\gamma}{\gamma-1}}$$

$$= \gamma v(\rho^{\gamma-1}e^S)^{\frac{1}{\gamma-1}} e^{\frac{-S}{\gamma-1}} + \frac{1}{\gamma-1}\frac{-m_1-m_2^2}{2m_3^2} v(\rho^{\gamma-1}e^S)^{\frac{\gamma}{\gamma-1}} e^{\frac{-S}{\gamma-1}}$$

$$= \gamma v\rho e^{S(\frac{1}{2}-\frac{1}{2})} + \frac{1}{\gamma-1} v\rho^\gamma e^S \left(\frac{-\rho^{\gamma-1}e^S(\gamma-S)(\gamma-1)^2}{(\gamma-1)\rho^{2(\gamma-1)}e^{2S}}\right)$$

$$= \gamma v\rho + \rho v\left(-\gamma+S\right)$$

$$= \rho v S$$

Therefore, we have the following exact form of $\mathbf{f}$:

$$\boxed{\mathbf{f} = \left(\rho v, p+\rho v^2, \rho v S\right)}$$

Clearly, now the equation

$$\boxed{\mathbf{q}_t + \mathbf{f}_x = 0}$$

gives the equations for the Euler subsystem.

3.6.4 Evaluation of $F_\mathbf{G}$:

We know,

$$F_G = \mathbf{p}\cdot\mathbf{f} - vL$$

$$\implies F_G = \frac{\rho^\gamma}{\gamma-1} v e^S(\gamma-S) - \frac{1}{2}\rho v^3 + \rho^\gamma v e^S + \rho v^3 + \frac{\rho^\gamma}{\gamma-1} v S e^S$$

$$\implies F_G = \frac{\rho^\gamma}{\gamma-1} v e^S \gamma + \frac{1}{2}\rho v^3 + \rho^\gamma v e^S$$

Thus, we have the following value of F_G:

$$\boxed{F_G = \rho^\gamma v e^S \left(\frac{2\gamma - 1}{\gamma - 1}\right) + \frac{1}{2}\rho v^3}$$

3.7 Semi-discrete finite volume scheme of the Euler Equation

The Godunov form of the inviscid Euler subsystems is as follows:

$$\mathbf{q} = \partial_{\mathbf{p}} L, \quad \mathbf{p} = \partial_{\mathbf{q}} \mathcal{E}, \quad \mathbf{f} = \partial_{\mathbf{p}}(vL), \quad F_G = \mathbf{p} \cdot \mathbf{f} - vL \tag{3.11}$$

$$(\partial_{\mathbf{p}} L)_t + \partial_x(\partial_{\mathbf{p}}(vL)) = 0 \tag{3.12}$$

A semi-discrete finite volume scheme for (3.12) is as follows:

$$\frac{d}{dt}\mathbf{q}^\ell = -\frac{\mathbf{f}^{\ell+\frac{1}{2}} - \mathbf{f}^{\ell-\frac{1}{2}}}{\Delta x} = -\frac{(\mathbf{f}^{\ell+\frac{1}{2}} - \mathbf{f}^{\ell}) - (\mathbf{f}^{\ell-\frac{1}{2}} - \mathbf{f}^{\ell})}{\Delta x} \tag{3.13}$$

with $\mathbf{f}^\ell = \mathbf{f}(\mathbf{q}^\ell)$ and $\mathbf{f}(\mathbf{q}) = (\rho v, \rho v^2 + p, \rho S v)^T$, the fluxes of the Euler subsystem and $F_G = v(\mathcal{E}_1 + \mathcal{E}_2 + p)$ the total associated energy flux.
In order to obtain a discrete total energy conservation law as a consequence of the discretization of (3.1)-(3.3), we first compute the dot product of the discrete dual variables, $\mathbf{p}^\ell = \partial_{\mathbf{q}} \mathcal{E}(\mathbf{q}^\ell)$, with the semi-discrete scheme:

$$\mathbf{p}^\ell \cdot \frac{d}{dt}\mathbf{q}^\ell = \frac{d}{dt}\mathcal{E}^\ell = -\mathbf{p}^\ell \cdot \frac{(\mathbf{f}^{\ell+\frac{1}{2}} - \mathbf{f}^{\ell}) - (\mathbf{f}^{\ell} - \mathbf{f}^{\ell-\frac{1}{2}})}{\Delta x} = -\frac{D_{\mathcal{E}}^{\ell-\frac{1}{2},-} + D_{\mathcal{E}}^{\ell-\frac{1}{2},+}}{\Delta x} \tag{3.14}$$

The energy fluctuation is defined as $D_{\mathcal{E}}^{\ell-\frac{1}{2},-} = \mathbf{p}^\ell \cdot (\mathbf{f}^{\ell+\frac{1}{2}} - \mathbf{f}^{\ell}), D_{\mathcal{E}}^{\ell-\frac{1}{2},+} = \mathbf{p}^\ell \cdot (\mathbf{f}^{\ell} - \mathbf{f}^{\ell-\frac{1}{2}})$, which must satisfy the consistency property (i.e., the change

in flux is equal to the total change in energy density)

$$D_{\mathcal{E}}^{\ell+\frac{1}{2},-} + D_{\mathcal{E}}^{\ell+\frac{1}{2},+} = \mathbf{p}^{\ell} \cdot (\mathbf{f}^{\ell+\frac{1}{2}} - \mathbf{f}^{\ell}) + \mathbf{p}^{\ell+1} \cdot (\mathbf{f}^{\ell+1} - \mathbf{f}^{\ell-\frac{1}{2}}) = F_G^{\ell+1} - F_G^{\ell} \quad (3.15)$$

in order to obtain a conservative discretization of (3.4). F_G^{ℓ} is the discrete total energy flux F_G in cell Ω^{ℓ} in the above equation.
Using (3.11), i.e $f^{\ell+\frac{1}{2}} = \partial_{\mathbf{p}}(v_1 L)^{(\ell+\frac{1}{2})}$, we get the following:

$$-\partial_{\mathbf{p}}(vL)^{\ell+\frac{1}{2}} \cdot (\mathbf{p}^{\ell+1} - \mathbf{p}^{\ell}) + \mathbf{p}^{\ell+1} \cdot f^{\ell+1} - \mathbf{p}^{\ell} \cdot f^{\ell} = \mathbf{p}^{\ell+1} \cdot f^{\ell+1} - (vL)^{\ell+1} - \mathbf{p}^{\ell} \cdot f^{\ell} + (vL)^{\ell}. \quad (3.16)$$

Also, the numerical flux $f^{\ell+\frac{1}{2}}$ must verify the Roe-type property,

$$\mathbf{f}^{\ell+\frac{1}{2}} \cdot (\mathbf{p}^{\ell+1} - \mathbf{p}^{\ell}) = \partial_{\mathbf{p}}(vL) \cdot (\mathbf{p}^{\ell+1} - \mathbf{p}^{\ell}) = (vL)^{\ell+1} - (vL)^{\ell}. \quad (3.17)$$

Now, we will take into consideration the basic ideas of path conservative schemes, the path integral of the flux $\mathbf{f} = \partial_{\mathbf{p}}(vL)$ satisfy the identity

$$(vL)^{\ell+1} - (vL)^{\ell} = \int_{\mathbf{p}^{\ell}}^{\mathbf{p}^{\ell+1}} \partial_{\mathbf{p}}(vL) \cdot d\mathbf{p} = \int_0^1 \partial_{\mathbf{p}}(vL) \cdot \frac{\partial \boldsymbol{\psi}}{\partial s} ds, \quad (3.18)$$

for any path $\boldsymbol{\psi}, s \in [0, 1]$ in phase space. We chose the simple straight line segment path in $\mathbf{p}$ variables, i.e.

$$\boldsymbol{\psi}(s) = \mathbf{p}^{\ell} + s(\mathbf{p}^{\ell+1} - \mathbf{p}^{\ell}), \quad \frac{\partial \boldsymbol{\psi}}{\partial s} = \mathbf{p}^{\ell+1} - \mathbf{p}^{\ell}, \quad 0 \leq s \leq 1. \quad (3.19)$$

Inserting the segment path (3.19) in (3.18) gives us the following:

$$(vL)^{\ell+1} - (vL)^{\ell} = \left(\int_0^1 \mathbf{f}(\boldsymbol{\psi}(s)) ds \right) \cdot (\mathbf{p}^{\ell+1} - \mathbf{p}^{\ell}). \quad (3.20)$$

As a result, the thermodynamically compatible numerical flux for the inviscid Euler subsystem

$$\mathbf{f}_{\mathbf{p}}^{\ell+\frac{1}{2}} = \int_0^1 \mathbf{f}(\boldsymbol{\psi}(s))ds = \left(f_{\rho}^{\ell+\frac{1}{2}}, \mathbf{f}_{\rho\mathbf{v}}^{\ell+\frac{1}{2}}, f_{\rho S}^{\ell+\frac{1}{2}}\right)^T \tag{3.21}$$

As we have the exact form of the equations, we can employ them and do a quantitative study on the total energy conservation for the Euler subsystem.

