

Fall 2011

Development of a Higher-Order Navier-Stokes Solver for Transient Compressible Flows

Arjun Vijayanarayanan
Embry-Riddle Aeronautical University

Follow this and additional works at: <https://commons.erau.edu/edt>



Part of the [Aerodynamics and Fluid Mechanics Commons](#)

Scholarly Commons Citation

Vijayanarayanan, Arjun, "Development of a Higher-Order Navier-Stokes Solver for Transient Compressible Flows" (2011). *Doctoral Dissertations and Master's Theses*. 256.
<https://commons.erau.edu/edt/256>

This Thesis - Open Access is brought to you for free and open access by Scholarly Commons. It has been accepted for inclusion in Doctoral Dissertations and Master's Theses by an authorized administrator of Scholarly Commons. For more information, please contact commons@erau.edu.

Development of a Higher-Order Navier-Stokes Solver for Transient Compressible Flows

By

Arjun Vijayanarayanan

A Graduate Thesis Submitted to the

Department of Aerospace Engineering

In Partial Fulfillment of the Requirement for the Degree of

Master of Science in Aerospace Engineering

Embry-Riddle Aeronautical University

Daytona Beach, Florida

Fall 2011

Development of a Higher-Order Navier Stokes Solver for Transient Compressible Flows

By

Arjun Vijayanarayanan

This thesis was prepared under the direction of the candidate's thesis committee chairman, Dr. William A. Engblom, Professor of Mechanical Engineering, and has been approved by the members of his thesis committee. It was submitted to the Aerospace engineering department and was accepted in partial fulfillment of the requirements for the degree of Master of Science in Aerospace Engineering.

THESIS COMMITTEE



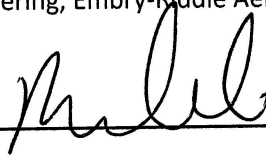
Dr. William Engblom, Chairman

Professor of Mechanical Engineering, Embry-Riddle Aeronautical University



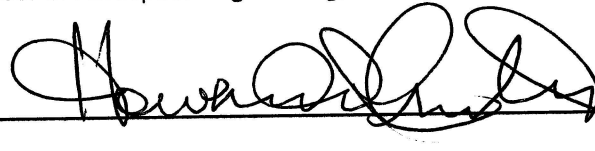
Dr. Eric Perrell, Co-Chairman

Professor of Aerospace Engineering, Embry-Riddle Aeronautical University



Dr. Reda Mankbadi, Member

Professor of Aerospace Engineering, Embry-Riddle Aeronautical University



Dr. Howard Curtis, Department Chair

Aerospace Engineering, Embry-Riddle Aeronautical University

 12-1-2011

Dr. Robert Oxley

Associate Vice President for Academics, Embry-Riddle Aeronautical University

Acknowledgements

I would like to thank my family and friends who have extended their support and love for me to stay strong and focused in a land far away from home. I thank my advisor, Dr. William Engblom, for the confidence, support, and guidance he has always extended and his lessons on CFD which he has taught me over these years. I also like to thank Dr. Mankbadi and Dr. Perrell for being on my thesis committee. Finally I thank Gaurav Bellamkonda, Lap Nguyen, Jacob Brodnick, and Sathish Xavier who have always helped and guided me in my work.

Table of Contents

Introduction	1
Governing Equations.....	1
Euler Equations	2
Grid Generation	4
Discretization: Finite Volume Method.....	5
Reynolds Transport Theorem	5
Code Development	9
First order Euler Code	9
Riemann Invariants	12
Pressure Far-Field Boundary Conditions.....	19
Wall Boundary Conditions	23
Second-order Upwind Scheme	28
MUSCL Approach	28
The Development of Limiters.....	31
The Entropy Condition	32
Monotonicity.....	34
Total Variation Diminishing (TVD) Schemes	35
Limiters	38
Comparison between the limiters	42
Influence of Viscosity	43
Code Verification.....	47
The 2-D Bump	47
Results and Discussion	48
2-D Bump Results.....	48
First-order Euler Code:.....	48
Second-order Euler Code:	52
The Shock Tube Problem	56
Shock Tube Results	59
First-order Euler Code:.....	59
Second-order Euler code without limiters:.....	60
Second-order Euler code using Van Leer limiter:	61

Second-order Euler code using Van Albada limiter:	62
Second-order Euler code using Min-mod limiter:.....	63
First-order Navier-Stokes code using the Van Leer limiter	65
Second-order Navier-Stokes code using the Van Leer limiter:	66
Suggestions for Future Work	68
References	69
Appendix	70

LIST OF FIGURES

FIGURE 1. AN EXAMPLE OF A CONTROL VOLUME.....	5
FIGURE 2. THE STRUCTURED GRID DEVELOPED FOR THE STUDY	9
FIGURE 3. FLUX SPLITTING	24
FIGURE 4. PIECEWISE LINEAR REPRESENTATION WITHIN CELLS.	29
FIGURE 5. MONOTONE AND NON-MONOTONE BEHAVIOR	34
FIGURE 6. MINMOD LIMITER	40
FIGURE 7. VAN LEER LIMITER.....	41
FIGURE 8. THE SHOCK TUBE.....	56
FIGURE 9. INITIAL PRESSURE DISTRIBUTION IN THE SHOCK TUBE	57
FIGURE 10. SHOCK TUBE ANALYTICAL RESULTS AT T=6.1 MS.....	58
FIGURE 11. RESULTS OBTAINED USING THE FIRST-ORDER EULER CODE	59
FIGURE 12. RESULTS OBTAINED FROM THE SECOND-ORDER EULER CODE WITHOUT LIMITERS.....	60
FIGURE 13. RESULTS OBTAINED USING THE SECOND-ORDER EULER CODE USING VAN LEER LIMITER	61
FIGURE 14. RESULTS OBTAINED FROM THE SECOND-ORDER EULER CODE USING VAN ALBADA LIMITER	62
FIGURE 15. RESULTS OBTAINED FROM THE SECOND-ORDER EULER CODE USING MIN-MOD LIMITER.....	63
FIGURE 16. COMPARISON BETWEEN THE FIRST-ORDER VISCOUS AND INVISCID SOLUTIONS USING VAN LEER LIMITER	65
FIGURE 17. COMPARISON BETWEEN THE SECOND-ORDER VISCOUS AND INVISCID CODES USING THE VAN LEER LIMITER	66
FIGURE 18. A SECOND ORDER VISCOUS VS INVISCID COMPARISON.....	67

List of Tables

TABLE 1. A COMPARISON BETWEEN THE LIMITERS USED IN THE RESEARCH.....	42
--	----

Abstract

Author: Arjun Vijayanarayanan

Title: Development of a Higher-Order Navier-Stokes Solver for Transient Compressible Flows

Institution: Embry-Riddle Aeronautical University

Year: 2011

A higher-order density based navier-stokes solver was developed for 2-Dimensional flows using the finite volume approach. The Van leer flux-splitting technique was used to calculate the fluxes. The second-order spatial accuracy was achieved using the variable extrapolation method developed by Van leer called the Montone Upstream Centered Scheme for Conservation Laws (MUSCL) approach. The code development was done using Matlab. The code was verified using two validation cases. Firstly, subsonic, transonic, and supersonic flows over the 2-D bump were simulated, and the results of the code were compared to the results from Fluent. Secondly, the shock-tube problem was chosen, and the results of the code were compared to the analytical results obtained from Sod's shocktube experiment.

Introduction

The 18th and 19th centuries saw the development of theoretical fluid dynamics. Fluid dynamics operated using a “two-approach world” of theory and experiment. But with the advent of high speed digital computers and development of numerical algorithms, a revolution was seen in the practice of fluid dynamics and Computational Fluid Dynamics (CFD) was born (Anderson, 1995).

CFD is a branch of fluid mechanics which uses numerical methods to simulate fluid flows. CFD is the art and science of replacing partial differential equations (PDE's), which govern fluid flows, with a set of algebraic equations which can be solved using digital computers. These PDE's are called *governing equations*, and will be discussed in the next section.

Governing Equations

The governing equations of fluid flow are a coupled system of non-linear PDE's called the *Navier-Stokes equations*, used for viscous flows. An inviscid flow is governed by a special case of the Navier-Stokes equation, called the *Euler equations*. These equations are derived after removing the viscous terms from the Navier-Stokes equations. The governing equations can be obtained in two forms, namely *conservative* and *non-conservative* forms.

Euler Equations

The Euler equations are a system of equations consisting of the continuity, momentum, and energy equations, as seen below.

Continuity equation

Nonconservative form

$$\frac{D\rho}{Dt} + \rho \nabla \cdot V = 0$$

Conservative form

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho V) = 0$$

Momentum Equation

Nonconservative form

X-Momentum:

$$\rho \frac{Du}{Dt} = -\frac{\partial p}{\partial x} + \rho f_x$$

Y-Momentum:

$$\rho \frac{Dv}{Dt} = -\frac{\partial p}{\partial y} + \rho f_y$$

Conservative form

X-Momentum:

$$\frac{\partial(\rho u)}{\partial t} + \nabla \cdot (\rho u V) = -\frac{\partial p}{\partial x} + \rho f_x$$

Y-Momentum:

$$\frac{\partial(\rho v)}{\partial t} + \nabla \cdot (\rho v V) = -\frac{\partial p}{\partial y} + \rho f_y$$

Energy Equation

Non-Conservative form:

$$\rho \frac{D(e + \frac{V^2}{2})}{Dt} = \rho \dot{q} - \frac{\partial(up)}{\partial x} - \frac{\partial(vp)}{\partial y} + \rho f \cdot V$$

Conservative form:

$$\frac{\partial}{\partial t} \left(\rho \left(e + \frac{V^2}{2} \right) \right) + \nabla \cdot \left(\rho \left(e + \frac{V^2}{2} \right) V \right) = \rho \dot{q} - \frac{\partial(up)}{\partial x} - \frac{\partial(vp)}{\partial y} + \rho f \cdot V$$

To solve these governing equations computationally, a basic approach is followed:

Pre-Processing: In this phase, the boundary of the problem is defined. Then the volume occupied by the fluid is divided into discrete cells, called Grid. The equations of motion, enthalpy, and boundaries conditions are defined now.

Analysis: This phase involves solving the equations iteratively based on the inflow and boundary conditions.

Post-Processing: Here the visualization and analysis of the solutions are performed.

Grid Generation

Grid Generation defines the computational domain on which the flow variables such as pressure and velocity are calculated. Grids can be structured and unstructured grids. For this research, a structured grid type was chosen and developed in Gridgen. An example of a structured grid is shown in Figure 1. The structured grid possesses a geometric regularity, the sense the $(i+1)^{\text{th}}$ cell is immediately to the right of the i^{th} cell, unlike the unstructured grids where cells are placed in the grid in an irregular manner.

Unstructured grids are advantageous because they are flexible and can be used around complex contours matching the mesh well with the boundary surface. Also due to their triangular and quadrilateral shapes they provide less skewing. But unstructured grids do not work well where shocks need to be captured and where boundary layer has to be analyzed in detail. Since our problem involves both shocks and boundary layer effects, a structured grid, which is simple and reduces computational time, was chosen.

Discretization: Finite Volume Method

A *Finite Volume Method*, where the governing equations are solved over discrete control volumes, was chosen for discretization. A finite volume refers to a small volume surrounding each grid point on a mesh. Here the Euler equations are discretized in its integral conservative form and conservation of the fluxes through a particular control volume is guaranteed. A control volume is shown as an example in Figure 2.

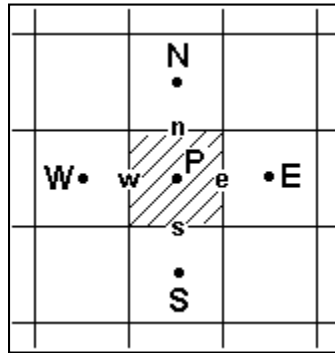


Figure 1. An example of a control volume

Reynolds Transport Theorem

The integral conservative form of the Euler equations is derived using the Reynolds Transport Theorem. According to the theorem, *“The total rate of change of an arbitrary extensive property B of the system is equal to the time rate of change of B within the control volume plus the net flux of B through the control surface.”*

Consider a differential area dS and the outward unit normal, \hat{n} . The flow of a property is thus given by $\rho b \vec{V} \cdot \hat{n} dS$. Here $\vec{V} \cdot \hat{n}$ denotes the normal component of velocity and $b = B/m$.

Now, the net rate of B coming out of the entire control volume is given by

$$\dot{B}_{net} = \iint \rho b \vec{V} \cdot \hat{n} dS$$

Again the amount of B within the control volume is given by

$$\dot{B}_{cv} = \iiint \rho b \vec{V} \cdot \hat{n} dS$$

Thus Reynolds Transport Theorem can be written as:

$$\frac{d\dot{B}_{sys}}{dt} = \frac{d}{dt} \iiint \rho b dV + \iint \rho b \vec{V} \cdot \hat{n} dS$$

Now considering a 2-D domain of surface area 'dS', and letting \vec{V}_r be the velocity of the fluid relative to the boundary, we write the Euler equations in its conservative form.

Conservation of Mass:

$$\frac{d}{dt} \iiint \rho dV + \iint \rho (\vec{V}_r \cdot \hat{n}) dS = 0$$

Conservation of Momentum:

According to the conservation of momentum, the time rate of change of linear momentum within a control volume added together with the convective flux of linear momentum through surfaces should equal the sum of the external forces.

$$\frac{d}{dt} \iiint \rho \vec{V} dV + \iint \rho \vec{V} (\vec{V}_r \cdot \hat{n}) dS = \sum \vec{F}_{ext}$$

For an inviscid flow,

$$\sum \vec{F}_{ext} = \iint -(P \cdot \hat{n}) dS$$

Thus splitting the above equations into x and y components, we get

X-Momentum:

$$\frac{d}{dt} \iiint \rho u dV + \iint (\rho u (\vec{V}_r \cdot \hat{n}) + P \cdot \hat{n}_x) dS = 0$$

Y-Momentum:

$$\frac{d}{dt} \iiint \rho v dV + \iint (\rho v (\vec{V}_r \cdot \hat{n}) + P \cdot \hat{n}_y) dS = 0$$

Energy Equation:

Based on the first law of thermodynamics, the time rate of change of energy within a control volume added with the convective flux of energy through the surface should equal the externally applied rates of heat and work done.

$$\frac{d}{dt} \iiint \rho e_o dV + \iint \rho h_o (\vec{V}_r \cdot \hat{n}) dS = 0$$

All the above equations can be written in a generic form, as stated below:

$$\frac{d}{dt} \iiint \bar{U} d\mathbf{V} + \iint \bar{F}_{inv} \cdot \hat{n} dS = 0$$

Where,

$$\bar{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_o \end{bmatrix} \text{ and } \bar{F}_{inv} \cdot \hat{n} = \begin{bmatrix} \rho(\bar{V}_r \cdot \bar{n}) \\ \rho u(\bar{V}_r \cdot \bar{n}) + \hat{n}_x P \\ \rho v(\bar{V}_r \cdot \bar{n}) + \hat{n}_y P \\ \rho h_o(\bar{V}_r \cdot \bar{n}) \end{bmatrix}$$

Rewriting into x and y components,

$$\frac{d}{dt} \iiint \bar{U} d\mathbf{V} + \iint \bar{F}_x \cdot \hat{n}_x dS + \iint \bar{F}_y \cdot \hat{n}_y dS = 0$$

$$\bar{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_o \end{bmatrix} \quad \bar{F}_x = \begin{bmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ \rho h_o u \end{bmatrix} \quad \bar{F}_y = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + P \\ \rho h_o v \end{bmatrix}$$

\bar{U} is the conservative state vector or the solution vector and \bar{F}_x, \bar{F}_y are the flux vectors in the x and y directions.

Code Development

A code was developed using Matlab, and code development was done in three stages, as discussed below.

1. Firstly, a first order Euler code was developed using the Van Leer flux-splitting scheme (the details of the flux-splitting scheme will be discussed later.)
2. Secondly, a higher order, second order in our case, Euler code was developed using the Monotone Upstream Centered Scheme for Conservation Laws (MUSCL) approach. Three limiters; Van leer limiter, Van Albada limiter, and the Min-Mod limiter were used and compared against each other.
3. Thirdly, viscous terms were introduced into the code, making the code a Navier-Stokes code.

These stages are discussed in detail in the sections below.

First order Euler Code

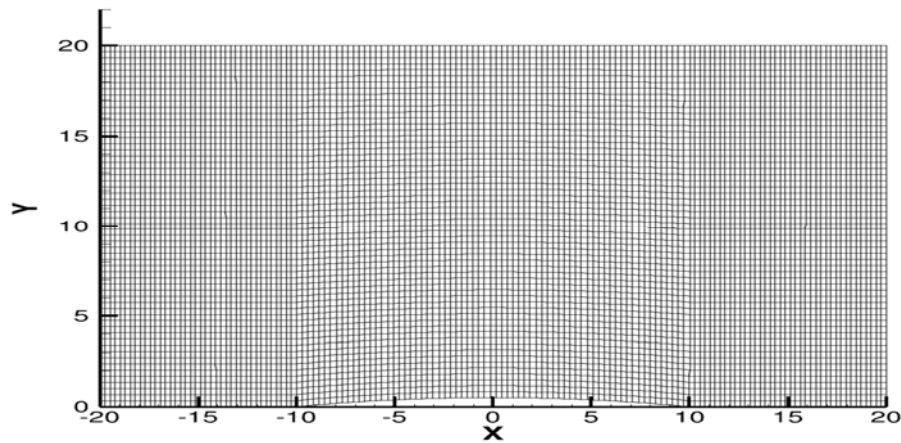


Figure 2. The structured grid developed for the study

To read the above grid, a *grid loader* was developed in Matlab and used. The area of each face, the respective outward normal was calculated. The volume of each cell was calculated too.

Initialize Free Stream Conditions

The free stream conditions like density (ρ), velocity in the x-direction (u), velocity in the y-direction (v), pressure (P), and temperature (T), as seen in Table 1, are considered as the inflow conditions. Other values specified include; universal gas constant, R ; specific heat ratio, γ ; specific heat at constant volume. Every cell in the grid was initialized with these values, before the time loop begins.

Primitive and Conservative Vectors

The primitive state vector (V) is an array containing ρ , u , v , P of every cell.

$$V = [\rho \quad u \quad v \quad P]^T$$

The conservative state vector (U) is an array which includes the flux variables.

$$U = \left[\rho \quad \rho u \quad \rho v \quad \rho \left(\frac{C_v P}{\rho R} + \frac{1}{2} (u^2 + v^2) \right) \right]^T$$

The conservative state vector and the primitive state vectors have to be defined for every cell.

Time Loop

The time loop is started to run the calculation for a particular number of iterations. At the end of each time loop, the solution vector is updated using the Euler step. After the calculation of the updated flux variables from the solution vector, the primitive state variables are updated and used for the next iteration.

Calculating Contravariant Velocities and Mach Numbers

A contravariant velocity (\hat{u}), is a velocity projected in the direction normal to the face. This velocity is calculated by the dot product of the velocity and the outward normal of the face. When these contravariant velocities are divided by the speed of sound (a) of the particular cell, the contravariant mach numbers (\hat{M}) are obtained.

$$\hat{u} = u * \hat{n}_x + v * \hat{n}_y$$
$$\hat{M} = \frac{\hat{u}}{a}$$

Time step calculation

The time step is calculated by dividing the face length by the wave speed corresponding to that cell. Since there are four faces in a cell, the minimum of the above calculated value is considered and multiplied with the *Courant-Friedrichs-Lewy* number (CFL number). A CFL number of 0.1 was used in this research.

$$\Delta t = \left(\frac{FaceLength}{|\hat{u}| + c} \right) * CFL$$

Boundary Conditions

Developing appropriate boundary conditions is very important for flow analysis. These conditions determine the flow behavior at the boundary of the domain, and vital for the solving the governing equations.

In this research we used the *pressure far field* boundary condition, which is used to model the free-stream conditions at infinity, with the free stream and static conditions being specified. This boundary condition is also referred to as the characteristic boundary condition, since it uses the characteristic information (Riemann Invariants) to determine the flow variables at the boundary surface.

Riemann Invariants

The Euler equations, in its non-conservative form can be written as:

The Continuity Equation:

$$\frac{d\rho}{dt} + u \frac{d\rho}{dx} + \rho \frac{du}{dx} = 0$$

The X-Momentum Equation:

$$\frac{du}{dt} + u \frac{du}{dx} + \frac{1}{\rho} \frac{\partial P}{\partial x} = 0$$

The Energy Equation:

$$\frac{\partial P}{\partial t} + u \frac{dP}{dx} + \rho c^2 \frac{\partial u}{\partial x} = 0$$

For the primitive variable vector, \vec{V} , these equations can be written as;

$$\frac{\partial \vec{V}}{\partial t} + \tilde{A} \frac{\partial \vec{V}}{\partial x} = 0$$

Where, \tilde{A} is the jacobian matrix given by

$$\begin{bmatrix} u & \rho & 0 \\ 0 & u & \frac{1}{\rho} \\ 0 & \rho c^2 & u \end{bmatrix}$$

Now we need to find the Eigen values and Eigen vectors for the above jacobian matrix.

$$\det|\tilde{A} - I\lambda| = 0$$

$$\begin{vmatrix} u - \lambda & \rho & 0 \\ 0 & u - \lambda & \frac{1}{\rho} \\ 0 & \rho c^2 & u - \lambda \end{vmatrix} = 0$$

The eigen values were calculated as u , $u + c$, and $u - c$ and the eigen vectors defined up to an arbitrary normalization are;

$$\tilde{l}^{(1)} = \begin{pmatrix} \alpha & 0 & -\frac{\alpha}{c^2} \end{pmatrix}$$

$$\tilde{l}^{(2)} = \begin{pmatrix} 0 & \beta & \frac{\beta}{\rho c} \end{pmatrix}$$

$$\tilde{l}^{(3)} = \begin{pmatrix} 0 & \delta & -\frac{\delta}{\rho c} \end{pmatrix}$$

Where α , β , and δ are normalization coefficients. Now considering $\alpha=\beta=\delta=1$, we obtain the diagonalization matrix given by

$$L^{-1} = \begin{vmatrix} 1 & \rho & -\frac{1}{c^2} \\ 0 & 1 & \frac{1}{\rho c} \\ 0 & 1 & -\frac{1}{\rho c} \end{vmatrix}$$

Therefore,

$$L = \begin{vmatrix} 1 & \frac{\rho}{2c} & -\frac{\rho}{2c} \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{\rho c}{2} & -\frac{\rho c}{2} \end{vmatrix}$$

Note that the columns of L are the right Eigen vectors of \tilde{A}

Now substituting in the original compatibility equation for the primitive variable,

$$L^{-1} \frac{\partial \vec{V}}{\partial t} + (L^{-1} \tilde{A} L) L^{-1} \frac{\partial \vec{V}}{\partial x} = L^{-1} \tilde{Q}$$

OR

$$L^{-1} \frac{\partial \vec{V}}{\partial t} + \wedge L^{-1} \frac{\partial \vec{V}}{\partial x} = L^{-1} \tilde{Q}$$

Where \tilde{Q} is the source vector and

$$\Lambda = L^{-1} \tilde{A} L = \begin{vmatrix} u & 0 & 0 \\ 0 & u+c & 0 \\ 0 & 0 & u-c \end{vmatrix}$$

Now writing these equations explicitly, we obtain

$$\frac{\partial \rho}{\partial t} - \frac{1}{c^2} \frac{\partial \rho}{\partial t} + u \frac{\partial \rho}{\partial x} - \frac{u}{c^2} \frac{\partial \rho}{\partial x} = 0$$

$$\frac{\partial u}{\partial t} + \frac{1}{\rho c} \frac{\partial \rho}{\partial t} + (u+c) \left(\frac{\partial u}{\partial x} - \frac{1}{\rho c} \frac{\partial \rho}{\partial x} \right) = 0$$

$$\frac{\partial u}{\partial t} + \frac{1}{\rho c} \frac{\partial \rho}{\partial t} + (u-c) \left(\frac{\partial u}{\partial x} - \frac{1}{\rho c} \frac{\partial \rho}{\partial x} \right) = 0$$

Now the characteristic variables are defined as

$$\delta W = (\delta w_1 \quad \delta w_2 \quad \delta w_3)^T$$

Where δW represents an arbitrary variation, either ∂_t or ∂_x , obtained from $\delta W = L^{-1} \delta V$

$$\delta w_1 = \delta \rho - \frac{1}{c^2} \delta p$$

$$\delta w_2 = \delta u + \frac{1}{\rho c} \delta p$$

$$\delta w_3 = \delta u - \frac{1}{\rho c} \delta p$$

Now the characteristic form of the 1-D Euler equations can be written as

$$\frac{\partial W}{\partial t} + \Lambda \frac{\partial W}{\partial x} = L^{-1} \tilde{Q}$$

Now that entropy is constant at the boundary (which means $\tilde{Q}=0$) the above equation can be re-written as

$$\frac{\partial}{\partial t} \begin{vmatrix} w_1 \\ w_2 \\ w_3 \end{vmatrix} + \begin{vmatrix} u & 0 & 0 \\ 0 & u+c & 0 \\ 0 & 0 & u-c \end{vmatrix} \frac{\partial}{\partial x} \begin{vmatrix} w_1 \\ w_2 \\ w_3 \end{vmatrix} = 0$$

Riemann Variables:

The above equation shows that the quantities $w_{(j)}$ propagate along the corresponding characteristics with speeds $\lambda_{(j)}$.

$$\delta w_1 = \delta \rho - \frac{1}{c^2} \delta p, \text{ propagates with velocity } u \text{ along the characteristic } C_0 \text{ defined by } \frac{dx}{dt} = u$$

$$\delta w_2 = \delta u + \frac{1}{\rho c} \delta p = \delta u + \frac{c}{\rho} \delta p, \text{ (since } c = \sqrt{\left(\frac{\partial P}{\partial \rho}\right)_s} \text{), propagates with velocity } u+c \text{ along the}$$

characteristic C_+ defined by $\frac{dx}{dt} = u+c$

$$\delta w_3 = \delta u - \frac{1}{\rho c} \delta p = \delta u - \frac{c}{\rho} \delta p, \text{ propagates with velocity } u-c \text{ along the characteristic } C_-$$

defined by $\frac{dx}{dt} = u-c$

The C_+ and C_- characteristics are also called *Mach Lines*.

Thus the characteristic form of the Euler equations can now be written as

$$\frac{dw_{(j)}}{dt} + \lambda_{(j)} \frac{\partial w_{(j)}}{\partial x} = 0$$

Along the characteristic line $w_{(j)}$ remains a constant: $\frac{dx}{dt} = \lambda$, Thus

$$\frac{dw}{dt} + \frac{dx}{dt} \cdot \frac{\partial w}{\partial x} = 0$$

Where the variables “ w ” are called *Riemann Variables*. When they are a constant, they are referred to as *Riemann Invariants*.

Now in the system of characteristic form of the Euler equations, the first equation represents

the constant transport of entropy along the path line $\frac{dx}{dt} = u$.

We know that $ds = -\frac{\gamma C_v}{\rho} \left(d\rho - \frac{d\rho}{c^2} \right)$. Now the first equation that we are considering is

represented by the condition $d^{(0)}s = 0$ along $\frac{dx}{dt} = u$ or $\frac{ds}{dt} + u \frac{\partial s}{\partial x} = 0$. Also, as long as there is

no discontinuity the entropy propagates along the path line and is conserved along that characteristic.