3.8 Numerical Flux

We try to find a numerical flux $\mathbf{f}$ that satisfies

$$\mathbf{f} \cdot [[\mathbf{p}]] = [[pv]] \tag{3.22}$$

where $[[.]]$ denotes the jump between the two states. We will denote the arithmetic average by an overbar.

$$\overline{(.)} = \frac{1}{2}[(.)_l + (.)_r] \tag{3.23}$$

3.8.1 Approach

Write all jump terms in terms of jumps in ρ, T, v where $T = \frac{p}{\rho}$. Now, we know that

$$S = ln(p) - \gamma ln(\rho) = ln(\rho) + ln(T) - \gamma ln(\rho) = -(\gamma - 1)ln(\rho) + ln(T) \tag{3.24}$$

From this, we get

$$[[S]] = -(\gamma - 1)\frac{[[\rho]]}{\hat{\rho}} + \frac{[[T]]}{\hat{T}} \tag{3.25}$$

Where we define the logarithmic average as follows:

$$\hat{\phi} = \frac{[[\phi]]}{[[ln(\phi)]]} \tag{3.26}$$

Then, we have the following equation:

$$[[p_1]] = \frac{\gamma}{\gamma-1}[[T]] - \frac{1}{\gamma-1}[[TS]] - \overline{v}[[v]]$$

$$= \frac{\gamma-\overline{S}}{\gamma-1}[[T]] - \frac{\overline{T}}{\gamma-1}[[S]] - \overline{v}[[v]]$$

$$= \left(\frac{\gamma-\overline{S}-\frac{\overline{T}}{\hat{T}}}{\gamma-1}\right)[[T]] - \overline{v}[[v]] + \frac{\overline{T}}{\hat{\rho}}[[\rho]]$$

$$[[p_2]] = [[v]]$$

$$[[p_3]] = \frac{1}{\gamma-1}[[T]]$$

$$[[pv]] = [[\rho v T]]$$

$$= \overline{\rho v}[[T]] + \overline{T}[[pv]]$$

$$= \overline{\rho v}[[T]] + \overline{T}\,\overline{\rho}[[v]] + \overline{T}\,\overline{v}[[\rho]]$$

Thus, finally, we get,

$$f_1 = \hat{\rho}\overline{v}, \quad f_2 = \overline{\rho}\,\overline{T} + \overline{v} f_1, \quad f_3 = (\gamma-1)\overline{\rho v} - \left(\gamma - \overline{S} - \frac{\overline{T}}{\hat{T}}\right) f_1 \tag{3.27}$$

Chapter 4

Result Analysis and Discussion

4.1 Algorithm

The code was implemented in Python programming language and has the following algorithmic structure:

Algorithm 1 Algorithm for Semi-Discrete FVS

- Discretize the space by creating the numerical grid
- Set the initial conditions
- Main timestep evolution loop
  - Compute the timestep
  - Loop to advance one step (count depends on the number of stages in the integrator)
    - Reconstruct the state to interfaces
    - Find the fluxes through the interface
    - Do a conservative update of the state to the stage
- Output

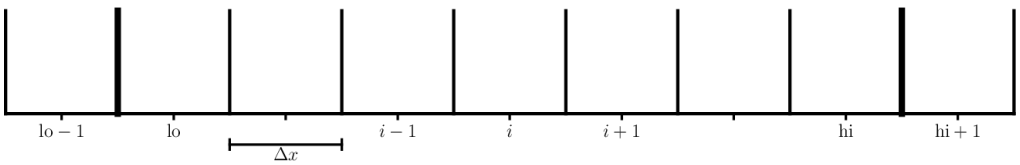

Figure 4.1: Visual Image of the Grid

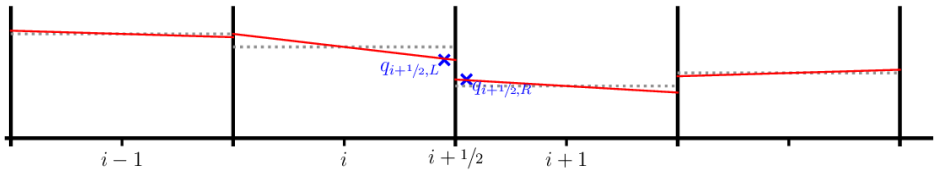

Figure 4.2: Visual Image of the states at the interfaces

4.1.1 Implementation Details

We manage our 1-d grid via a class FVGrid. We divide the domain into several zones (or volumes) that will store the state. We traditionally use ghost cells to implement boundary conditions–extra cells added to each end of the domain. We use *lo* and *hi* to refer to our domain's first and last zone. We need to use the cell averages to determine the fluid state on the interfaces. We'll reconstruct the cell averages as piecewise lines that give us the same average in the zone. We then follow these lines to the interfaces to define each interface's left and right states. Usually, we work in terms of the primitive variables, $q = (\rho, u, p)$ So, we first write a routine to do the algebraic transformation from conservative to primitive variables:

$$\rho = \rho$$

$$u = \frac{\rho u}{\rho}$$

$$p = \left((\rho E) - \frac{1}{2} \frac{(\rho u)^2}{\rho} \right) (\gamma - 1)$$

Next, we need a routine to create the interface states. Here we'll construct a slope for each zone, δq, based on the average state in the neighboring zones. This gives us a line representing the value of the fluid state as a function of position in each zone:

$$q_i(x) = \langle q \rangle_i + \frac{\Delta q_i}{\Delta x}(x - x_i)$$

Note that there is a unique $q_i(x)$ for each zone—this is usually called piecewise linear reconstruction. By design, the average of $q_i(x)$ over the zone is the cell average, so it is conservative.

We use this equation for a line to find the fluid state right at the interface. For zone i, the line $q_i(x)$ gives us the right state on the left interface, $q_{i+\frac{1}{2},R}$, and the left state on the right interface, $q_{i+\frac{1}{2},L}$.
There's one additional wrinkle—2^{nd} order codes tend to produce oscillations near discontinuities, so we usually need to limit the slopes, Δq_i , so we don't introduce new minima or maxima in the evolution. We'll use the minmod limiter (A.2):

$$\frac{\delta a}{\delta x} = minmod_i \left(\frac{a_i - a_{i-1}}{\Delta x}, \frac{a_{i+1} - a_i}{\Delta x} \right)$$

with

$$minmod(a, b) = \begin{cases} a \text{ if } |a| < |b| \text{ and } a.b > 0 \\ b \text{ if } |a| > |b| \text{ and } a.b > 0 \\ 0 \text{ otherwise} \end{cases}$$

After doing our reconstruction, we are left with a left and right state on an interface. To find the unique fluid state on the interface, we solve a Riemann problem,

$$q_{i+\frac{1}{2}} = \mathcal{R} \left(q_{i+\frac{1}{2},L}, q_{i+\frac{1}{2},R} \right)$$

Once we have the interface state, we can compute the fluxes using this state. Also, the time stamp is calculated as we cannot allow information to move more than one zone per step. The update in the driver code looks

like this:

$$U^* = U^n + \frac{\Delta t}{2} A(U^n)$$

$$U^{n+1} = U^n + \Delta t A(U^*)$$

4.2 Results

The ratio of specific heat of the fluid is chosen to be $\gamma = 1.4$. For all problems, the spatial domain is the interval [0,1] which is discretized with $M = 100$ computing cells. The codes were run on various problems like Sod Problem, Lax Problem [19], etc., with different initial conditions, parameters, and grids.

The initial conditions of the test problems are given in the table below:

Initial Conditions for Various Problems						
Test Name	ρ_L	u_L	p_L	ρ_R	u_R	p_R
Sod Problem	1.0	0.0	1.0	0.125	0.0	0.1
Lax Problem	0.445	0.698	3.528	0.5	0.0	0.571
Strong Shock Problem	1.0	0.0	1000.0	1.0	0.0	0.01
Woodward and Colella Problem	1.0	0.0	0.01	1.0	0.0	100.0

Table 4.1: Initial conditions of the test problems

4.2.1 Sod Problem

The Sod problem is a standard test problem consisting of a left and right state separated by an initial discontinuity. As time evolves, a rightward moving shock, contact, and leftward moving rarefaction form. The curve obtained with our scheme matches the exact solution. The plot shown below is drawn for the point of discontinuity $x = 0.5$ at output time 0.15 units:

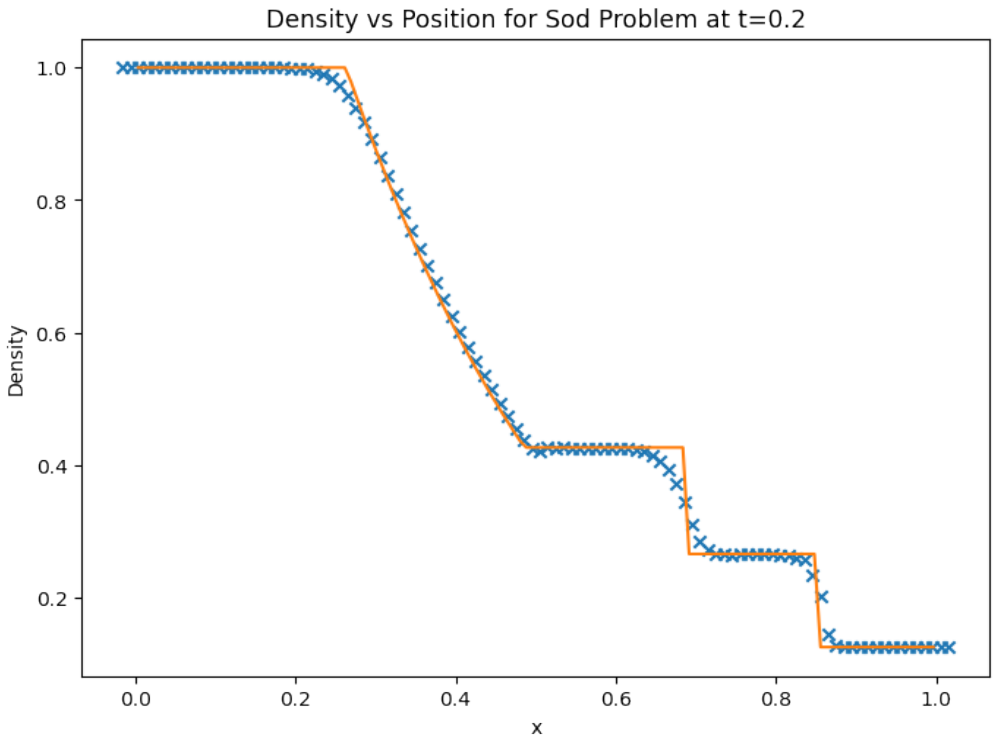

Figure 4.3: Density vs. Position for the Sod problem

The solid line gives the exact solution, while the solution is obtained through our schemes. We can see that it matches the exact solution.

4.2.2 Lax Problem

The initial conditions are given in the table above. The plots obtained through our scheme and the exact solution are given below. The plots are drawn for the point of discontinuity $x = 0.5$ at the output time 0.13 units.