Now for isentropic flows, the Riemann Variables can be integrated as follows;

For C_+ ,

$$w_2 = u + \int \frac{dp}{\rho c} = u + \int c(\rho) \frac{d\rho}{\rho}$$

We from the isentropic relations that $p = k\rho^\gamma$ and that $c^2 = k\gamma\rho^{\gamma-1}$, which gives

$$w_2 = u + \frac{2}{\gamma-1}c \text{ and } w_3 = u - \frac{2}{\gamma-1}c$$

Substituting these values back into the characteristic equations, we obtain

$$\frac{\partial s}{\partial t} + u \frac{\partial s}{\partial x} = 0$$

$$\frac{\partial}{\partial t} \left(u + \frac{2}{\gamma-1}c \right) + (u+c) \frac{\partial}{\partial x} \left(u + \frac{2}{\gamma-1}c \right) = 0$$

$$\frac{\partial}{\partial t} \left(u - \frac{2}{\gamma-1}c \right) + (u-c) \frac{\partial}{\partial x} \left(u - \frac{2}{\gamma-1}c \right) = 0$$

Now even though the flow is 1-D, y-momentum is needed as 'v' is convected,

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

Now including the y-momentum in the characteristic form we get;

$$\frac{\partial}{\partial t} \begin{bmatrix} v \\ s \\ j^+ \\ j^- \end{bmatrix} + \begin{bmatrix} u & 0 & 0 & 0 \\ 0 & u & 0 & 0 \\ 0 & 0 & u+c & 0 \\ 0 & 0 & 0 & u \end{bmatrix} \frac{\partial}{\partial x} \begin{bmatrix} v \\ s \\ j^+ \\ j^- \end{bmatrix} = 0$$

Where j^+ and j^- are the Riemann Invariants, given by,

$$j^+ = u + \frac{2}{\gamma-1}c \text{ and } j^- = u - \frac{2}{\gamma-1}c$$

Pressure Far-Field Boundary Conditions

At the boundaries, depending on whether the flow is moving upstream or downstream, information has to be specified from the exterior or extrapolated from the interior.

Supersonic Flow:

For a Supersonic flow, $u, u+c, u-c$ all are > 0 . Thus all the waves move downstream and we must specify all the four state values at the inflow and the values at the outflow are extrapolated.

Inflow:

$$\vec{V}_{b\text{infl}} = \vec{V}_{\infty}, \text{ where}$$

Where, $\vec{V}_{b\text{infl}}$ is the primitive state vector and \vec{V}_{∞} is the free stream values

$$\text{The enthalpy, } h_{b\text{infl}} = C_p \left(\frac{P_{\infty}}{\rho_{\infty} R} \right) + \frac{1}{2} (u_{\infty}^2 + v_{\infty}^2)$$

Outflow:

All the values are extrapolated from the cell just before the outflow boundary

$$\vec{V}_{b\text{ofl}} = \vec{V}_{\text{cell}(i)}$$

$$h_{bofl} = C_p \left(\frac{P_{cell(i)}}{\rho_{cell(i)} R} \right) + \frac{1}{2} (u_{cell(i)}^2 + v_{cell(i)}^2)$$

Subsonic Flow:

For a subsonic flow,

$$u > 0$$

$$u + c > 0$$

$$u - c < 0$$

Thus three waves go downstream and one wave ($u - c$) moves upstream. Therefore, for the inflow we specify three values and extrapolate one value and for the outflow we specify one value and extrapolate three values.

Inflow:

Based on Riemann Invariants, we have

$$v_{bin} = v_{\infty}$$

$$s_b = \frac{P_{\infty}}{\rho_{\infty}^{\gamma}} \geq \frac{P_{bin}}{\rho_{bin}^{\gamma}} = \frac{P_{\infty}}{\rho_{\infty}^{\gamma}}$$

$$u_{bin} + \frac{2c_{bin}}{\gamma - 1} = u_{\infty} + \frac{2c_{\infty}}{\gamma - 1}$$

$$u_{bin} - \frac{2c_{bin}}{\gamma - 1} = u_{cell(i)} - \frac{2c_{cell(i)}}{\gamma - 1}$$

$$c_{bin} = \sqrt{\left(\frac{\gamma P_{bin}}{\rho_{bin}}\right)}$$

$$c_{\infty} = \sqrt{\left(\frac{\gamma P_{\infty}}{\rho_{\infty}}\right)}$$

Solving the five equations, we obtain:

$$v_{bin} = v_{\infty}$$

$$u_{bin} = 0.5 * \left((u_{\infty} + u_{cell(i)}) + \frac{2}{\gamma - 1} (c_{\infty} + c_{cell(i)}) \right)$$

$$c_{bin} = 0.25 * (\gamma - 1) * \left((u_{\infty} - u_{cell(i)}) + \frac{2}{\gamma - 1} (c_{\infty} + c_{cell(i)}) \right)$$

$$\rho_{bin} = \left(\frac{(c_{bin})^2 (\rho_{\infty}^{\gamma})}{\gamma * P_{\infty}} \right)^{\frac{1}{\gamma - 1}}$$

$$P_{bin} = \left(\frac{(P_{\infty}) (\rho_{bin}^{\gamma})}{\rho_{\infty}^{\gamma}} \right)$$

The enthalpy is calculated using the boundary inflow conditions.

Outflow:

The $u - c$ wave is specified and the remaining waves are extrapolated.

Again from the Riemann Invariants,

$$v_{ob} = v_{cell(i)}$$

$$s_b = \frac{P_{cell(i)}}{\rho_{cell(i)}^\gamma} \geq \frac{P_{ob}}{\rho_{ob}^\gamma} = \frac{P_{cell(i)}}{\rho_{cell(i)}^\gamma}$$

$$u_{ob} + \frac{2c_{ob}}{\gamma - 1} = u_{cell(i)} + \frac{2c_{cell(i)}}{\gamma - 1}$$

$$u_{ob} - \frac{2c_{ob}}{\gamma - 1} = u_\infty - \frac{2c_\infty}{\gamma - 1}$$

$$c_{ob} = \sqrt{\left(\frac{\gamma P_{ob}}{\rho_{ob}} \right)}$$

$$c_\infty = \sqrt{\left(\frac{\gamma P_\infty}{\rho_\infty} \right)}$$

Solving the above equations we get:

$$v_{ob} = v_{cell(i)}$$

$$u_{ob} = 0.5 * \left((u_{cell(i)} + u_{\infty}) + \frac{2}{\gamma - 1} (c_{cell(i)} - c_{\infty}) \right)$$

$$c_{ob} = 0.25 * (\gamma - 1) * \left((u_{cell(i)} - u_{\infty}) + \frac{2}{\gamma - 1} (c_{\infty} + c_{cell(i)}) \right)$$

$$\rho_{ob} = \left(\frac{(c_{ob})^2 (\rho_{cell(i)}^{\gamma})}{\gamma * P_{cell(i)}} \right)^{\frac{1}{\gamma - 1}}$$

$$P_{ob} = \left(\frac{(P_{cell(i)}) (\rho_{ob}^{\gamma})}{\rho_{cell(i)}^{\gamma}} \right)$$

Wall Boundary Conditions

In our case, we have inviscid, slip walls, present at all four boundaries. Since the walls are non-porous, the component of velocity normal to the wall is zero, and the flow is tangential to the surface.

$$u_w = u_{cell(i)} - \left((u_{cell(i)} \times \hat{n}_x) + (v_{cell(i)} \times \hat{n}_y) \right)$$

$$v_w = v_{cell(i)} - \left((u_{cell(i)} \times \hat{n}_x) + (v_{cell(i)} \times \hat{n}_y) \right)$$

Calculating Fluxes on the Boundary Faces

The fluxes, in both x and y directions, are calculated on all the boundaries.

$$\vec{F}_x = \begin{bmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ \rho h_o u \end{bmatrix} \quad \vec{F}_y = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + P \\ \rho h_o v \end{bmatrix}$$

The total flux is now calculated using,

$$\vec{F} = \vec{F}_x * \hat{n}_x + \vec{F}_y * \hat{n}_y$$

Calculating Fluxes on the Non-Boundary Faces

The normal shocks and the expansion waves created in the shock tube produces interesting behaviors in the flow. The Van Leer flux splitting scheme, as discussed below, was used for calculating the fluxes on the non-boundary faces when a sub-sonic flow existed.

Van Leer Flux-Splitting Scheme:

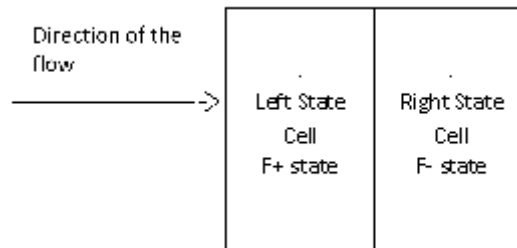


Figure 3. Flux Splitting

The Van Leer flux-splitting scheme uses the Eigen vectors of the flux vector, which are basically the wave speeds of the characteristic form of the Euler equations. Each wave speed, with a positive component (left flux, arising from positive wave speeds) and the negative component (right flux, arising from negative wave speeds) contribute to the flux. This method of flux calculations are based on the splitting of the Mach number, and offer continuity at the sonic and stagnation points.

$$F_i = F_i^+ + F_i^-$$

Where,

F_i^+ is the left state flux given by,

$$f^+ = \frac{\rho a}{4} (\hat{M} + 1)^2 \begin{bmatrix} 1 \\ u + \frac{\hat{n}_x(-\hat{u} + 2a)}{\gamma} \\ v + \frac{\hat{n}_y(-\hat{u} + 2a)}{\gamma} \\ h_o - \frac{a^2(\hat{M} - 1)^2}{\gamma + 1} \end{bmatrix}$$

F_i^- is the right state flux given by,

$$f^- = -\frac{\rho a}{4} (\hat{M} - 1)^2 \begin{bmatrix} 1 \\ u + \frac{\hat{n}_x(-\hat{u} - 2a)}{\gamma} \\ v + \frac{\hat{n}_y(-\hat{u} - 2a)}{\gamma} \\ h_o - \frac{a^2(\hat{M} + 1)^2}{\gamma + 1} \end{bmatrix}$$

For supersonic flows, since the information is passed only in one direction, the total flux can be calculated by the using the formula below.

$$F = \begin{bmatrix} \rho u \\ \rho \hat{u}u + \hat{n}_x P \\ \rho \hat{u}v + \hat{n}_y P \\ \rho \hat{u}h_o \end{bmatrix}$$

Calculating Residuals

The time rate of change of the conserved variables is defined as Residuals. It is calculated by,

$$Residual = \int_s (\vec{F}_x \hat{n}_x) ds + \int_s (\vec{F}_y \hat{n}_y) ds$$

These residuals are used to update the conservative state vector. If the residuals are equal to zero, there will be no change in the conservative state vector ($\frac{d\vec{U}}{dt} = 0$), and the solution is called as a *converged solution*.

Updating the Conservative state vector

The solution vector is updated using the *Euler Step*, which is:

$$\bar{U}_i^{n+1}(c) = \bar{U}_i^n(c) - \frac{\Delta t}{\mathbf{V}(c)} [R_i(c)]$$

Where,

$\bar{U}_i^{n+1}(c)$ is the updated solution vector for the next time step

$\bar{U}_i^n(c)$ is the solution vector of the current time step

Δt is the global time step

$R_i(c)$ is the residual of the cell

$V(c)$ is the volume of the cell

Updating the primitive state vector

Updating the primitive state vectors can be done after we get the updated conservative state vectors. The updated primitive state vectors are used for the next iteration.

$$\bar{V}(c) = \begin{bmatrix} \bar{U}_1(c) \\ \bar{U}_2(c)/\bar{U}_1(c) \\ \bar{U}_3(c)/\bar{U}_1(c) \\ \left\{ \bar{U}_4(c) - \left(0.5 * \bar{U}_1(c) * \left(\frac{\bar{U}_2(c)}{\bar{U}_1(c)} \right)^2 + \left(\frac{\bar{U}_3(c)}{\bar{U}_1(c)} \right)^2 \right) \right\} * \left(\frac{R}{C_v} \right) \end{bmatrix}$$

Calculating L2 Norms

L2 norms were calculated to check for convergence of the solution. L2 will be equal to zero for a converged solution.

$$L_2 = \sqrt{\frac{\sum_{c=1}^n (\text{Residual})^2}{n}}$$

Second-order Upwind Scheme

According to Charles Hirsch (Hirsch, 1988), *“Motivations behind upwind schemes is the hope that the introduction of physical propagation properties in the discretization will prevent the generation of oscillations in the numerical solutions. This is only partly fulfilled in the sense that for non-linear equations, such as the Euler equations, oscillation-free results can be obtained for weak stationary discontinuities. However, this is not a general property, since it can be shown theoretically that linear second-order upwind schemes always generate oscillations”* (p.493).

The generation of oscillations around discontinuities was seen with the replacement of the second-order formulae over the first-order upwind schemes. It was noted that the first order accuracy of the Godunov schemes, came from the projection stage, where the solution is projected in each cell ($i-1/2$, $i+1/2$) on piece-wise constant states (Hirsch, 1988). Based on this observation, Van leer, in order to achieve higher spatial approximations, proposed to modify the first projection stage without modifying the Riemann solver. This method of generating second-order upwind schemes, developed by Van Leer (1979) was called as the Monotone Upstream-Centered Schemes for Conservation Laws (MUSCL), and is discussed below.

MUSCL Approach

In the MUSCL approach, the discrete state variables represent the average within the cells and the piecewise linear or quadratic distributions have to average out these values (Hirsch, 1988), as seen in Figure 7.

Considering a general local representation at a specific instant, valid within a cell i ,

$$U(x) = U_i + \left(\frac{1}{\Delta x}\right)(x-x_i)\delta_i U + \left(\frac{3\kappa}{2\Delta x^2}\right)\left[(x-x_i)^2 - \frac{\Delta x^2}{12}\right]\delta_i^2 U \quad x_{i-\frac{1}{2}} < x < x_{i+\frac{1}{2}}$$

Where,

U_i is the average value, defined by

$$U_i = \left(\frac{1}{\Delta x}\right) \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} U(x) dx$$

$\delta_i U$, $\delta_i^2 U$ are the estimations of the first and second derivatives within cell i .

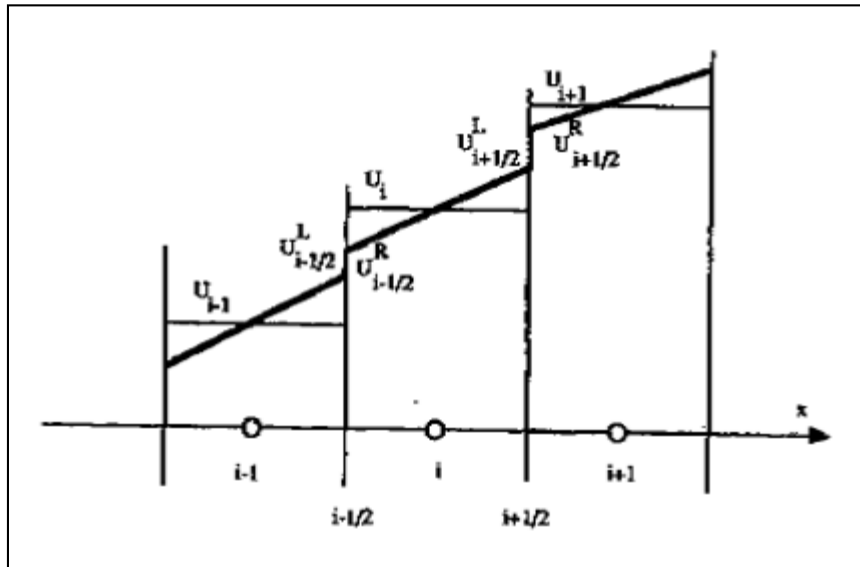


Figure 4. Piecewise linear representation within cells.

The values of $\delta_i U$, $\delta_i^2 U$ are estimated by central differencing the averaged values from the adjacent cells.

Hence,

$$\delta_i U = \frac{(U_{i+1} - U_{i-1}))}{2}$$

$$\delta_i^2 U = U_{i+1} - 2 U_i + U_{i-1}$$

Now setting $x = x_i \pm \frac{\Delta x}{2}$, we obtain:

$$U_{i+\frac{1}{2}}^L = U_i + (1/2) \delta_i U + (\kappa/4) \delta_i^2 U$$

$$= U_i + \left(\frac{1}{4}\right)(1-\kappa)(U_i - U_{i-1}) + \left(\frac{1}{4}\right)(1+\kappa)(U_{i+1} - U_i)$$

$$U_{i-\frac{1}{2}}^R = U_i - (1/2) \delta_i U + (\kappa/4) \delta_i^2 U$$

$$= U_i - \left(\frac{1}{4}\right)(1+\kappa)(U_i - U_{i-1}) - \left(\frac{1}{4}\right)(1-\kappa)(U_{i+1} - U_i)$$

In these equations, the first term on the right-hand side correspond to the first-order schemes and the higher order accuracy in space is obtained by the additional dependence on adjacent points. By introducing a parameter, ϵ , both the options are combined. When $\epsilon = 0$, the scheme is first-order and $\epsilon = 1$ for a higher order scheme.

$$U_{i+\frac{1}{2}}^L = U_i + (\varepsilon/4) [(1-\kappa) (U_i - U_{i-1}) + (1+\kappa) (U_{i+1} - U_i)]$$

$$U_{i+\frac{1}{2}}^R = U_{i+1} - (\varepsilon/4) [(1+\kappa) (U_{i+1} - U_i) + (1-\kappa) (U_{i+2} - U_{i+1})]$$

In this research, a fully upwind scheme ($\kappa = -1$) is considered. Therefore, the above equations change to:

$$U_{i+\frac{1}{2}}^L = U_i + (\varepsilon/2) (U_i - U_{i-1})$$

$$U_{i+\frac{1}{2}}^R = U_{i+1} - (\varepsilon/2) (U_{i+2} - U_{i+1})$$

The Development of Limiters

It is noted that the upwind schemes by themselves are not sufficient to avoid the appearance of oscillations around the discontinuities with the second-order schemes. The numerical generation of oscillations is due to the numerical treatment of the second-order schemes. It was noted that the development of the second-order schemes from the first-order counterpart was fully contained in the representation of the state-variables as piecewise linear within each cell, instead of piecewise constants. It was also observed that next to the cell average, the additional variable term is the slope of the linear variation (Hirsch, 1988) . Thus when the slopes become larger than the difference of the two adjacent values, undesirable oscillations appear. Hence, it was concluded that to avoid the appearance of oscillations, larger gradients need to be prevented.

In 1959 Godunov (as cited by Hirsch, 1988), showed that all monotone schemes, discussed below, can be at most of first-order accuracy. Thus, it was proposed that correction factors, called *Limiters*, should be introduced into the second-order schemes to achieve the intended goal. Thus, the role of the limiters is to force the gradients to follow a smooth trend and forcefully reduce the gradients to follow the smooth trend. The non-oscillatory properties of the Euler equations are expressed by,

- The entropy condition
- Monotonicity
- Total Variation Diminishing (TVD) schemes

These properties are discussed below.

The Entropy Condition

According to the second law of thermodynamics, in any physical adiabatic evolution, the entropy only increases during the transformation of the system. So, only compression shocks are retained and the expansion shocks, which correspond to negative entropy variation, are excluded because they cannot occur in real flows. Now due to the lack of a built-in dissipative mechanism in the inviscid flow models, such as viscous effects, there is a need of an additional condition to be added to the system of equations to select the correct shock and reject the physically impossible ones. This condition is called the entropy condition.

The condition to be satisfied by the discontinuous solutions of the hyperbolic conservation law is that the wave speed $a(u) = df/du$ is such that,

$$a_r = a(u_r) < C < a(u_L) = a_L$$

where C is the speed of propagation of the discontinuity (which satisfies the Rankine-Hugoniot relations) and u_r, u_L are values of u on the right and left sides of the discontinuity. This shows that the characteristics on either side of the discontinuity surface Σ will ultimately intersect the discontinuity.

Also, the flux function, f , should satisfy,

$$\frac{f(u) - f(u_R)}{u - u_R} \leq \frac{f(u_L) - f(u_R)}{u - u_R} \leq \frac{f(u_L) - f(u)}{u_L - u} \quad \text{for } u_R \leq u \leq u_L \quad \text{for } u_R \leq u \leq u_L$$

Every initial value problem has a generalized unique solution satisfying the entropy condition, which can be considered as the limit for vanishing coefficient v .

For example, consider the parabolic equation

$$u_t + f_x = v u_{xx}$$

The above equation is similar to a 1-D Navier-Stokes equation. Practical application of the entropy condition in numerical computation of inviscid flows can be derived from the above result. Thus if appropriate artificial viscosity is added to the discretized equations, non-physical discontinuities will never occur.

Monotonicity

A scheme is considered as monotone if it does not lead to an oscillatory behavior, as seen in Figure 8. The smooth behavior of a numerical solution can be largely attributed to monotonicity and weakly attributed to total variation (Hirsch, 1988), discussed in the next section.

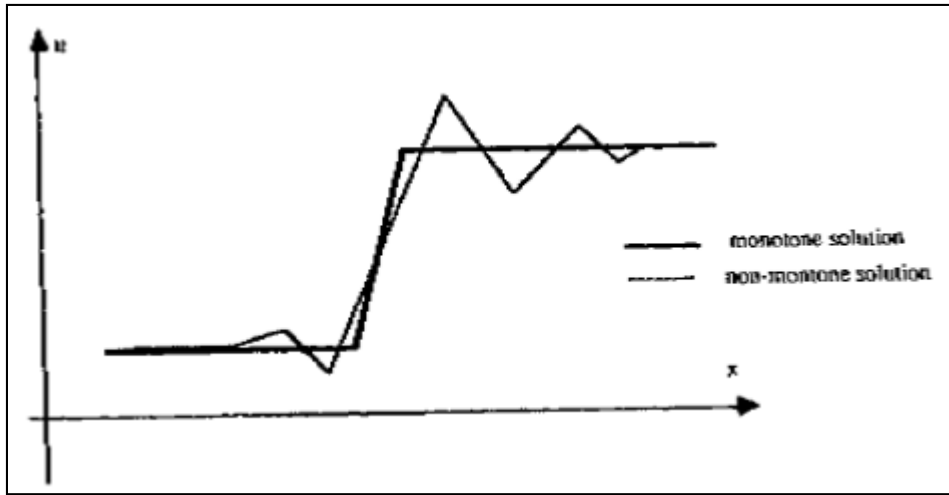


Figure 5. Monotone and Non-Monotone behavior

The monotonicity condition applied to a scalar conservation equation $u_i + f_x = 0$ can be expressed in the form:

$$u_i^{n+1} = H(u_{i-k}^n, u_{i-k+1}^n, \dots, u_{i+k}^n)$$

The scheme is said to be monotone if H is a monotone increasing function of each of its arguments,

$$\frac{\partial H}{\partial u_j} (u_{i-k}^n, u_{i-k+1}^n, \dots, u_{i+k}^n) \geq 0 \quad \text{for all } i-k \leq j \leq i+k$$

For any linear scheme in the form, $u_i^{n+1} = \sum_k b_k u_{i+k}^n$, the monotonicity condition requires that the coefficients b_k be non-negative.

Earlier we had discussed that the conservative monotone schemes for the Euler equations are only of first-order accuracy. But first-order accurate schemes are not sufficient for practical purposes. Thus, it was necessary that a condition less severe than that of monotonicity be developed allowing the higher order schemes to generate entropy condition satisfying solutions without oscillations. These schemes are referred to as *High resolution schemes*. The concept of *total variation* of a numerical solution was introduced, and will be discussed in the section below.

Total Variation Diminishing (TVD) Schemes

The condition of total variation boundedness is less severe than monotonicity, but sufficient to guarantee convergence. It is based on the principle that the total variation of any physically admissible solution, $TV = \int |\partial u / \partial x| dx$, does not increase in time (Lax, 1973 as cited by Hirsch, 1988). The total variation in the x-direction of a discrete solution to a scalar conservation law is defined by:

$$TV(u) \equiv \sum_i |u_{i+1} - u_i|$$

A numerical scheme is said to be total variation diminishing if,

$$TV(u^{n+1}) \leq TV(u^n)$$

Also, if all the following properties of monotones are preserved as a function of time,

- No new local extrema in x can be created
- The value of a local minimum is non-decreasing, the value of a local maximum is non-increasing

then the scheme is *monotonicity preserving*. It is important to note that:

- All monotone schemes are TVD
- All TVD schemes are monotonicity preserving.

Consider a linearized second-order upwind scheme, with $f^\pm = a^\pm u$, written as a multi-point scheme:

$$\frac{du_i}{dt} = -\frac{a^+}{2\Delta x} [3(u_i - u_{i-1}) - (u_{i-1} - u_{i-2})] - \frac{a^-}{2\Delta x} [3(u_{i+1} - u_i) - (u_{i+2} - u_{i+1})]$$

Thus the coefficients of expansion are

$$C_{i+\frac{1}{2}}^{-(1)} = \frac{3}{2}a^- \quad C_{i+\frac{3}{2}}^{-(2)} = \frac{-1}{2}a^-$$

$$C_{i-\frac{1}{2}}^{+(1)} = \frac{3}{2}a^+ \quad C_{i-\frac{3}{2}}^{+(2)} = \frac{-1}{2}a^+$$

The scheme is TVD if and only if

$$C_{i+\frac{1}{2}}^{-(1)} \leq C_{i+\frac{1}{2}}^{-(2)}$$

$$C_{i+\frac{1}{2}}^{+(1)} \geq C_{i+\frac{1}{2}}^{+(2)}$$

In our case, if the coefficients $C^{\pm(1)}$ satisfy the TVD conditions, the coefficients $C^{\pm(2)}$ have the wrong sign. This shows that second-order upwind schemes are not TVD and oscillations will appear around discontinuities.

If the above scheme is re-written in the form,

$$\frac{du_i}{dt} = -\frac{a^+}{2\Delta x} \left[3 - \frac{(u_{i-1} - u_{i-2})}{(u_i - u_{i-1})} \right] (u_i - u_{i-1}) - \frac{a^-}{2\Delta x} \left[3 - \frac{(u_{i+2} - u_{i+1})}{(u_{i+1} - u_i)} \right] (u_{i+1} - u_i)$$

The coefficients are now defined as

$$C_{i-\frac{1}{2}}^+ = \frac{a^+}{2} \left[3 - \frac{(u_{i-1} - u_{i-2})}{(u_i - u_{i-1})} \right]$$

$$C_{i+\frac{1}{2}}^- = \frac{a^-}{2} \left[3 - \frac{(u_{i+2} - u_{i+1})}{(u_{i+1} - u_i)} \right]$$

Now, if the ratios in the R.H.S become large enough to dominate the first term,

$$\left[\frac{(u_{i+2} - u_{i+1})}{(u_{i+1} - u_i)} \right] \geq 3 \quad \text{and} \quad \left[\frac{(u_{i-1} - u_{i-2})}{(u_i - u_{i-1})} \right] \geq 3$$

the schemes are not TVD. Thus to limit these gradients and ensure TVD, non-linear components called limiters, discussed in the next section, have to be implemented.