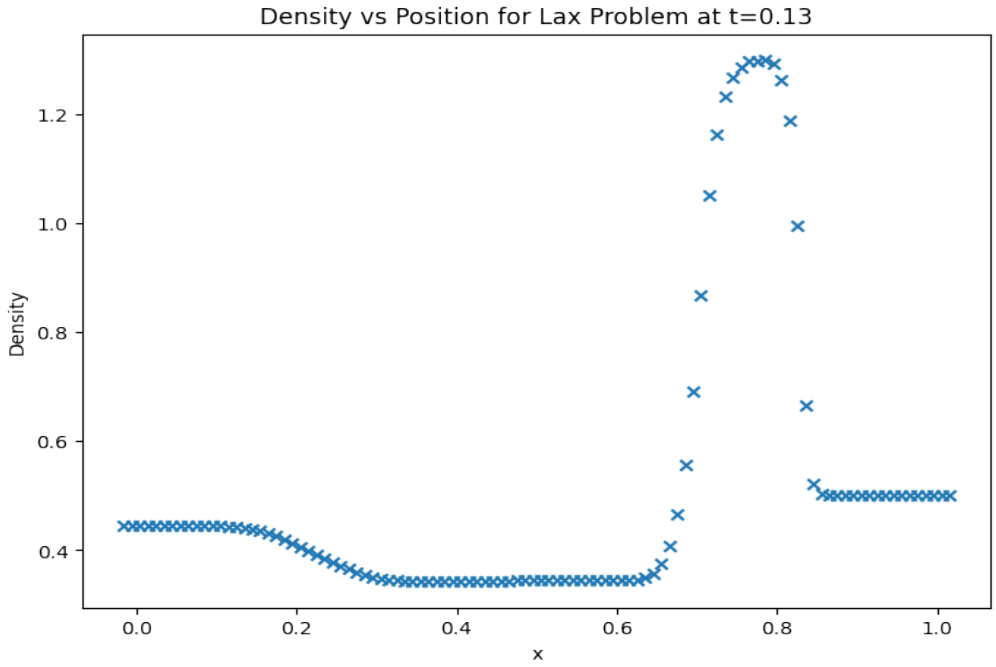

Figure 4.4: Density vs. Position for the Lax problem

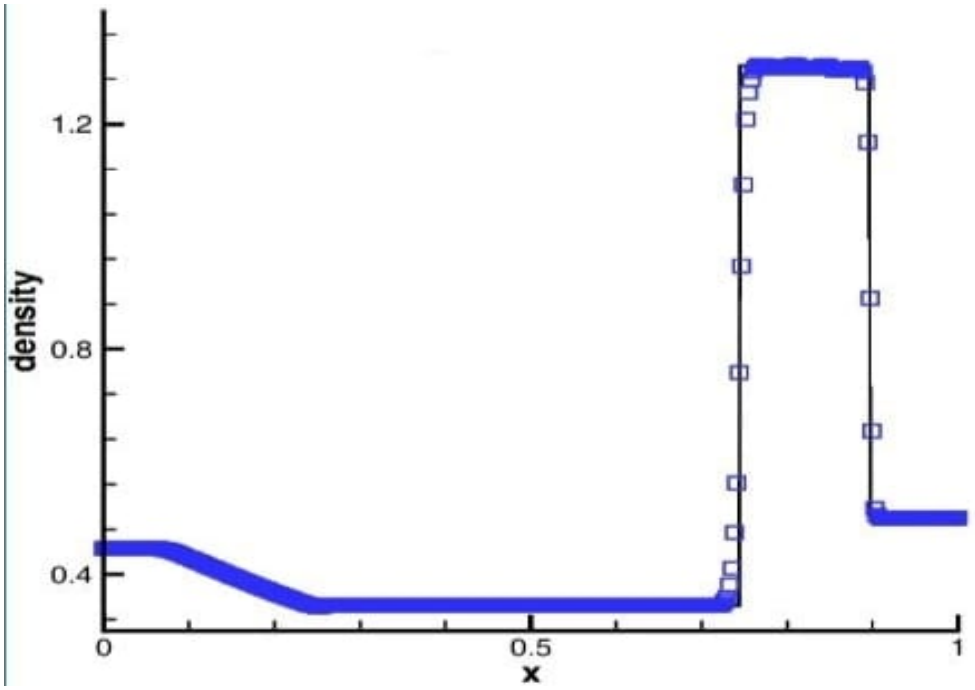

Figure 4.5: Exact plot of Density vs. Position for the Lax problem

In this case, as well, our scheme matches the exact solution.

4.2.3 Strong Shock Problem

The above table contains the initial conditions. The plots obtained by our approach, as well as the exact solution, are shown below. The plots are drawn for the point of discontinuity $x = 0.5$ at the output time 0.012 units.

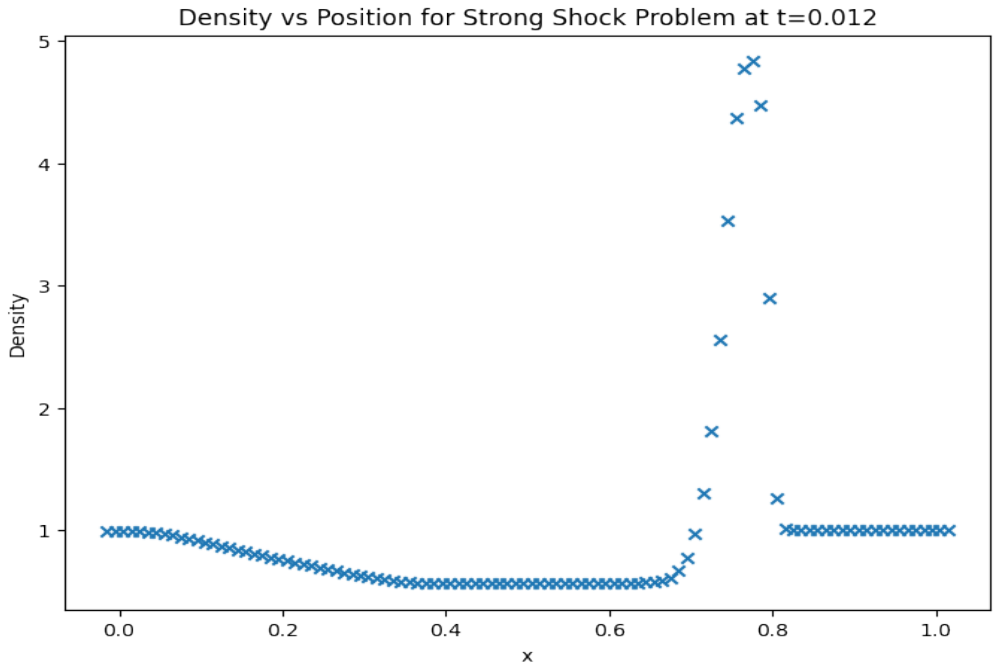

Figure 4.6: Density vs. Position for the Strong Shock problem

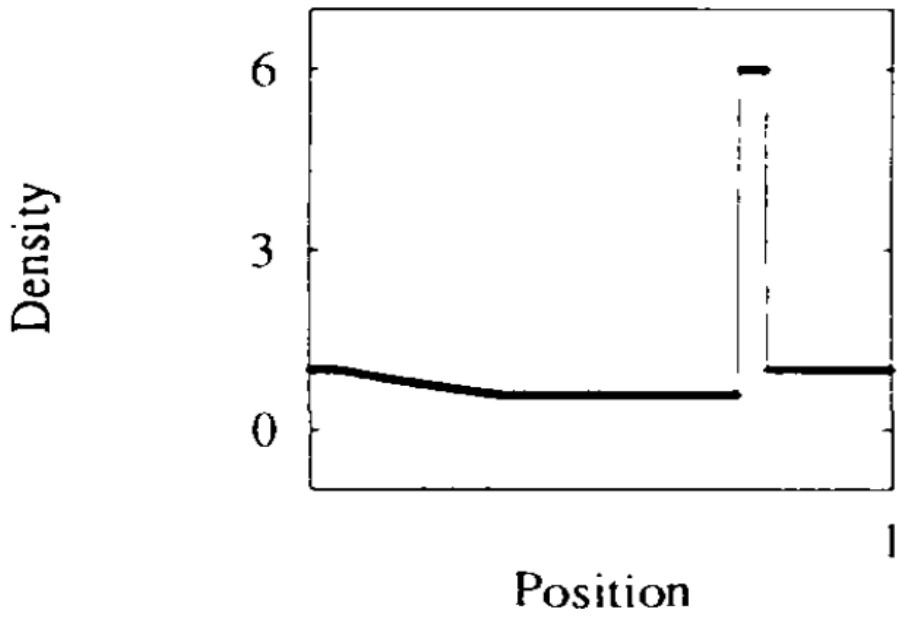

Figure 4.7: Exact plot of Density vs. Position for the Strong Shock problem

We see that the plot acquired by our approach is identical to the exact solution.

4.2.4 Woodward and Colella Problem

The starting circumstances are outlined in the table that can be found above. The plots that were generated by our approach, as well as the precise solution, can be found below. The plots are drawn for the point of discontinuity $x = 0.5$ at the output time 0.035 units.