Limiters

We had discussed earlier that to limit the variable extrapolation at faces to values within the adjacent cell values, we require a limiter. Considering linear reconstruction about i , we require that,

$$U_i - \left(U_{i+\frac{1}{2}}^L - U_i \right) \geq U_{i-1}$$

$$U_i + \left(U_{i+\frac{1}{2}}^L - U_i \right) \leq U_{i+1}$$

We know from the MUSCL approach,

$$U_{i+\frac{1}{2}}^L - U_i = (1/4) \left[(1-\kappa)(U_i - U_{i-1}) + (1+\kappa)(U_{i+1} - U_i) \right] = \delta u^L$$

Thus combining the two equations, we obtain

$$U_i - U_{i-1} \geq \delta u^L$$

$$U_{i+1} - U_i \geq \delta u^L$$

Now to enforce these above constraints, we introduce Ψ^L , such that

$$\Psi^L \delta u^L \leq U_i - U_{i-1}$$

$$\Psi^L \delta u^L \leq U_{i+1} - U_i$$

Thus, we can obtain Ψ^L from

$$\Psi^L \leq \min \left[\frac{U_i - U_{i-1}}{(1/4)[(1-\kappa)(U_i - U_{i-1}) + (1+\kappa)(U_{i+1} - U_i)]}, \frac{U_{i+1} - U_i}{(1/4)[(1-\kappa)(U_i - U_{i-1}) + (1+\kappa)(U_{i+1} - U_i)]} \right]$$

We let, $r^L = \frac{(U_{i+1} - U_i)}{(U_i - U_{i-1})}$, thus

$$\Psi^L(r^L) \leq \min \left[\frac{4}{[(1-\kappa) + (1+\kappa)(r^L)]}, \frac{4 * r^L}{[(1-\kappa) + (1+\kappa)(r^L)]} \right]$$

Similarly, letting $r^R = \frac{(U_{i+2} - U_{i+1})}{(U_{i+1} - U_i)}$, we obtain

$$\Psi^L(r^R) \leq \min \left[\frac{4}{[(1-\kappa) + (1+\kappa)(r^R)]}, \frac{4 * r^R}{[(1-\kappa) + (1+\kappa)(r^R)]} \right]$$

In this research, three limiters, as stated below, were used.

- Min-mod limiter
- Van Leer limiter
- Van Albada limiter

Min-mod limiter

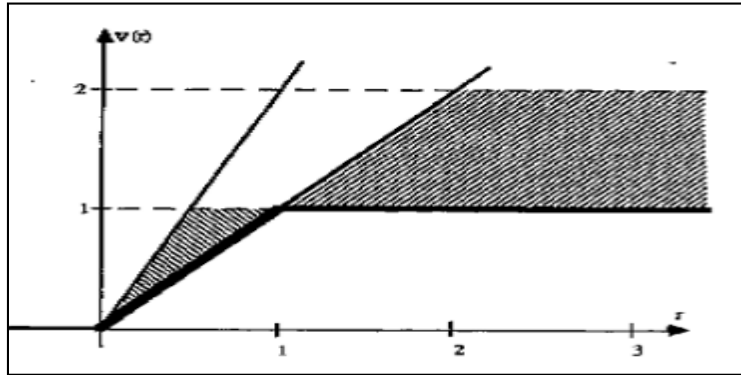


Figure 6. Minmod limiter

The lowest boundary of the TVD region is realized when the limiter incorporates the minimum-modulus (min-mod) function, as seen in Figure 10.

For a fully upwind scheme ($\kappa = -1$),

$$\Psi(r) = \begin{cases} \min(r, 1); & r > 0 \\ 0; & r \leq 0 \end{cases}$$

Here we need to impose $\Psi(r=1) = 1$ to reconstruct linear data.

Van Leer limiter

The Van Leer limiter (Van Leer, 1974 as cited by Hirsch, 1988) includes both extremes of the upper and lower boundaries, as seen in Figure 11.

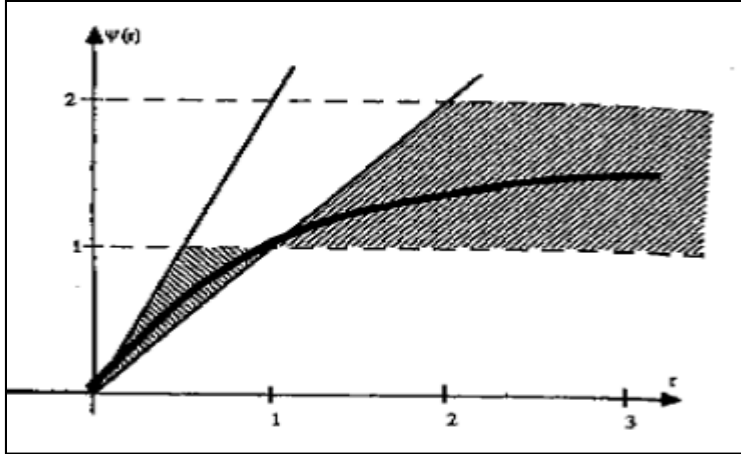


Figure 7. Van Leer limiter

Van leer limiter is defined as,

$$\Psi (r) = \begin{cases} \left(\frac{r + |r|}{1 + r} \right); r > 0 \\ 0; r \leq 0 \end{cases}$$

The Van Leer limiter is noted to be monotone increasing and satisfying the symmetry property.

Van Albada limiter

A limiter similar to Van Leer limiter, but with a smoother behavior was introduced by Van Albada (1982, as cited by Hirsch, 1988). The Van Albada limiter was represented by,

$$\Psi (r) = \begin{cases} \left(\frac{r + r^2}{1 + r^2} \right); r > 0 \\ 0; r \leq 0 \end{cases}$$

A general comparison between the limiters used in this research is discussed in Table 2

Comparison between the limiters

Table 2 shows the comparison between the three limiters used in this research.

Table 1. A Comparison between the Limiters Used in the Research

Van Leer limiter	Van Albada limiter	Min-mod limiter
1. It satisfies the TVD property.	1. It satisfies the TVD property.	1. It satisfies the TVD property.
2. It includes both the extremes of the upper and lower boundaries.	2. It includes both the extremes of the upper and lower boundaries.	2. The min-mod limiter only considers the lower boundary.
3. Van leer limiter possesses a smooth behavior.	3. Van Alabada limiter possesses a smoother behavior than the Van Leer limiter.	3. The min-mod limiter possesses poor smoothness behavior.
4. Van Leer limiter is monotone increasing and satisfies the symmetry property.	4. Van Albada limiter is monotone increasing and satisfies the symmetry property.	4. The min-mod limiter does not satisfy the symmetry property.
5. The Van Leer limiter gives better resolution of both the shock and the contact.	5. The Van Albada limiter gives better resolution of both the shock and the contact than the Van leer limiter.	5. The min-mod limiter resolves contact discontinuities poorly.

Influence of Viscosity

The next stage of the code development saw the introduction of viscous terms, to make the code practically usable. Thus, the Euler equations were converted to Navier-Stokes equations.

Governing Equations

The Navier-Stokes equations in the 2-D conservative integral form are represented as:

$$\frac{d}{dt} \iiint_v \vec{u} dv + \oint_s (\vec{F}_{inv} \cdot \hat{n}) ds = \oint_s (\vec{F}_{vis} \cdot \hat{n}) ds$$

Where,

$$\vec{u} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_o \end{bmatrix} \quad \vec{F}_{inv} \cdot \hat{n} = \begin{bmatrix} \rho(\vec{v} \cdot \hat{n}) \\ \rho u(\vec{v} \cdot \hat{n}) + P^* n_x \\ \rho v(\vec{v} \cdot \hat{n}) + P^* n_y \\ \rho h_o(\vec{v} \cdot \hat{n}) \end{bmatrix}$$

$$\vec{F}_{vis} \cdot \hat{n} = \begin{bmatrix} 0 \\ \tau_{xx} n_x + \tau_{yx} n_y \\ \tau_{xy} n_x + \tau_{yy} n_y \\ \left\{ (\tau_{xx} u + \tau_{yx} v) n_x + (\tau_{xy} u + \tau_{yy} v) n_y + K \left(\frac{\delta T}{\delta x} n_x + \frac{\delta T}{\delta y} n_y \right) \right\} \end{bmatrix}$$

Now splitting the flux vectors into x and y directions,

$$\vec{F}_x = \begin{bmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ \rho h_o u \end{bmatrix} \quad \vec{F}_y = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + P \\ \rho h_o v \end{bmatrix}$$

$$\vec{F}_{vx} = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xx}u + \tau_{yx}v + K \frac{\delta T}{\delta x} \end{bmatrix} \quad \vec{F}_{vy} = \begin{bmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{xy}u + \tau_{yy}v + K \frac{\delta T}{\delta y} \end{bmatrix}$$

where,

$$\tau_{xx} = \text{normal stress in the x-direction} = \tau_{xx} = \lambda(\nabla \cdot V) + 2\mu \frac{\partial u}{\partial x}$$

$$\tau_{yy} = \text{normal stress in the y-direction} = \tau_{yy} = \lambda(\nabla \cdot V) + 2\mu \frac{\partial v}{\partial y}$$

$$\tau_{xy} \text{ and } \tau_{yx} = \text{shear stresses} = \tau_{xy} = \mu * \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) = \tau_{yx}$$

Shear Strain

In this research, only the shear strains, defined below, was added to the x-momentum equation.

$$\tau_{yx} = \mu * \left(\frac{\partial u}{\partial y} \right)$$

$$\tau_{xx} = 2\mu \frac{du}{dx}$$

Where, μ denotes dynamic viscosity calculated using the *Sutherland's law*, as seen below.

$$\mu = \mu_o \left(\frac{T}{T_o} \right)^{(3/2)} \frac{(T_o + S)}{(T + S)}$$

For Air,

$$T_o = 300 \text{ K}$$

$$\mu_o = 1.716 \times 10^{-5} \text{ Ns/m}^2$$

$$S = 110.4 \text{ K}$$

The viscous flux vector in the x-direction becomes,

$$\vec{F}_{vx} = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ 0 \end{bmatrix}$$

The viscous flux vector in the y-direction becomes,

$$\vec{F}_{vy} = \begin{bmatrix} 0 \\ \tau_{yx} \\ 0 \\ 0 \end{bmatrix}$$

For this viscous code, the residuals were calculated by,

$$\text{Residual} = \oint_s (\vec{F}_x \hat{n}_x) ds + \oint_s (\vec{F}_y \hat{n}_y) ds + \oint_s (\vec{F}_v \hat{n}) ds$$

Note. The ' \hat{n} ' vector in the viscous flux term should always point in the opposite direction of the velocity.

Now that the code has been developed, the next step is to verify the proper working of the code. To validate the results of the code, the shock tube problem, as described below, was chosen.

Code Verification

The 2-D Bump

The 2-D bump case was used for the first and second order Euler code verification. Four cases; Subsonic flow ($M=0.3$), Transonic flow ($M=0.7$), Supersonic flow ($M=1.2$ and $M=3$) were run and the results obtained from the code were compared to the results obtained from the commercial software Ansys Fluent.

In fluent, a density based explicit solver was used. Since an Euler code is being validated the inviscid setting was activated. For calculating the fluxes, the Advection Upstream Splitting Method (AUSM), which works very similar to the Van leer flux splitting scheme (used in the code) was used.

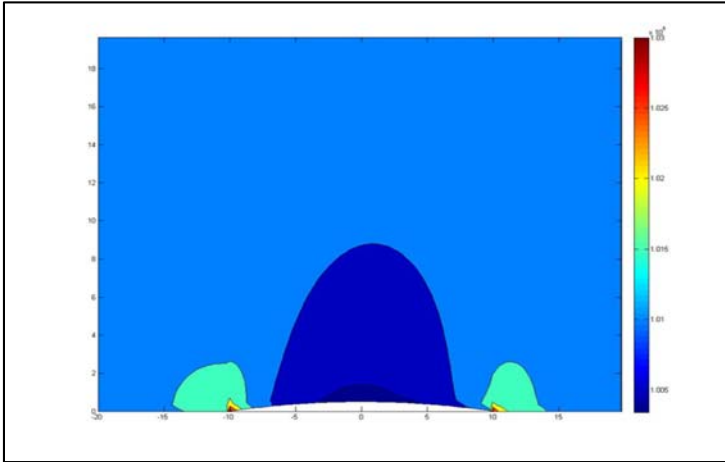
The 2-D bump case tested the 2-D flow capability of the flow and the code's capability in capturing shocks in a steady flow environment. Now, we were interested in evaluating the code's capability to perform in a transient unsteady flow environment. To evaluate this capability of the code, the shock tube problem proposed by Gary Sod, discussed below, was chosen.