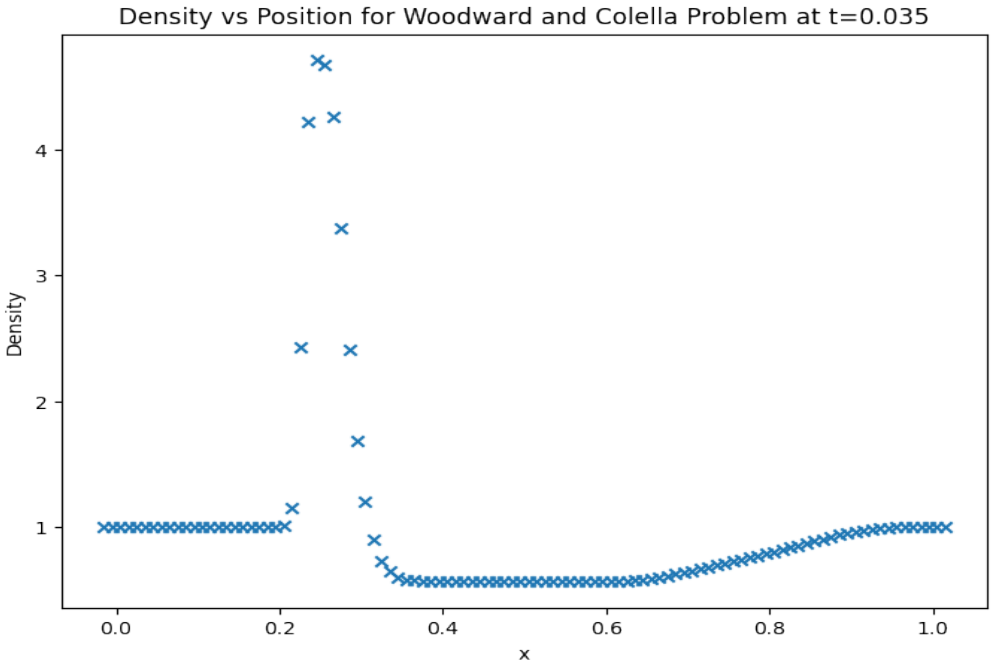

Figure 4.8: Density vs. Position for the Woodward and Colella problem

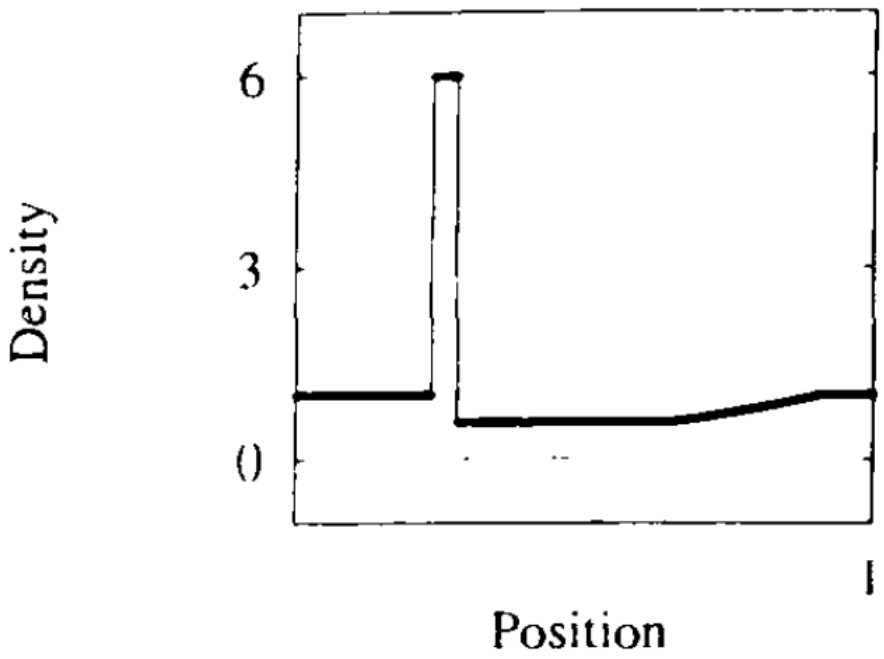

Figure 4.9: Exact plot of Density vs. Position for the Woodward and Colella problem

Also, in this instance, our plan corresponds precisely with the exact solution. We observe that the numerical solution obtained through our scheme is convergent and consistent with the exact solutions of various test problems.

4.3 Compatible Discretization of the Euler system in 1D

The inviscid Euler subsystem (3.1) - (3.3) with the related extra conservation law for the total energy density reads

$$\partial_t \mathbf{q} + \partial_x \mathbf{f}_1(\mathbf{q}) = 0 \tag{4.1}$$

$$\partial_t \mathcal{E}^{12} + \partial_x F_1^{12} = 0 \tag{4.2}$$

A semi-discrete finite volume scheme for the above is given by

$$\frac{d}{dt}\mathbf{q}^\ell = -\frac{F^{\ell+\frac{1}{2}} - F^{\ell-\frac{1}{2}}}{\Delta x} = -\frac{\left(F^{\ell+\frac{1}{2}} - \mathbf{f}^\ell\right) - \left(F^{\ell-\frac{1}{2}} - \mathbf{f}^\ell\right)}{\Delta x} = -\frac{D_-^{\ell+\frac{1}{2}} + D_+^{\ell+\frac{1}{2}}}{\Delta x} \tag{4.3}$$

with $\mathbf{f}^\ell = \mathbf{f}_1(\mathbf{q}^\ell)$ to ease the notation, $\mathbf{f}_1(\mathbf{q}) = (\rho v_1, \rho v_i v_1 + p\delta_{i1}, \rho S v_1)$ the fluxes of the Euler subsystem and the associated total energy of the Euler subsystem $F_1^{12} = v_1(\mathcal{E}_1 + \mathcal{E}_2 + p)$ and the numerical flux $F^{\ell+\frac{1}{2}}$. The numerical fluxes and the fluctuations are related as follows:

$$D_-^{\ell+\frac{1}{2}} = F^{\ell+\frac{1}{2}} - \mathbf{f}^\ell, \quad D_+^{\ell-\frac{1}{2}} = \mathbf{f}^\ell - F^{\ell-\frac{1}{2}} \tag{4.4}$$

In order to get the total energy conservation law as a result of the discretization of the above equations, we compute the dot product of the discrete

dual variable, $\mathbf{p}^\ell = \partial_q \mathcal{E}(\mathbf{q}^\ell)$, with the above semi-discrete scheme (4.3),

$$\mathbf{p}^\ell \cdot \frac{d}{dt}\mathbf{q}^\ell = \frac{d}{dt}\mathcal{E}^\ell = -\mathbf{p}^\ell \cdot \frac{\left(F^{\ell+\frac{1}{2}} - \mathbf{f}^\ell\right) - \left(F^{\ell-\frac{1}{2}} - \mathbf{f}^\ell\right)}{\Delta x} = -\frac{D_{\epsilon-}^{\ell+\frac{1}{2}} + D_{\epsilon,+}^{\ell+\frac{1}{2}}}{\Delta x} \tag{4.5}$$
$$= -\frac{F_1^{12,\ell+\frac{1}{2}} - F_1^{12,\ell-\frac{1}{2}}}{\Delta x}$$

We now find a suitable numerical flux $F^{\ell+\frac{1}{2}}$ that is consistent with the thermodynamic compatibility of the finite volume scheme (4.3) and with the discrete form of the total energy conservation law. In order to find it, we define the left and the right energy fluctuations as

$$D_{\epsilon,-}^{\ell+\frac{1}{2}} = \mathbf{p}^\ell \cdot D_-^{\ell+\frac{1}{2}} = \mathbf{p}^\ell \cdot (F^{\ell+\frac{1}{2}} - \mathbf{f}^\ell), \quad D_{\epsilon,+}^{\ell-\frac{1}{2}} = \mathbf{p}^\ell \cdot D_+^{\ell+\frac{1}{2}} = \mathbf{p}^\ell \cdot (\mathbf{f}^\ell - F^{\ell-\frac{1}{2}}) \tag{4.6}$$

which must satisfy the consistency property,

$$D_{\epsilon,-}^{\ell+\frac{1}{2}} + D_{\epsilon,+}^{\ell+\frac{1}{2}} = \mathbf{p}^\ell \cdot D_-^{\ell+\frac{1}{2}} + \mathbf{p}^{\ell+1} \cdot D_+^{\ell+\frac{1}{2}} = \mathbf{p}^\ell \cdot (F^{\ell-\frac{1}{2}} - \mathbf{f}^\ell) + \mathbf{p}^{\ell+1} \cdot (\mathbf{f}^{\ell+1} - F^{\ell+\frac{1}{2}}) \tag{4.7}$$
$$= F_1^{12,\ell+1} - F_1^{12,\ell}$$

so as to obtain a conservative discretization of the extra conservation law. The energy fluctuations $F_1^{l2,\ell\pm\frac{1}{2}}$ and the numerical total energy fluxes as

$$D_{\epsilon,-}^{\ell+\frac{1}{2}} = F_1^{12,\ell+\frac{1}{2}} - F_1^{12,\ell}, \quad D_{\epsilon,+}^{\ell-\frac{1}{2}} = F_1^{12,\ell} - F_1^{12,\ell+\frac{1}{2}} \tag{4.8}$$

with $F_1^{12,\ell} = F_1^{12,\ell}$ the discrete total energy flux related to the Euler subsystem evaluated in cell Ω^ℓ. We assume the thermodynamic-compatible numerical flux $F^{\ell+\frac{1}{2}}$ to have the rather general form

$$F^{\ell+\frac{1}{2}} = \tilde{F}^{\ell+\frac{1}{2}} - \alpha^{\ell+\frac{1}{2}}\left(p^{\ell+1} - p^\ell\right) = \left(F_\rho^{\ell+\frac{1}{2}}, F_{\rho v}^{\ell+\frac{1}{2}}, F_{\rho S}^{\ell+\frac{1}{2}}\right) \tag{4.9}$$

where $\tilde{F}^{\ell+\frac{1}{2}}$ could be, in principle, any central numerical that does not necessarily guarantee discrete thermodynamic compatibility of the system of the conservation laws with the extra conservation law, and thus is corrected using a suitable scale parameter $\alpha^{\ell+\frac{1}{2}}$ to guarantee the discrete thermodynamic compatibility. Using the equation (4.7) on the flux equation (4.9), we get

$$\mathbf{p}^{\ell}\cdot(\tilde{F}^{\ell+\frac{1}{2}}-\mathbf{f}^{\ell})+\mathbf{p}^{\ell+1}\cdot(\mathbf{f}^{\ell+1}-\tilde{F}^{\ell+\frac{1}{2}}-\alpha^{\ell+\frac{1}{2}}\mathbf{p}^{\ell}\cdot(\mathbf{p}^{\ell+1}-\mathbf{p}^{\ell})+\alpha^{\ell+\frac{1}{2}}\mathbf{p}^{\ell+1}\cdot(\mathbf{p}^{\ell+1}-\mathbf{p}^{\ell}) \tag{4.10}$$
$$= F_1^{12,\ell+1} - F_1^{12,\ell}$$

On equating the above equation and rearranging the terms, we get

$$-\tilde{F}^{\ell+\frac{1}{2}}\cdot(\mathbf{p}^{\ell+1}-\mathbf{p}^{\ell})+\mathbf{p}^{\ell+1}\cdot\mathbf{f}^{\ell+1}-\mathbf{p}^{\ell}\cdot\mathbf{f}^{\ell}+\alpha^{\ell+\frac{1}{2}}(\mathbf{p}^{\ell+1}-\mathbf{p}^{\ell})^2 = F_1^{12,\ell+1} - F_1^{12,\ell} \tag{4.11}$$

From the above equation, we can have the formula of the scalar correlation factor $\alpha^{\ell+\frac{1}{2}}$

$$\alpha^{\ell+\frac{1}{2}} = \frac{F_1^{12,\ell+1} - F_1^{12,\ell} + \tilde{F}^{\ell+\frac{1}{2}}\cdot(\mathbf{p}^{\ell+1}-\mathbf{p}^{\ell}) - (\mathbf{p}^{\ell+1}\cdot\mathbf{f}^{\ell+1}-\mathbf{p}^{\ell}\cdot\mathbf{f}^{\ell})}{(\mathbf{p}^{\ell+1}-\mathbf{p}^{\ell})^2} \tag{4.12}$$

Using this correction factor α, we can guarantee the discrete thermodynamic compatibility of the scheme (4.3) with the extra energy conservation law. But, here, a problem arises when the denominator value is 0, i.e., $(p^{\ell+1}-p^{\ell})^2 = 0$. In that case, we set $\alpha^{\ell+\frac{1}{2}} = 0$. Here, the dissipation-less central flux is used as the underlying numerical flux, i.e.,

$$\tilde{F}^{\ell+\frac{1}{2}} = \frac{1}{2}\left(\mathbf{f}^{\ell}+\mathbf{f}^{\ell+1}\right) \tag{4.13}$$

as suitable numerical dissipation terms that are compatible with the first and second laws of thermodynamics.

4.4 Thermodynamically Compatible discretization of Dissipation Terms in 1D

In order to obtain a dissipative thermodynamically compatible scheme, we need to add a compatible numerical dissipation to the inviscid flux derived in the previous section.

We will add an additional dissipative flux and a corresponding entropy production term to the scheme (4.3) as follows:

$$\frac{d}{dt}\mathbf{q}^{\ell} + \frac{\mathbf{f}^{\ell+\frac{1}{2}} - \mathbf{f}^{\ell-\frac{1}{2}}}{\Delta x} = \frac{\mathbf{g}^{\ell+\frac{1}{2}} - \mathbf{g}^{\ell-\frac{1}{2}}}{\Delta x} + \mathbf{P}^{\ell} \tag{4.14}$$

The dissipative part of the numerical flux is taken of the form

$$\mathbf{g}^{\ell+\frac{1}{2}} = \epsilon^{\ell+\frac{1}{2}} \frac{\Delta \mathbf{q}^{\ell+\frac{1}{2}}}{\Delta x}, \quad \Delta \mathbf{q}^{\ell+\frac{1}{2}} = \mathbf{q}^{\ell+1} - \mathbf{q}^{\ell} \tag{4.15}$$

where the scalar numerical dissipation coefficient can be either taken as a constant, $\epsilon^{\ell+\frac{1}{2}} = \epsilon$ or of the form

$$\epsilon^{\ell+\frac{1}{2}} = \frac{1}{2}\left(1 - \phi^{\ell+\frac{1}{2}}\right) \Delta x s_{max}^{\ell+\frac{1}{2}} \geq 0 \tag{4.16}$$

where $s_{max}^{\ell+\frac{1}{2}}$ is the maximum signal speed at the cell interface (Rusanov flux) and $\phi^{\ell+\frac{1}{2}}$ is a flux limiter that permits the numerical dissipation to be reduced in smooth regions. We employ the Minbee flux limiter (A.2) in this work, which reads

$$\phi^{\ell+\frac{1}{2}} = min\left(\phi_{-}^{\ell+\frac{1}{2}}, \phi_{+}^{\ell+\frac{1}{2}}\right), with \quad \phi_{\pm} = max\left(0, min\left(1, h_{\pm}^{\ell+\frac{1}{2}}\right)\right) \tag{4.17}$$

where the ratio of density slopes are

$$h_{-}^{\ell+\frac{1}{2}} = \frac{\rho^{\ell} - \rho^{\ell-1}}{\rho^{\ell+1} - \rho^{\ell}}, \quad h_{+}^{\ell+\frac{1}{2}} = \frac{\rho^{\ell+2} - \rho^{\ell+1}}{\rho^{\ell+1} - \rho^{\ell}} \tag{4.18}$$

We will now do the dot product of $\mathbf{p}^{\ell}$ with the updated equation (including the dissipative flux and entropy production terms), which leads to the following equation:

$$\frac{d\mathcal{E}^{\ell}}{dt} + \frac{1}{\Delta x}\left(F_G^{\ell+\frac{1}{2}} - F_G^{\ell-\frac{1}{2}}\right) = \frac{1}{\Delta x}\mathbf{p}^{\ell} \cdot \left(\mathbf{g}^{\ell+\frac{1}{2}} - \mathbf{g}^{\ell-\frac{1}{2}}\right) + \mathbf{p}^{\ell} \cdot \mathbf{P}^{\ell} \tag{4.19}$$

using the numerical energy flux derived from the total energy fluctuation at the interface as $F_G^{\ell+\frac{1}{2}} = D_{\epsilon}^{\ell+\frac{1}{2},-} + F_G^{\ell}$. Because the thermodynamic compatibility of the left-hand side has already been explored in the previous section, we can now concentrate on the right-hand side of the equation.

$$\mathbf{p}^{\ell} \cdot \mathbf{P}^{\ell} + \frac{1}{\Delta x}\mathbf{p}^{\ell} \cdot \left(\mathbf{g}^{\ell+\frac{1}{2}} - \mathbf{g}^{\ell-\frac{1}{2}}\right) \tag{4.20}$$

$$= \mathbf{p}^{\ell} \cdot \mathbf{P}^{\ell} + \frac{1}{\Delta x}\left(\frac{1}{2}\mathbf{p}^{\ell} \cdot \mathbf{g}^{\ell+\frac{1}{2}} + \frac{1}{2}\mathbf{p}^{\ell+1} \cdot \mathbf{g}^{\ell+\frac{1}{2}} + \frac{1}{2}\mathbf{p}^{\ell} \cdot \mathbf{g}^{\ell+\frac{1}{2}} - \frac{1}{2}\mathbf{p}^{\ell+1} \cdot \mathbf{g}^{\ell+\frac{1}{2}}\right)$$

$$-\frac{1}{\Delta x}\left(\frac{1}{2}\mathbf{p}^{\ell} \cdot \mathbf{g}^{\ell-\frac{1}{2}} + \frac{1}{2}\mathbf{p}^{\ell-1} \cdot \mathbf{g}^{\ell-\frac{1}{2}} + \frac{1}{2}\mathbf{p}^{\ell} \cdot \mathbf{g}^{\ell-\frac{1}{2}} - \frac{1}{2}\mathbf{p}^{\ell-1} \cdot \mathbf{g}^{\ell-\frac{1}{2}}\right)$$

$$\begin{aligned} = \mathbf{p}^{\ell} \cdot \mathbf{P}^{\ell} &+ \frac{1}{2}\frac{\mathbf{p}^{\ell+1} + \mathbf{p}^{\ell}}{\Delta x} \cdot \epsilon^{\ell+\frac{1}{2}}\frac{\Delta q^{\ell+\frac{1}{2}}}{\Delta x} - \frac{1}{2}\frac{\mathbf{p}^{\ell} + \mathbf{p}^{\ell-1}}{\Delta x} \cdot \epsilon^{\ell-\frac{1}{2}}\frac{\Delta q^{\ell-\frac{1}{2}}}{\Delta x} \\ &- \frac{1}{2}\frac{\mathbf{p}^{\ell+1} - \mathbf{p}^{\ell}}{\Delta x} \cdot \epsilon^{\ell+\frac{1}{2}}\frac{\Delta q^{\ell+\frac{1}{2}}}{\Delta x} - \frac{1}{2}\frac{\mathbf{p}^{\ell} - \mathbf{p}^{\ell-1}}{\Delta x} \cdot \epsilon^{\ell-\frac{1}{2}}\frac{\Delta q^{\ell-\frac{1}{2}}}{\Delta x} \end{aligned} \tag{4.21}$$

Also, We know the following identity

$$\int_{\mathbf{q}^\ell}^{\mathbf{q}^{\ell+1}} \mathbf{p} \cdot d\mathbf{q} = \int_{\mathbf{q}^\ell}^{\mathbf{q}^{\ell+1}} \partial_{\mathbf{q}} \mathcal{E} \cdot d\mathbf{q} = \mathcal{E}^{\ell+1} - \mathcal{E}^{\ell} = \Delta \mathcal{E}^{\ell+\frac{1}{2}} \tag{4.22}$$

We can now think of the term $\frac{1}{2}(p^{\ell+1} + p^{\ell}) \cdot \Delta \mathbf{q}^{\ell+\frac{1}{2}}$ as an approximation of the total energy density $\Delta \mathcal{E}^{\ell+\frac{1}{2}}$, where the path integral has been calculated using the trapezoidal rule. Thus, using the above two obtained equations, the energy flux, including the convective and diffusion terms, is

$$F_d^{\ell+\frac{1}{2}} = F_G^{\ell+\frac{1}{2}} - \frac{1}{2}(\mathbf{p}^{\ell+1} + \mathbf{p}^{\ell}) \cdot \epsilon^{\ell+\frac{1}{2}} \frac{\Delta q^{\ell+\frac{1}{2}}}{\Delta x} \approx F_G^{\ell+\frac{1}{2}} - \epsilon^{\ell+\frac{1}{2}} \frac{\Delta \mathcal{E}^{\ell+\frac{1}{2}}}{\Delta x} \tag{4.23}$$

Algorithm 2 Algorithm for Semi-Discrete FVS including the correction factor and dissipative terms

- Discretize the space by creating the numerical grid
- Set the initial conditions
- Main timestep evolution loop
  - Compute the timestep.
  - Loop to advance one step (count depends on the number of stages in the integrator).
    - Reconstruct the state to interfaces.
    - Find the fluxes through the interface.
    - Update the flux using the correction factor, α.
    - Add the dissipative term in the flux of the state.
    - Do a conservative update of the state to the stage.
- Output

4.5 Results and Analysis

Let's look at an interesting problem with the initial conditions as follows:

$$p = 1$$
$$v = 1$$
$$\rho = 2 + Sin(2\pi x)$$

The exact solution for this problem is

$$\rho(x,t) = 2 + Sin(2\pi(x-t))$$

We studied this problem and used our scheme to discretize it. The 1D semi-discrete model was used with 100 cells, with a stopping time of order of 0.1 and 3 ghost cells on either side so as to get the periodic boundary conditions. The two plots, exact and estimate, were plotted in the figure given below.