Results and Discussion

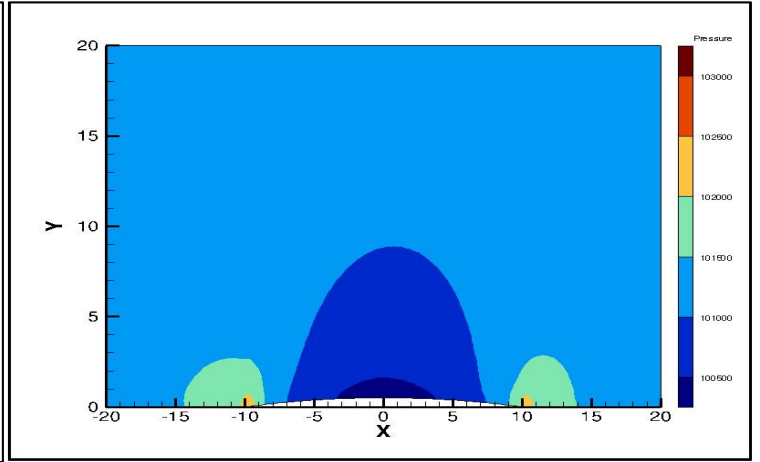
2-D Bump Results

First-order Euler Code:

Pressure Contour at Mach# 0.3

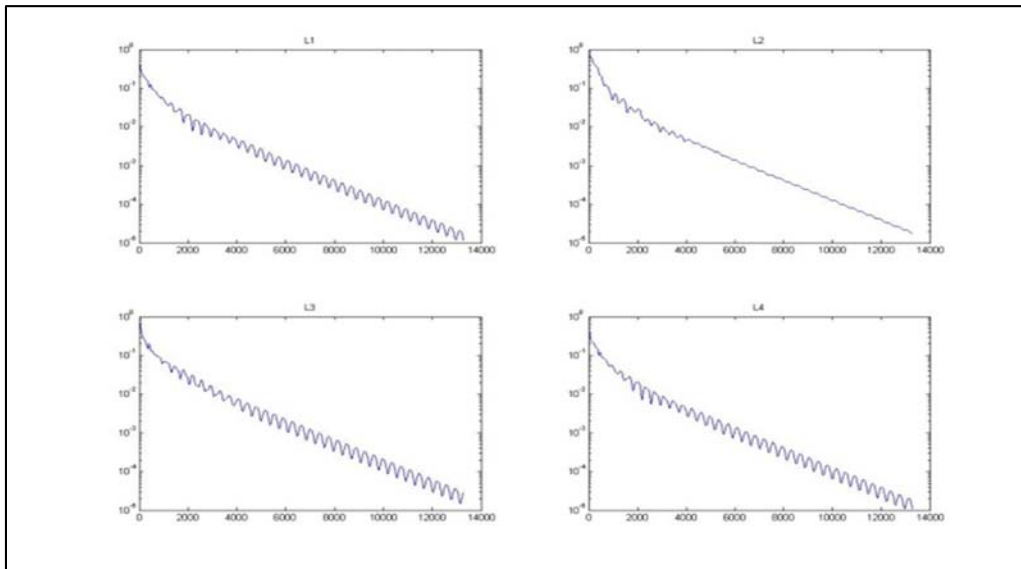


Code

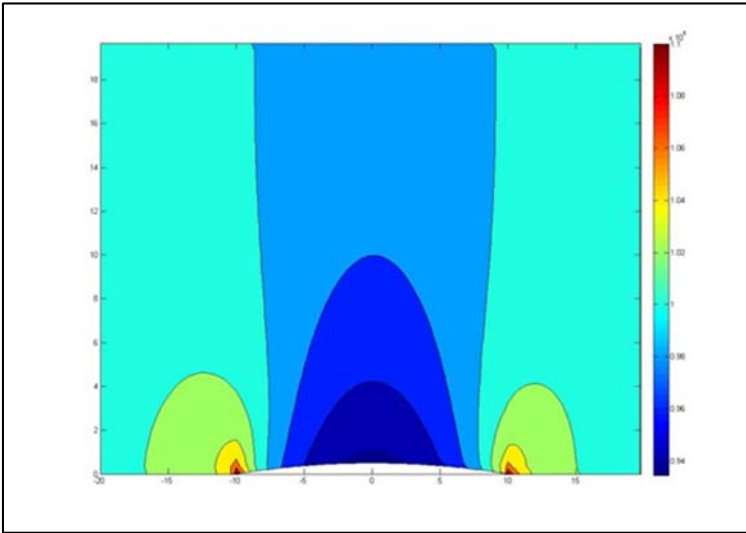


Fluent

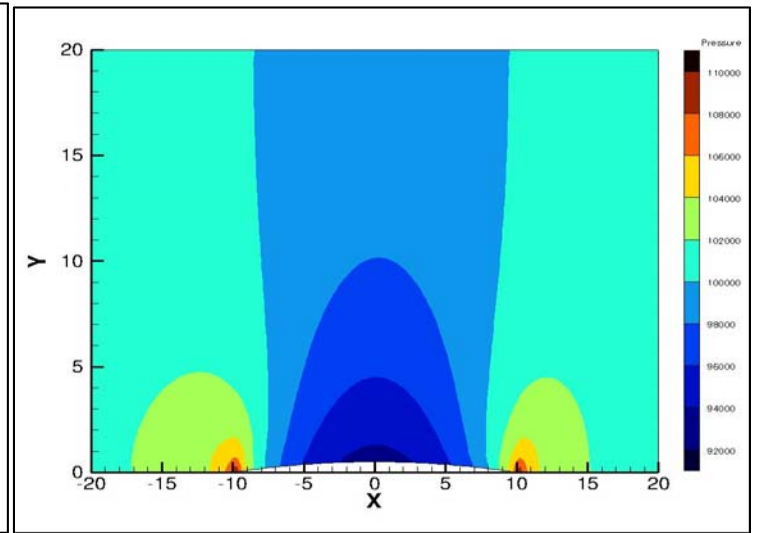
Convergence plot (L2 norms) at Mach# 0.3



Pressure Contour at Mach# 0.7

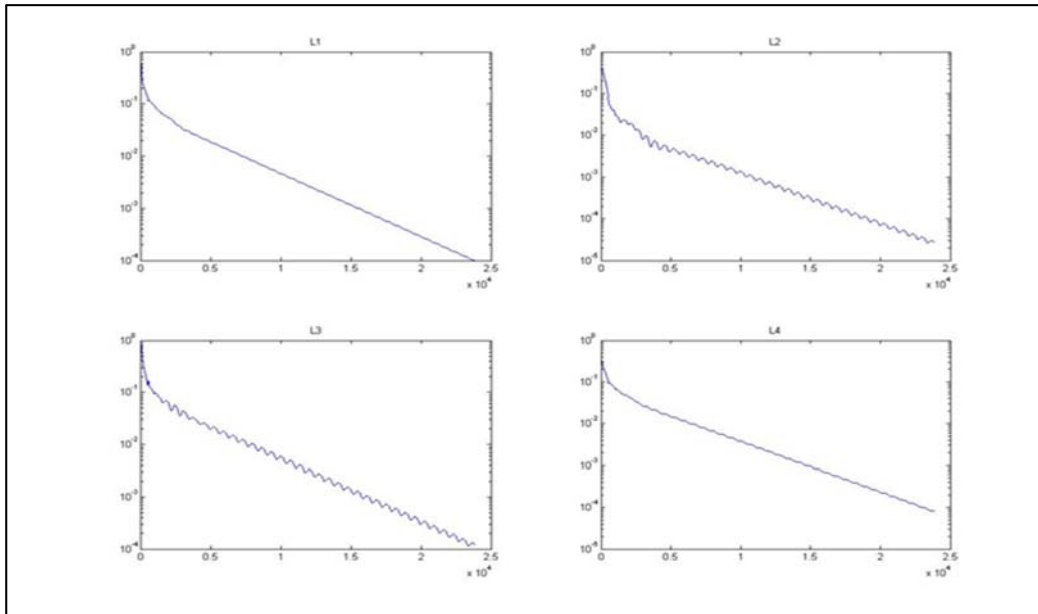


Code

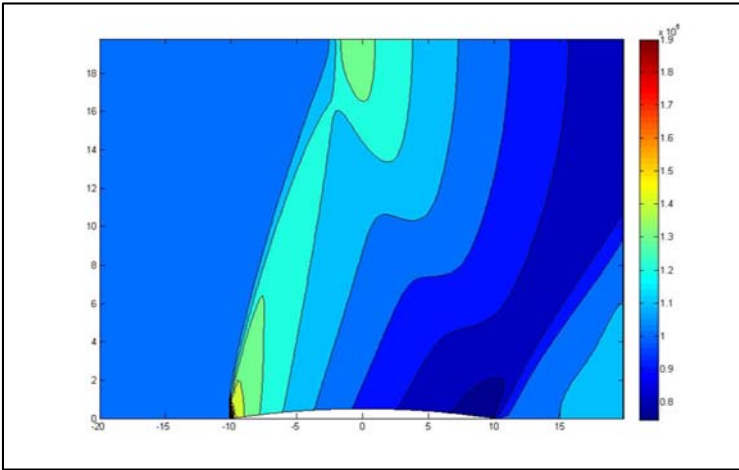


Fluent

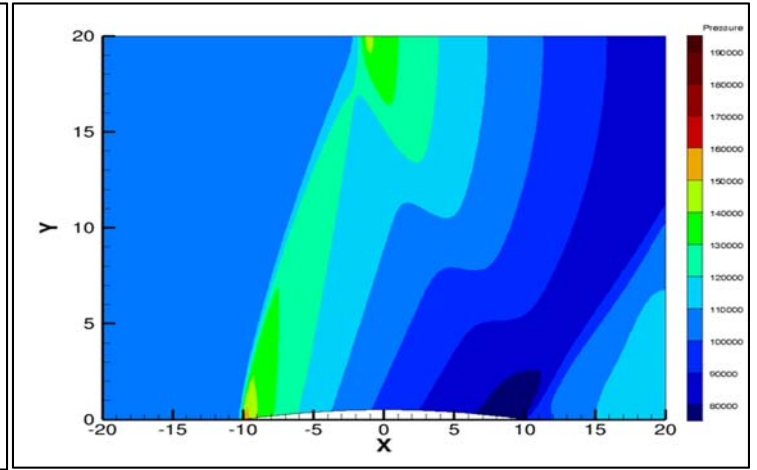
Convergence plot (L2 norms) at Mach# 0.7



Pressure Contour at Mach# 1.2

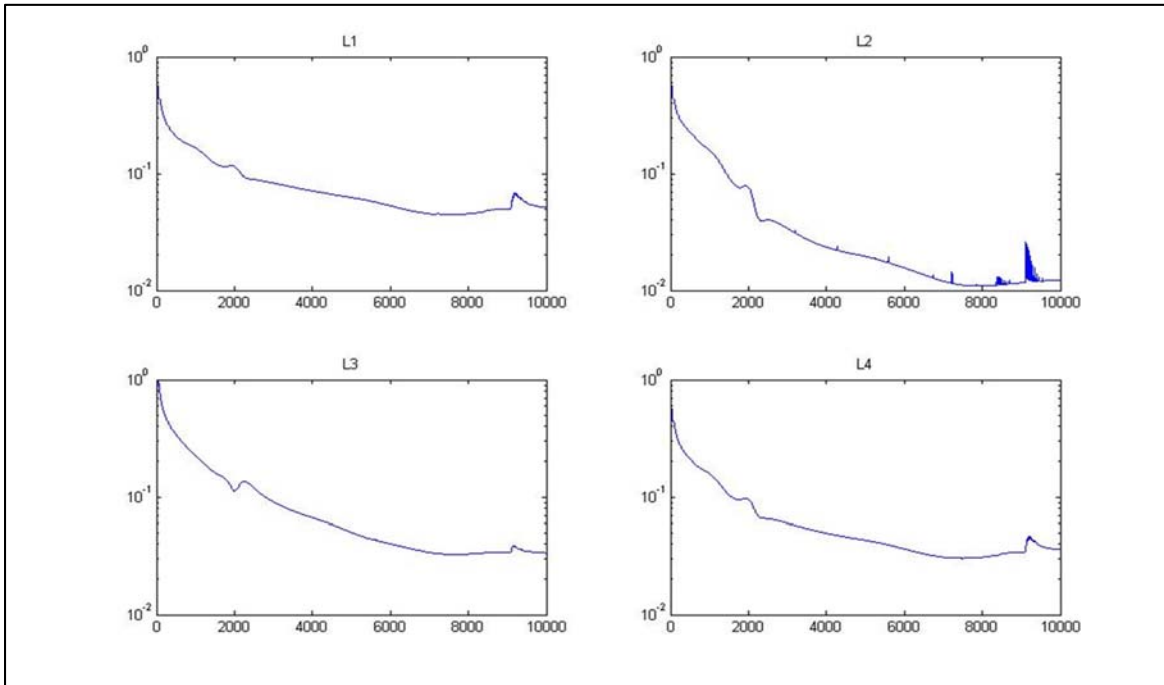


Code (at iter: 10000)

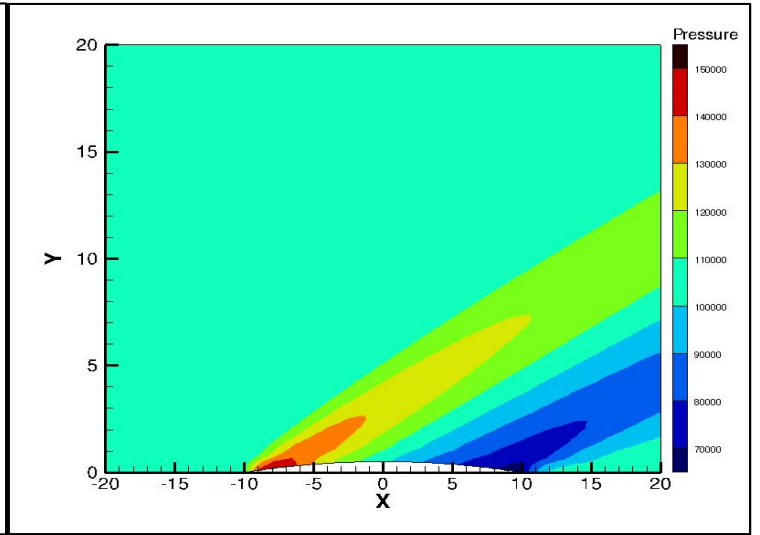
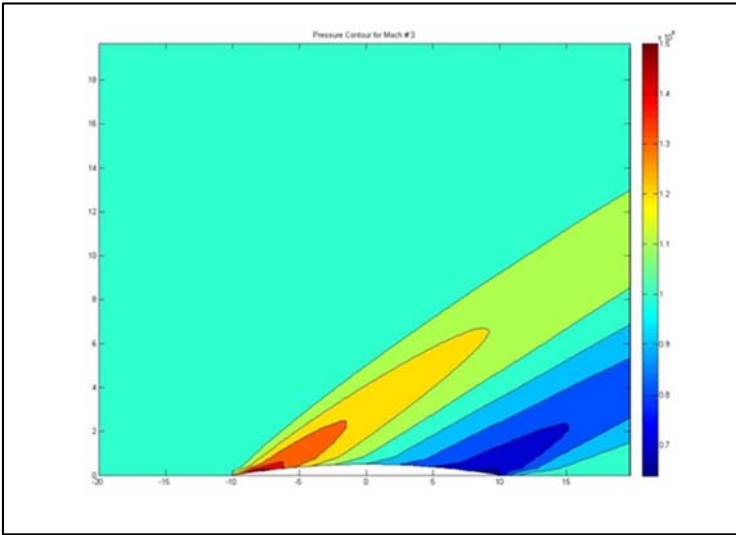


Fluent (at iter: 10000)

Convergence plot (L2 norms) at Mach# 1.2



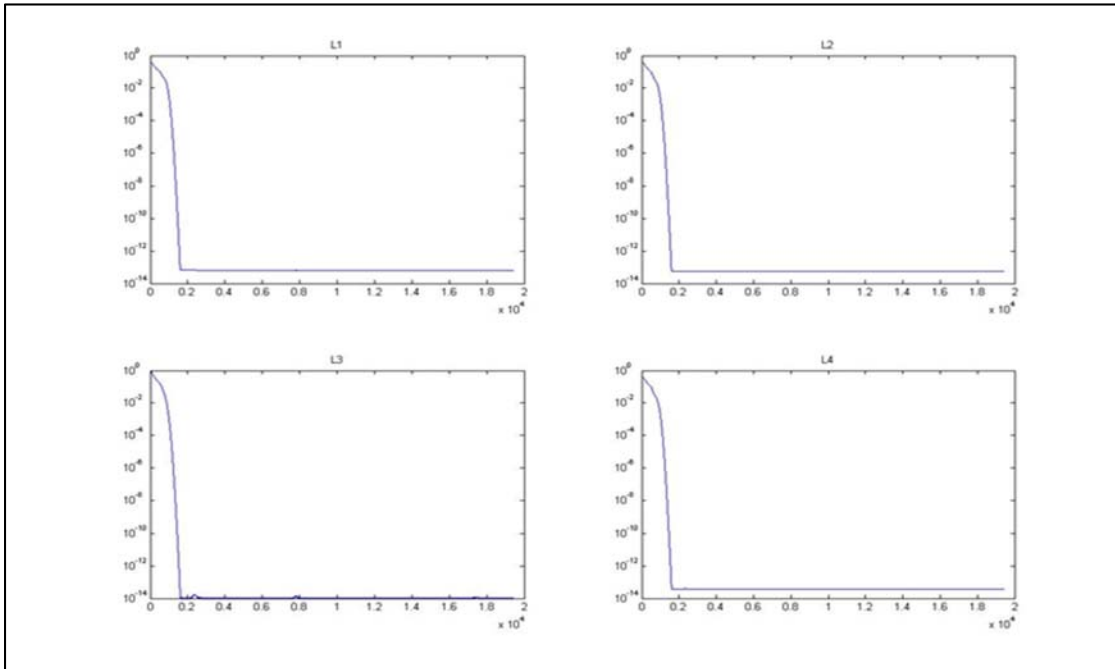
Pressure Contour at Mach# 3



Code

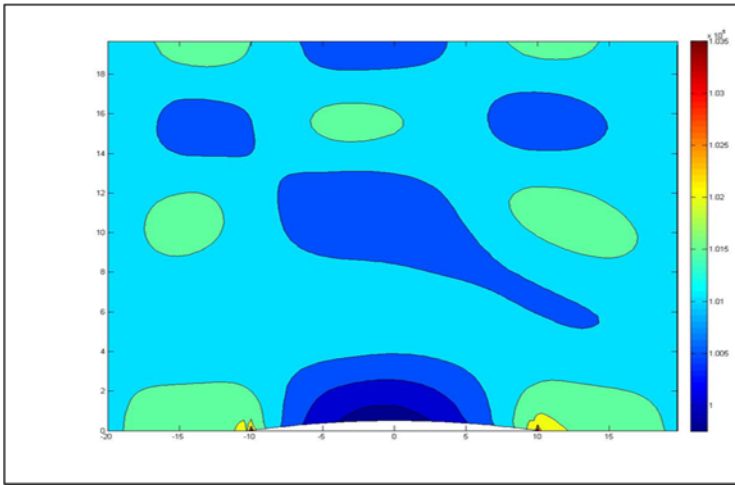
Fluent

Convergence plot (L2 norms) at Mach# 3

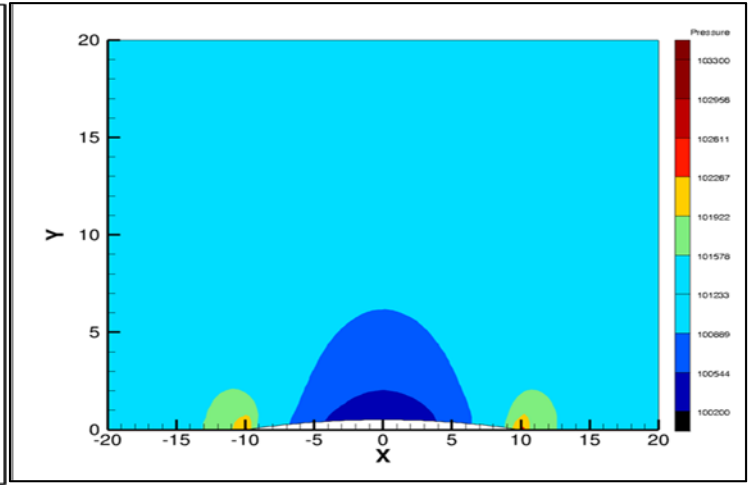


Second-order Euler Code:

Pressure Contour at Mach# 0.3

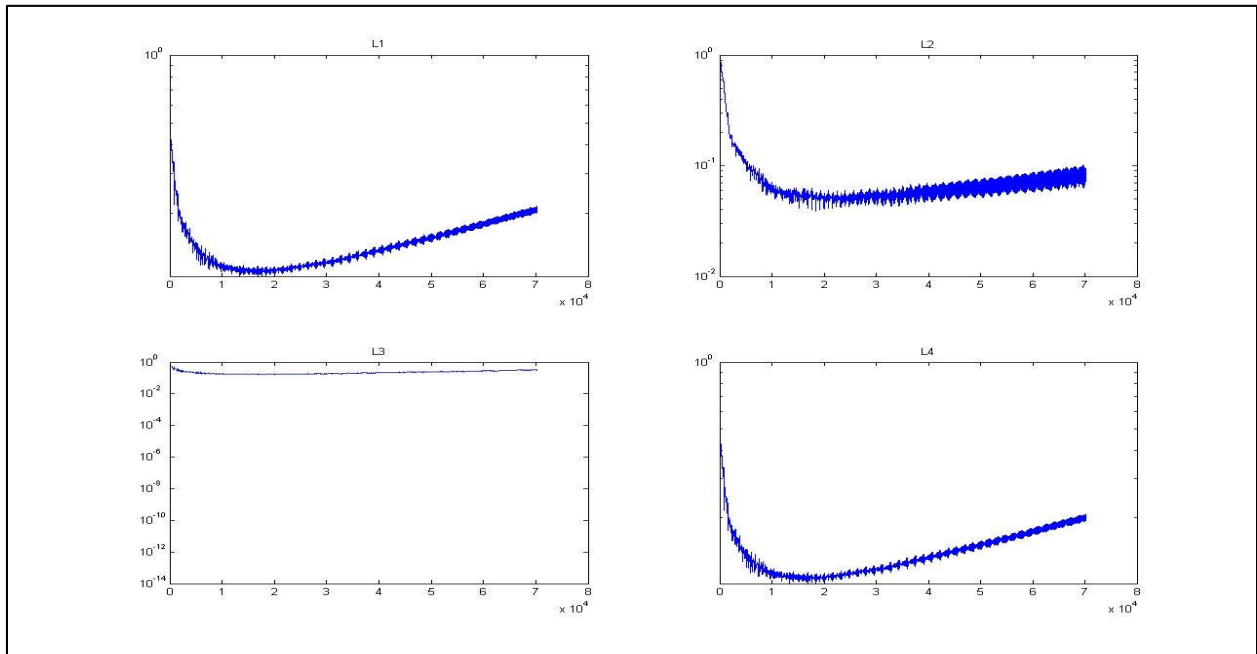


Code



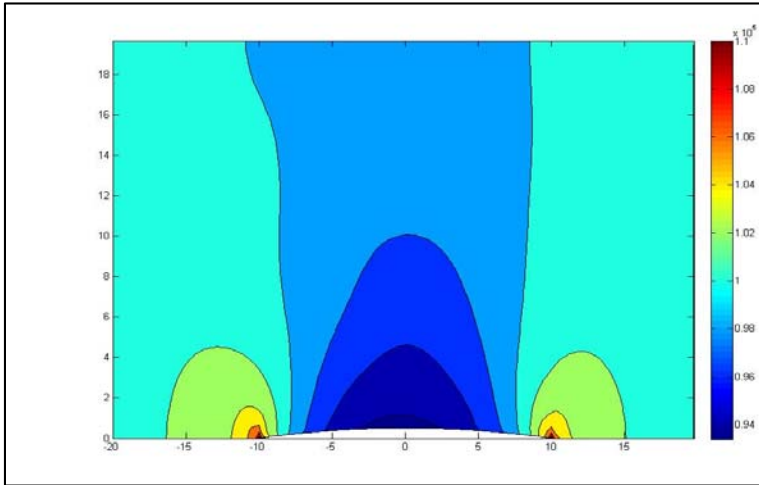
Fluent

Convergence plot (L2 norms) at Mach# 0.3

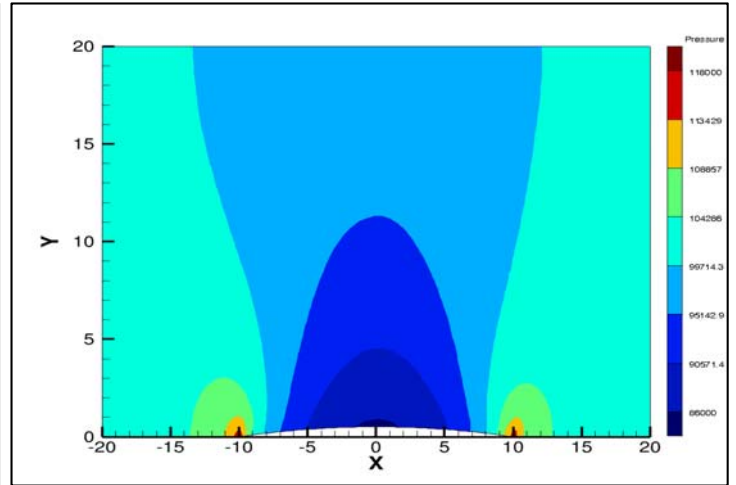


The results obtained from the code are not in accordance with the results obtained from Fluent. One possible reason could be that the code is not sophisticated enough to handle very low speed flows, which act almost like an incompressible flow, as the code is a density-based solver. The second-order code is also less dissipative which makes it difficult to reach convergence for low speed flows. The oscillations are due to the waves bouncing back from the north wall on a continued period of time

Pressure Contour at Mach# 0.7

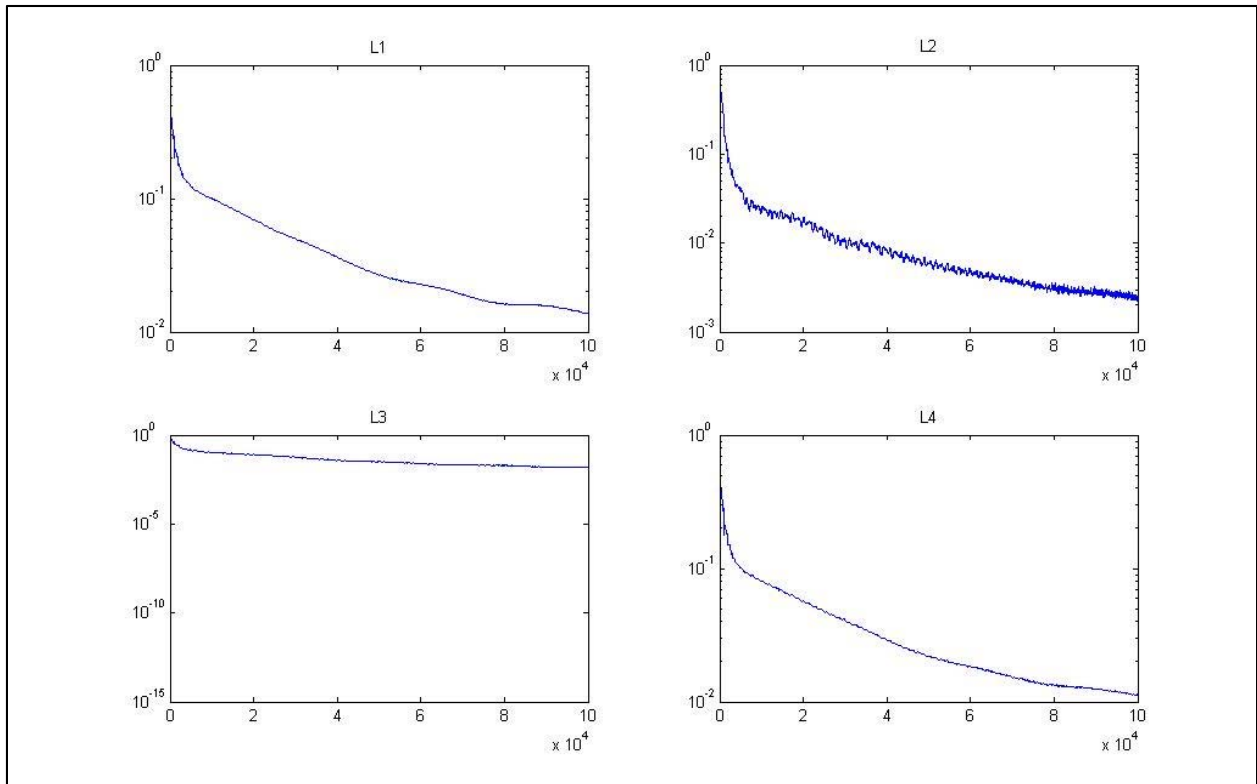


Code

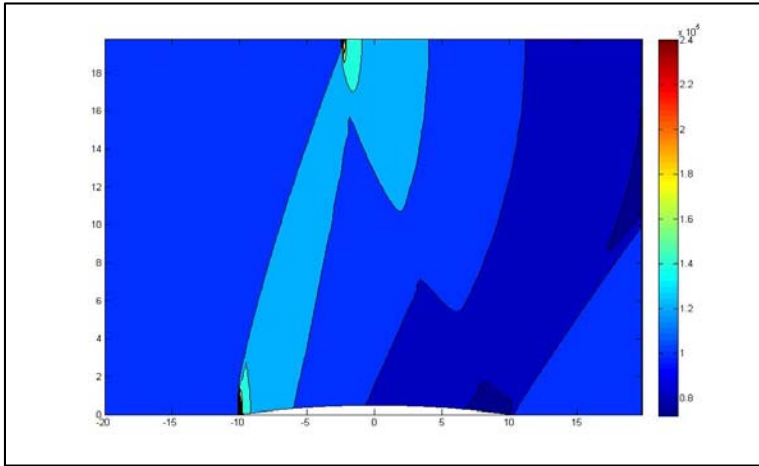


Fluent

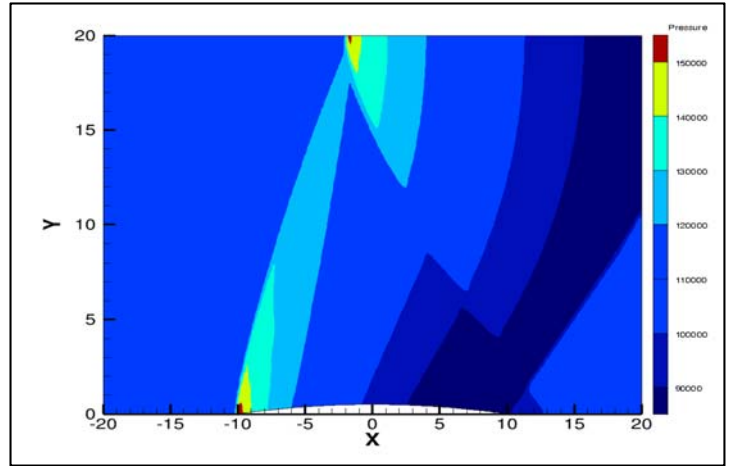
Convergence plot (L2 norms) at Mach# 0.7