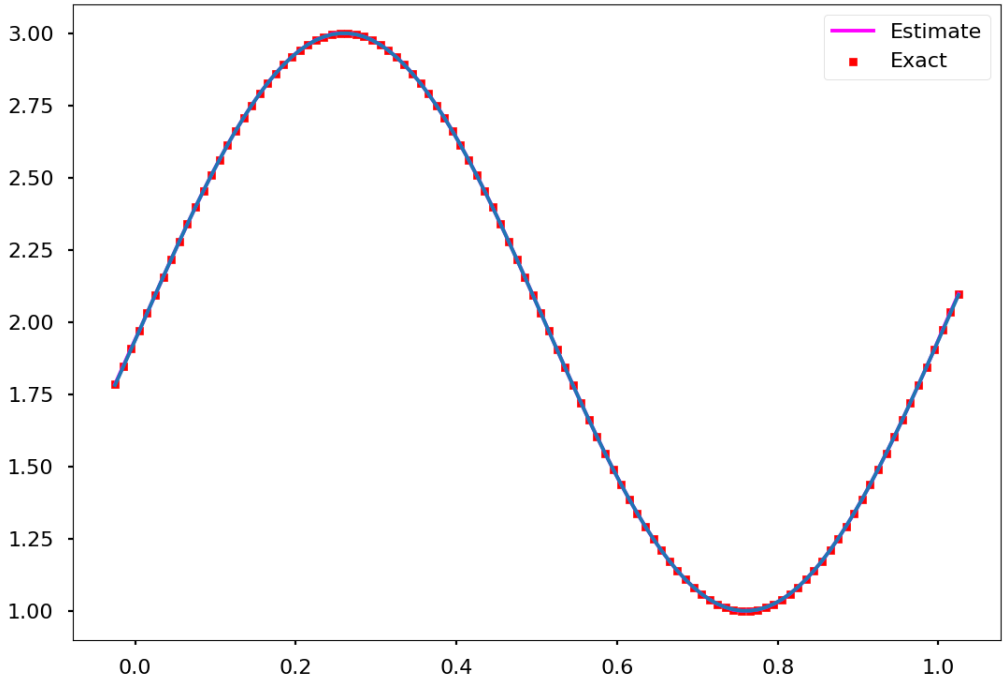

Figure 4.10: Density vs. Position for the above problem

The solid line represents the exact solution, while the cross-mark represents the solution through our scheme. We can see that the solution

obtained through o matches the exact solution.

4.5.1 Numerical Analysis of the above problem

The numerical order of convergence for the scheme is 2.0006141768554664, which is quite close to 2. The $L_2(\omega)$ error is calculated for the mesh size of 0.01. The value of the error is 0.0008647707644167368, which is of the order 10^{-4}.

Chapter 5

Conclusion

In conclusion, this thesis has presented a novel approach to solving the Euler equations of fluid dynamics by utilizing the entropy inequality instead of the traditional energy conservation law. By adopting this alternative framework, a stable FVM thermodynamic system was implemented, which showcased promising results in various test cases and demonstrated an accurate representation of fluid behavior. By focusing on the entropy function, which is a measure of system disorder, the algorithm ensures that the physical laws of thermodynamics are consistently satisfied throughout the simulations. The implemented finite volume scheme demonstrates remarkable accuracy and reliability in solving various fluid dynamics problems, including Riemann problems. The implemented finite volume scheme exhibits excellent accuracy and reliability, as evidenced by its ability to match exact solutions with an error of the order of 0.0001. This level of accuracy demonstrates the efficacy of the implemented FVM thermodynamic system in capturing the underlying physics of the problems considered.

In summary, this thesis successfully presents a stable FVM thermodynamic system based on the entropy inequality in the Euler equations. The order of convergence, numerical approximation, and error analysis have further validated the performance and reliability of the proposed algorithm. This research contributes to the advancement of computational fluid dynamics and opens up new possibilities for investigating complex fluid flow phenomena. The findings of this study contribute to the advancement of numerical methods in fluid dynamics and open new avenues for further research in the field.

We can extend this scheme to further include various relaxation terms and various energy production terms. Also, the scheme can be studied for multi-dimension space.

Appendix A

Some General Concepts

A.1 Finite Volume Method

In finite volume methods, we convert the PDE into a system of algebraic equations of cell averages. Finite Volume methods have a desirable property that the FVM solutions conserve mass, energy, etc., making them one of the first choices in numerical solutions of Computational Fluid Dynamics. Consider the advection equation

$$\frac{\partial u}{\partial t} + \frac{\partial f}{\partial x} = 0 \tag{A.1}$$

. For a particular cell i in the discretization of the domain, define the cell average as

$$\overline{u}_i(t) = \frac{1}{x_{i+1/2} - x_{i-1/2}} \int_{x_{i-1/2}}^{x_{i+1/2}} u(x,t)dx \tag{A.2}$$

After integrating A.1, the resulting equation is:

$$u(x,t_2) = u(x,t_1) - \int_{t_1}^{t_2} f_x dt$$

By integrating with respect to x between the limits of $x_{i-1/2}$ and $x_{i+1/2}$, then dividing by $\Delta x_i = x_{i+1/2} - x_{i-1/2}$ and utilizing the average equation A.2, as well as applying the divergence theorem, the resulting outcome is as follows:

$$\overline{u}_i(t_2) = \overline{u}_i(t_1) - \frac{1}{\Delta x_i}\left(\int_{t_1}^{t_2} f_{i+1/2}dt - \int_{t_1}^{t_2} f_{i+1/2}dt\right)$$

where $f_{i\pm1/2} = f(x_{i\pm1/2}, t)$

Therefore, it is possible to develop a semi-discrete scheme as

$$\frac{\partial \overline{u}_i}{\partial t} + \frac{1}{\Delta x_i}\left[f_{i+1/2} - f_{i-1/2}\right] = 0$$

This is an illustration of a finite volume scheme utilized in solving the advection equation.

A.2 Limiters

To accurately model convective fluid flow in hyperbolic equations, it is important to account for shocks and discontinuities. However, low-order schemes tend to be stable but dissipative near these points, while higher-order schemes are unstable and exhibit oscillations. To address this, a flux limiter can be used to create a high-resolution, stable, and accurate scheme. Some of the commonly employed limiters in FVM schemes are

A.2.1 MinMod Limitter

The minmod function is defined as

$$MinMod(\mathcal{M}) = \begin{cases} 0 & \exists a, b \in \mathcal{M} \quad s.t. \quad sgn(a) \neq sgn(b) \\ s \min_{a\in\mathcal{M}} |a| & s = sgn(a), a \in \mathcal{M} \end{cases} \tag{A.3}$$

A.2.2 Albada

The Albada limiter function is defined as

$$Albada(r) = \frac{r^2 + r}{r^2 + 1} \tag{A.4}$$

A.3 L2 Error

To quantify the approximation capability of an FVM, we calculate its L2 error which is defined as

$$\|u_{FVM} - u_{exact}\|^2 = \int_{\Omega} (u_{FVM} - u_{exact})^2 dx` \tag{A.5}$$

However, since in FVM, discretization of space is done, we calculate L^2 error by approximating the integral A.5 as

$$\|u_{FVM} - u_{exact}\|^2 = \sum_{n=1}^{N} (u^i_{FVM} - u^i_{exact})^2 \Delta x_i \tag{A.6}$$

A.4 Convergence Rate

A numerical method is said to have a convergence rate "p" if the error of the method is given by "Ch^p" where $h = \max_{i=1}^{N} \Delta x_i$ is the mesh size and C is a constant(not independent of function). To calculate the convergence rate of a numerical method from numerical experiments, we calculate the errors at mesh size $h, h/2, h/4$, and $h/8$.
Then the convergence rate is calculated as the average of these three values $((\log(e_h) - \log(e_{h/2})), (\log(e_{h/2}) - \log(e_{h/4})), (\log(e_{h/4}) - \log(e_{h/8})))$, where e_h denotes the L^2 error at a mesh size of h.

Appendix B

Code and Implementation

The 1D code for the given scheme is implemented in Python programming language. The code snippet for each block is given below. Each code block describes a function/class used in the implementation of the scheme in 1D.

Listing B.1: Importing Libraries

```
import numpy as np
from matplotlib import pyplot as plt
import math
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt
plt.style.use('seaborn-poster')
```

Listing B.2: Cell

```
class Cell:
def __init__(self):
    self.u = np.array([[0.0, 0.0, 0.0]] * (Config.NUM_RK_STEPS +
        1))
    # update in terms of rho, rho v and E
    self.uwest = np.array([0.0, 0.0, 0.0])
    self.uEast = np.array([0.0, 0.0, 0.0])
    self.cx = 0.0
    self.dx = 0.0
    self.totalFlux = np.array([0.0, 0.0, 0.0])
```

Listing B.3: Ghost Cell

```
class GhostCellUpdater:
@staticmethod
def updateGhostCells(cells, rkStep):
    for ghostCell in range(Config.NUM_GHOST_CELLS):
      cells[ghostCell].u[rkStep] = cells[Config.NUM_X_CELLS +
          ghostCell].u[rkStep]
      cells[Config.NUM_X_CELLS + Config.NUM_GHOST_CELLS +
          ghostCell].u[rkStep] = cells[Config.NUM_GHOST_CELLS +
          ghostCell].u[rkStep]
```

Listing B.4: Reconstruction Types

```
from enum import Enum
class ReconstructionTypes(Enum):
    FIRST_ORDER = 1
    LAX_WENDROFF = 2
    BEAM_WARMING = 3
    FROMM = 4
    LIMITED_LM = 5
```

Listing B.5: Config

```
class Config:
    NUM_X_CELLS = 100
    NUM_GHOST_CELLS = 3
    NUM_RK_STEPS = 1
    STOPPING_TIME = 0.01
    MAX_TIME_ITER = 1000
    MIN_X = 0.0
    MAX_X = 1.0
    ADVECTION_VEL = 1.0
```

```
    COURANT_NUM = 0.1
    RECONST_TYPE = ReconstructionTypes.LIMITED_LM
    gamma = 1.4
```

Listing B.6: Reconstruction Variables

```
class VariableReconstructor:
    @staticmethod
    def reconstructVariables(cells, rkStep):
        reconst_type = Config.RECONST_TYPE
        if reconst_type == ReconstructionTypes.FIRST_ORDER:
            for i in range(Config.NUM_GHOST_CELLS - 1,
                Config.NUM_X_CELLS + Config.NUM_GHOST_CELLS + 1):
                cells[i].uWest = cells[i].u[rkStep]
                cells[i].uEast = cells[i].u[rkStep]
        elif reconst_type == ReconstructionTypes.BEAM_WARMING:
            for i in range(Config.NUM_GHOST_CELLS - 1,
            Config.NUM_X_CELLS + Config.NUM_GHOST_CELLS + 1):
                du_dx = (cells[i].u[rkStep] - cells[i -
                    1].u[rkStep]) / cells[i].dx
                cells[i].uWest = cells[i].u[rkStep] - du_dx *
                    cells[i].dx / 2.0
                cells[i].uEast = cells[i].u[rkStep] + du_dx *
                    cells[i].dx / 2.0
        elif reconst_type == ReconstructionTypes.LAX_WENDROFF:
            for i in range(Config.NUM_GHOST_CELLS - 1,
                Config.NUM_X_CELLS + Config.NUM_GHOST_CELLS + 1):
                du_dx = (cells[i+1].u[rkStep] - cells[i].u[rkStep])
                    / cells[i].dx
                cells[i].uWest = cells[i].u[rkStep] - du_dx *
                    cells[i].dx / 2.0
```

```
            cells[i].uEast = cells[i].u[rkStep] + du_dx *
                cells[i].dx / 2.0
        elif reconst_type == ReconstructionTypes.FROMM:
            for i in range(Config.NUM_GHOST_CELLS - 1,
                Config.NUM_X_CELLS + Config.NUM_GHOST_CELLS + 1):
                du_dx = (cells[i+1].u[rkStep] -
                    cells[i-1].u[rkStep]) / 2.0 / cells[i].dx
                cells[i].uWest = cells[i].u[rkStep] - du_dx *
                    cells[i].dx / 2.0
                cells[i].uEast = cells[i].u[rkStep] + du_dx *
                    cells[i].dx / 2.0
        elif reconst_type == ReconstructionTypes.LIMITED_LM:
            for i in range(Config.NUM_GHOST_CELLS - 1,
                Config.NUM_X_CELLS + Config.NUM_GHOST_CELLS + 1):
                r = (cells[i].u[rkStep] - cells[i-1].u[rkStep]) /
                    (cells[i+1].u[rkStep] - cells[i].u[rkStep] +
                    0.000001)
                r = max(0, r.max())
                phi = (r*r + r) / (r*r + 1) # Alba Limiter
                du_dx = (cells[i+1].u[rkStep] - cells[i].u[rkStep])
                    / cells[i].dx
                cells[i].uWest = cells[i].u[rkStep] - phi * du_dx *
                    cells[i].dx / 2.0
                cells[i].uEast = cells[i].u[rkStep] + phi * du_dx *
                    cells[i].dx / 2.0
```

Listing B.7: Variable Copier

```
class VariableCopier:
    def CopyToZerothRKStep(cells):
        for i in range(Config.NUM_GHOST_CELLS,
```

```
            Config.NUM_X_CELLS+Config.NUM_GHOST_CELLS,1):
            cells[i].u[0] = cells[i].u[Config.NUM_RK_STEPS]
```

Listing B.8: Pressure Calculator

```
e_ref = 1.0
Cv = 1/(Config.gamma - 1)
def calculatePressure(cell):
    rho = cell.u[0][0]
    u = cell.u[0][1] / rho
    S = cell.u[0][2] / rho
    return (rho ** Config.gamma) * np.exp(S/Cv)
```

Listing B.9: Time Step Calculator

```
class TimeStepCalculator:
    @staticmethod
    def getTimeStep(cells):
        for cell in cells:
            cell.u = np.nan_to_num(cell.u, copy=True,
            nan=1e-5, posinf=None, neginf=None)
        press = [calculatePressure(cell) for cell in cells]
        press = np.array([v if v <1e10 else 0 for v in press])
        rho = np.array([cell.u[0][0] for cell in cells])
        soundSpeed = np.sqrt(Config.gamma * press/(rho + 1e-5))
        soundSpeed = np.nan_to_num(soundSpeed, copy = True,
        nan = 10)
        v = [0 if cell.u[0][0] > 1e3
        else cell.u[0][1]/cell.u[0][0] for cell in cells]
        v = np.array([v if np.abs(v) < 1e-5 else 0 for v in v])
        return [Config.COURANT_NUM * cells[0].dx * ((np.abs(v) +
            soundSpeed).max()), (np.abs(v) + soundSpeed).max()]
```

Listing B.10: Time Integration

```
class TimeIntegration:
    @staticmethod
    def updateCellAverage(cells, rkStep, dt):
        if(Config.NUM_RK_STEPS == 1):
            for i in range(Config.NUM_GHOST_CELLS,
                Config.NUM_X_CELLS+Config.NUM_GHOST_CELLS,1):
                cells[i].u[rkStep+1] = cells[i].u[rkStep] + dt /
                    cells[i].dx * cells[i].totalFlux
        elif(Config.NUM_RK_STEPS == 2):
            for i in range(Config.NUM_GHOST_CELLS,
                Config.NUM_X_CELLS+Config.NUM_GHOST_CELLS,1):
                if(rkStep == 0):
                    cells[i].u[rkStep+1] = cells[i].u[rkStep] + dt /
                        cells[i].dx * cells[i].totalFlux
                elif(rkStep == 1):
                    cells[i].u[rkStep+1] = 0.5 *
                        {cells[i].u[rkStep-1] + cells[i].u[rkStep] +
                        dt/cells[i].dx * cells[i].totalFlux}
        elif(Config.NUM_RK_STEPS == 3):
            for i in range(Config.NUM_GHOST_CELLS,
                Config.NUM_X_CELLS+Config.NUM_GHOST_CELLS,1):
                if(rkStep == 0):
                    cells[i].u[rkStep+1] = cells[i].u[rkStep] + dt /
                        cells[i].dx * cells[i].totalFlux
                elif(rkStep == 1):
                    cells[i].u[rkStep+1] = 0.75 *
                    cells[i].u[rkStep-1] + 0.25 *
                    cells[i].u[rkStep] + 0.25 * dt/cells[i].dx *
                        cells[i].totalFlux
```

```
            elif(rkStep == 2):
                cells[i].u[rkStep+1] = 0.33 *
                cells[i].u[rkStep-2] + 0.67 *
                cells[i].u[rkStep] + 0.67 * dt/cells[i].dx *
                    cells[i].totalFlux
```

Listing B.11: Energy Calculator

```
def calculateEnergy(cells, index, rkStep):
    m1 = cells[index].u[rkStep][0]
    m2 = cells[index].u[rkStep][1]
    m3 = cells[index].u[rkStep][2]
    energy = 0
    if(not(math.isnan(m2*m2/m1))):
        energy += m2*m2/m1
    if(not(math.isnan(np.exp(m3/m1))) and
        not(math.isnan(m1**Config.gamma-1)) and
        not(math.isnan(np.exp(m3/m1) * (m1**Config.gamma-1)))):
        energy += math.isnan(np.exp(m3/m1) * (m1**Config.gamma-1))
    return energy
```

Listing B.12: Flux Calculator

```
class Flux:
    @staticmethod
    def calculateFlux(cells, rkStep):
        for i in range(len(cells)):
            cells[i].totalFlux = [0.0, 0.0, 0.0]
        for interfaceIndex in range(Config.NUM_GHOST_CELLS,
            Config.NUM_GHOST_CELLS+Config.NUM_X_CELLS+1, 1):
            # Left Values
            left_m1 = cells[interfaceIndex-1].uEast[0]
```

```
left_m2 = cells[interfaceIndex-1].uEast[1]
left_m3 = cells[interfaceIndex-1].uEast[2]
left_c1, left_c2, left_c3 = left_m2, 0.0, 0.0
left_v, left_p = 0.0,
calculatePressure(cells[interfaceIndex-1])
if(not(math.isnan(left_m2/left_m1))):
    left_v = left_m2/left_m1
left_e1 = 0.5 * left_m1 * (left_v ** 2)
# Right Values
right_m1 = cells[interfaceIndex].uWest[0]
right_m2 = cells[interfaceIndex].uWest[1]
right_m3 = cells[interfaceIndex].uWest[2]
right_c1, right_c2, right_c3 = right_m2, 0.0, 0.0
right_v, right_p = 0.0,
calculatePressure(cells[interfaceIndex])
if(not(math.isnan(right_m2/right_m1))):
    right_v = right_m2/right_m1
right_e1 = 0.5 * right_m1 * (right_v ** 2)
# Updating the left values
left_c2 += left_p
if(not(math.isnan((left_m2 ** 2) / left_m1))):
    left_c2 += ((left_m2 ** 2) / left_m1)
if(not(math.isnan(left_m2 * left_m3 / left_m1))):
    left_c3 += (left_m2 * left_m3 / left_m1)
leftValue = np.array([left_c1, left_c2, left_c3])
# Updating the right values
right_c2 += right_p
if(not(math.isnan((right_m2 ** 2) / right_m1))):
    right_c2 += ((right_m2 ** 2) / right_m1)
if(not(math.isnan(right_m2 * right_m3 / right_m1))):
    right_c3 += (right_m2 * right_m3 / right_m1)
```

```
rightValue = np.array([right_c1, right_c2, right_c3])
# Now, update the flux by taking the average of
# the left and the right values.
# Schu-Osher Flux Updation
flux = 0.5 * (leftValue + rightValue)
# ALPHA: The Correction Term
alpha = 0.0
# p-vector -> Left Side
left_mul_term, left_p1, left_p2, left_p3 =
0.0, 0.0, 0.0, 0.0
if(not(math.isnan(np.exp(left_m3/left_m1))) and
    not(math.isnan(left_m1 ** (Config.gamma-1))) and
    not(math.isnan(np.exp(left_m3/left_m1) * (left_m1
    ** (Config.gamma-1))))):
    left_mul_term += np.exp(left_m3/left_m1) * (left_m1
        ** (Config.gamma-1))
    left_p3 += left_mul_term
if(left_mul_term != 0):
    left_p1 += Config.gamma * left_mul_term
    if(not(math.isnan(left_m3/left_m1))):
        left_p1 -= left_mul_term * left_m3/left_m1
if(not(math.isnan((left_m2**2)/left_m1))):
    left_p1 -= 0.5 * (left_m2**2)/left_m1
left_vec_p = np.array([left_p1, left_p2, left_p3])
# p-vector -> Left Side
right_mul_term, right_p1, right_p2, right_p3 =
0.0, 0.0, 0.0, 0.0
if(not(math.isnan(np.exp(right_m3/right_m1)))
and not(math.isnan(right_m1 ** (Config.gamma-1)))
and not(math.isnan(np.exp(right_m3/right_m1) *
(right_m1 ** (Config.gamma-1))))):
```

```
    right_mul_term += np.exp(right_m3/right_m1) *
    (right_m1 ** (Config.gamma-1))
    right_p3 += right_mul_term
if(right_mul_term != 0):
    right_p1 += Config.gamma * right_mul_term
    if(not(math.isnan(right_m3/right_m1))):
        right_p1 -= right_mul_term * right_m3/right_m1
if(not(math.isnan((right_m2**2)/right_m1))):
    right_p1 -= 0.5 * (right_m2**2)/right_m1
right_vec_p = np.array([right_p1, right_p2, right_p3])
# Difference vector of
# the right and the left p-vector values
diff_vec = right_vec_p - left_vec_p
if(np.dot(diff_vec, diff_vec) > 1e-3):
    left_F12 = left_v * (left_e1 +
    (2 * Config.gamma - 1) /
    (Config.gamma - 1) * left_p)
    right_F12 = right_v *
    (right_e1 + (2 * Config.gamma - 1) /
    (Config.gamma - 1) * right_p)
    alpha = right_F12 - left_F12 +
    np.dot(flux, diff_vec)-
    (np.dot(right_vec_p, rightValue) -
    np.dot(left_vec_p, leftValue))
    alpha = alpha / np.dot(diff_vec, diff_vec)
    flux = flux - alpha * diff_vec
# Epsilon: The Dissipation Term
# E values for previous to previous to
# Left, Left, Right, next to Right
left_prev_E = calculateEnergy(cells,
interfaceIndex-2, rkStep)
```

```
left_E = calculateEnergy(cells,
interfaceIndex-1, rkStep)
right_E = calculateEnergy(cells,
interfaceIndex, rkStep)
right_next_E = calculateEnergy(cells,
interfaceIndex+1, rkStep)
# Left and Right h values according to
# the SLIC scheme of TORO
left_h, right_h, left_phi, right_phi=
0.0, 0.0, 0.0, 0.0
if(abs(right_E - left_p) > 1e-3):
    left_h = (left_E - left_prev_E) /
    (right_E - left_p)
    right_h = (right_next_E - right_E) /
    (right_E - left_p)
    left_phi = max(left_phi, min(1, left_h))
    right_phi = max(right_phi, min(1, right_h))
phi = min(left_phi, right_phi)
arr_t = TimeStepCalculator.getTimeStep(cells)
s_max = arr_t[1]
epsilon = 0.5 * (1 - phi) * s_max
# Difference q-vector of
# the right and the left values.
diff_vec_q = np.array([
    right_m1 - left_m1,
    right_m2 - left_m2,
    right_m3 - left_m3
])
# Updation of flux after inclusion of
# the dissipation terms
# flux -= epsilon * diff_vec_q
```

```
            cells[interfaceIndex-1].totalFlux -= flux
            cells[interfaceIndex].totalFlux += flux
```

Listing B.13: Solution Initializer

```
import math
class SolutionInitializer:
    @staticmethod
    def initializeSolution(cells):
        rkStep = 0
        for cell in cells:
            rho = 2 + np.sin(2 * np.pi * cell.cx)
            cell.u[rkStep][0] = rho
            cell.u[rkStep][1] = rho
            cell.u[rkStep][2] = -rho * Config.gamma *
            np.log(rho) / (Config.gamma-1)
```

Listing B.14: CFD SOlver in 1D

```
time_dt=[]
class CFDSolverStructured:
    @staticmethod
    def main():
        cells = [Cell() for _ in range(Config.NUM_X_CELLS +
        2 * Config.NUM_GHOST_CELLS)]
        dx = (Config.MAX_X - Config.MIN_X) / Config.NUM_X_CELLS
        for i in range(len(cells)):
            cells[i] = Cell()
            cells[i].dx = dx
            cells[i].cx = Config.MIN_X + dx * (i + 0.5 -
            Config.NUM_GHOST_CELLS)
        SolutionInitializer.initializeSolution(cells)
```

```
data = [cell.u[0] for cell in cells]
time = 0.0
lastTimeStep = False
for timeiter in range(Config.MAX_TIME_ITER):
    arr = TimeStepCalculator.getTimeStep(cells)
    dt = arr[0]
    if(time + dt > Config.STOPPING_TIME):
        dt = Config.STOPPING_TIME - time
        lastTimeStep = True
    time_dt.append(dt)
    for rkStep in range(Config.NUM_RK_STEPS):
        GhostCellUpdater.updateGhostCells(cells, rkStep)
        VariableReconstructor.reconstructVariables(cells,
            rkStep)
        Flux.calculateFlux(cells, rkStep)
        TimeIntegration.updateCellAverage(cells, rkStep, dt)
    VariableCopier.CopyToZerothRKStep(cells)
    time += dt
    if(lastTimeStep):
        break
data = [cell.u[0][0] for cell in cells]
x = [cell.cx for cell in cells]
plt.plot(x, data, color = 'magenta')
plt.scatter(x, data , marker = 's', s = 30, c = 'r')
data = np.array(data)
y = 2 + np.sin(2 * np.pi * (x - sum(time_dt)))
print(np.linalg.norm(data-y))
plt.plot(x, y)
y = 2 + np.sin(2 * np.pi * (np.array(x)))
plt.legend(['Estimate', 'Exact'])
plt.show()
```

```
cf = CFDSolverStructured.main()
```

Bibliography

[1] *Introduction to Computational Astrophysical Hydrodynamics*. Texts in Applied Mathematics Ser. Micheal Zingale, 2017.

[2] Dumbser Abgrall, Busto. A simple and general framework for the construction of thermodynamically compatible schemes for computational fluid and solid mechanics. *Applied Mathematics and Computation, 440, 127629*, 2023.

[3] Saray Busto and Michael Dumbser. A new thermodynamically compatible finite volume scheme for mangnetohydronmaics. *Journal of Scientific Computing*, 2021.

[4] Saray Busto, Michael Dumbser, Sergey Gavrilyuk, and Kseniya Ivanova. On thermodynamically compatible finite volume methods and path-conservative ader discontinuous galerkin schemes for turbulent shallow water flows. *Journal of Scientific Computing*, 88(1), Jun 2021.

[5] Peric Ferziger. Computational methods for fluid dynamics (3rd ed.). *Springer, Manchester, UK*, 2002.

[6] K. O. Friedrichs and P. D. Lax. Systems of conservation equations with a convex extension. *Proceedings of the National Academy of Sciences*, 68(8):1686–1688, Aug 1971.

[7] S. K. Godunov. An interesting class of quasilinear systems. *Sov. Math., Dokl.*, 2:947–949, 1961.

[8] Jan S. Hesthaven and Tim Warburton. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*, volume Vol. 54 of *Texts in Applied Mathematics Ser.* Springer, Dec 2007.

[9] E. Romenski I. Peshkov. A hyperbolic model for viscous newtonian flows. *Cont. Mech. Thermodyn.*, 31:1512–1541, 2019.

[10] Rony Keppens. Numerical magnetohydrodynamics. *Guest lectures at Utrecht University, May-June 2009*, 2009.

[11] Kevin L Kreider. Numerical quadrature. *University of Akron: Advanced Numerical PDEs*, 2017.

[12] J.A. Gonzalez Lora-Clavijo, Cruz-Perez. Exact solution of the 1d riemann problem in newtonian and relativistic hydrodynamics. *Revista Mexicana de Fisica*, February 2013.

[13] F. Moukalled, L. Mangani, and M. Darwish. *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM® and Matlab*, volume 113 of *Fluid Mechanics and Its Applications Ser.* Springer, Aug 2015.

[14] S.V. Patankar. Numerical heat transfer and fluid flow. 1980.

[15] E. Romenski. Hyperbolic systems of thermodynamically compatible conservation laws in continuum mechanics. *J. Sci. Compit.*, 28:115–130, 1998.

[16] Sateesh Sivakoti. As5460 project-1. *Department of Aerospace Engineering, IIT Madras*, 2018.

[17] Eleuterio F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction.* Apr 2009.

[18] Bram van Leer, James Thomas, Philip Roe, and Richard Newsome. A comparison of numerical flux formulas for the euler and navier-stokes equations. 06 1987.

[19] Jian Yu and Jan Hesthaven. A data-driven shock capturing approach for discontinuous galekin methods. *Computers & Fluids*, 245:105592, 07 2022.

[20] Michael Zeltkevic. Forward and backward euler methods. *MIT Web Course: Differential Equations Notes*, 1998.