Pressure Contour at Mach# 1.2

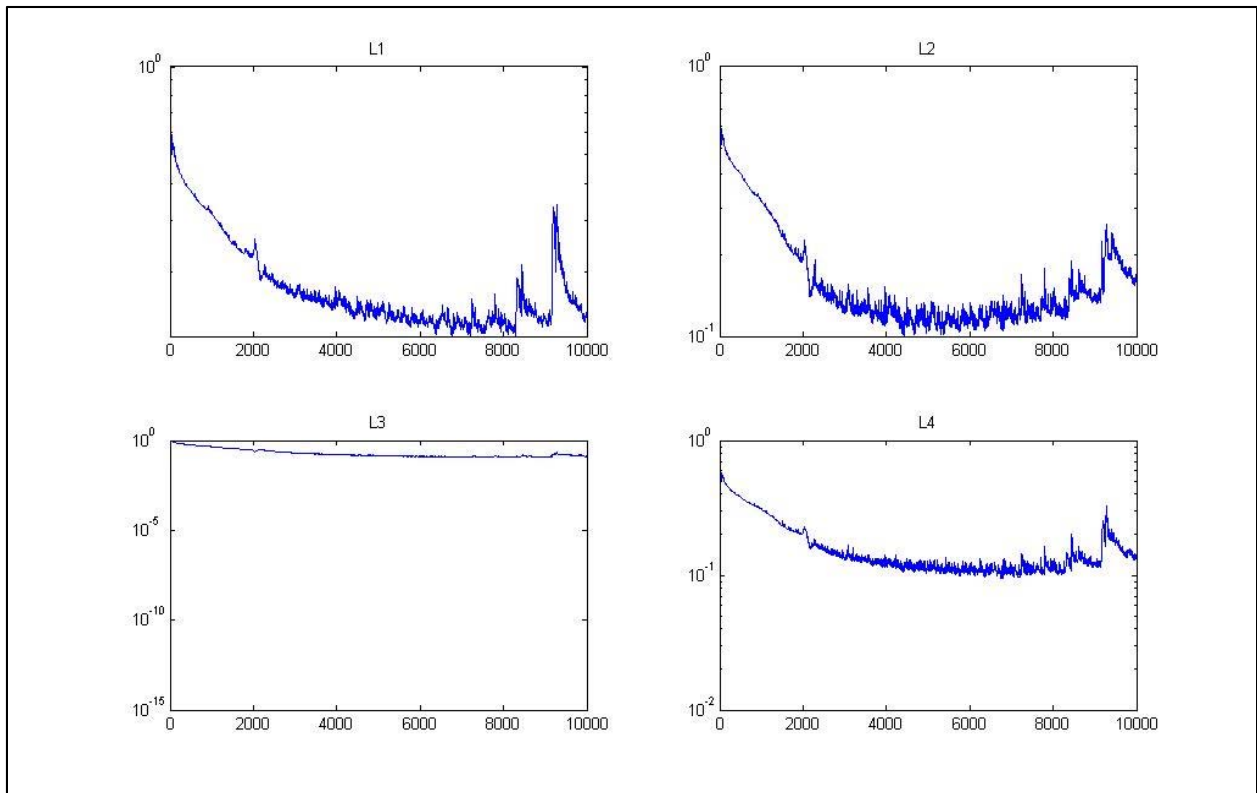


Code (at iter: 10000)

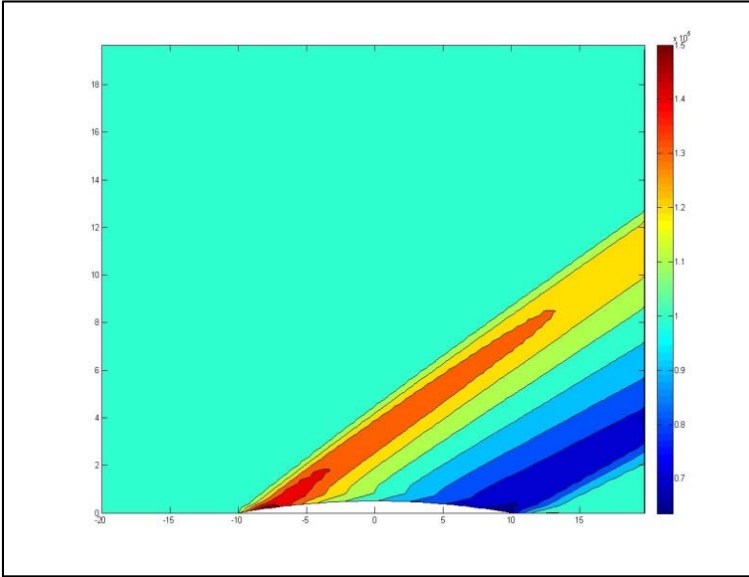


Fluent (at iter: 10000)

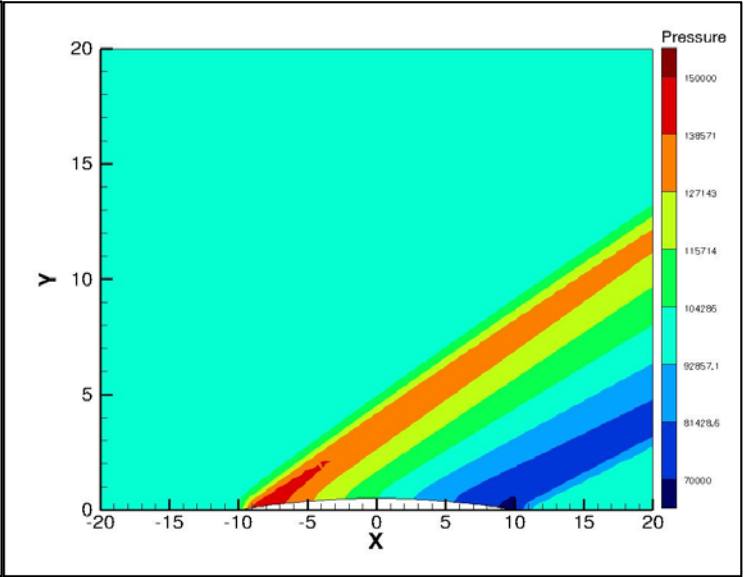
Convergence plot (L2 norms) at Mach# 1.2



Pressure Contour at Mach# 3

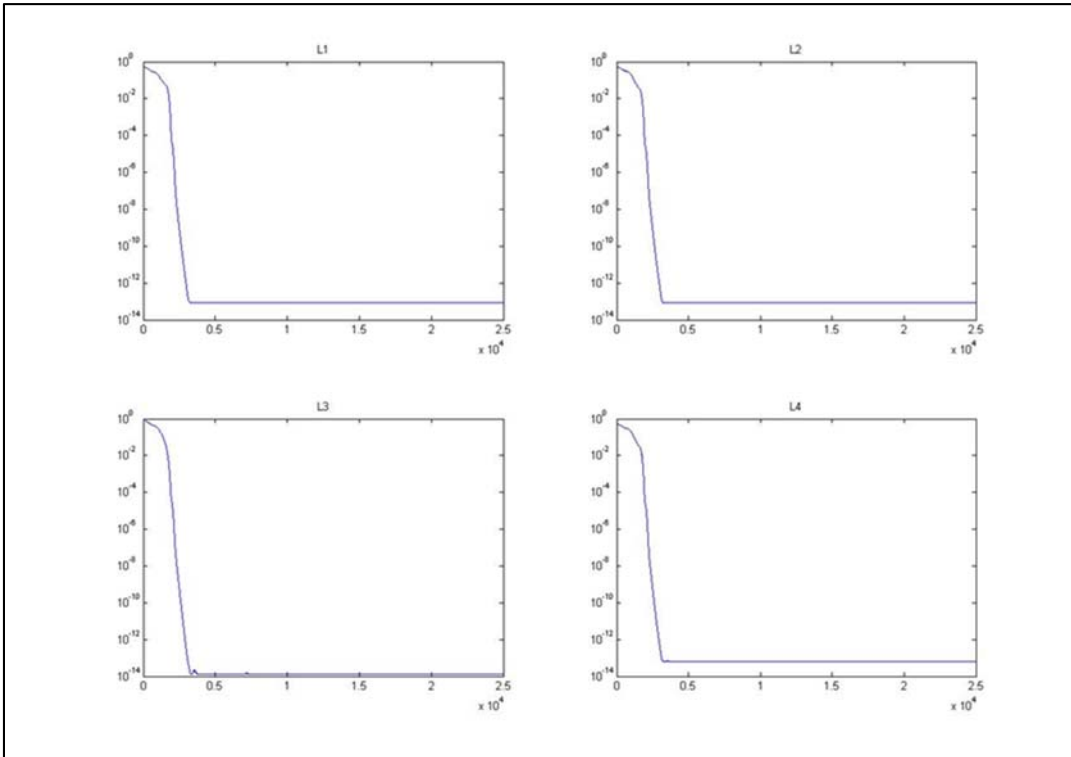


Code



Fluent

Convergence plot (L2 norms) at Mach# 3



The Shock Tube Problem

Gary A. Sod (1978) modeled a shock tube, as seen in Figure 9, of unit length, to test the ability of various algorithms in solving fluid dynamics problems which has a shock wave behavior. The diaphragm in the center separates the two regions having different pressures and densities.

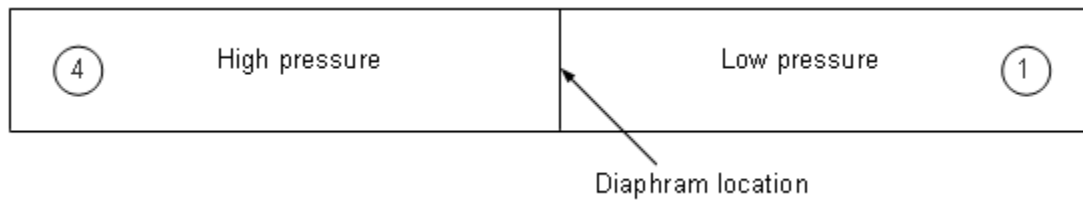


Figure 8. The Shock Tube

Initial Conditions:

$$\frac{P_1}{P_4} = 0.1 \text{ (as shown in Figure 4)}$$

$$\frac{\rho_1}{\rho_4} = 0.125$$

$u_1 = u_4 = 0$ (fluids are initially at rest)

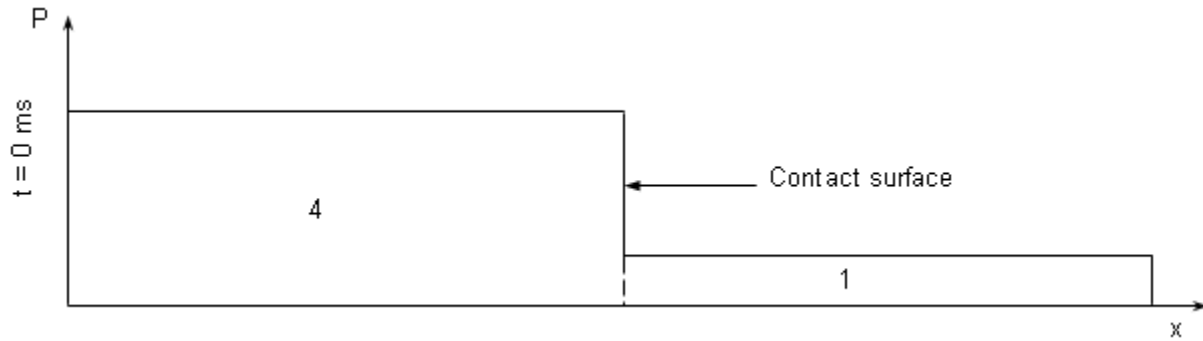


Figure 9. Initial pressure distribution in the shock tube

At time, $t > 0$, the diaphragm is broken. We notice that the unsteady normal shock waves move to the right and an unsteady isentropic expansion wave travels to the left. At a time, say $t = 6.1$ ms, when both the waves have not hit the walls, the flow variables were calculated and plotted, as seen in Figure 11.

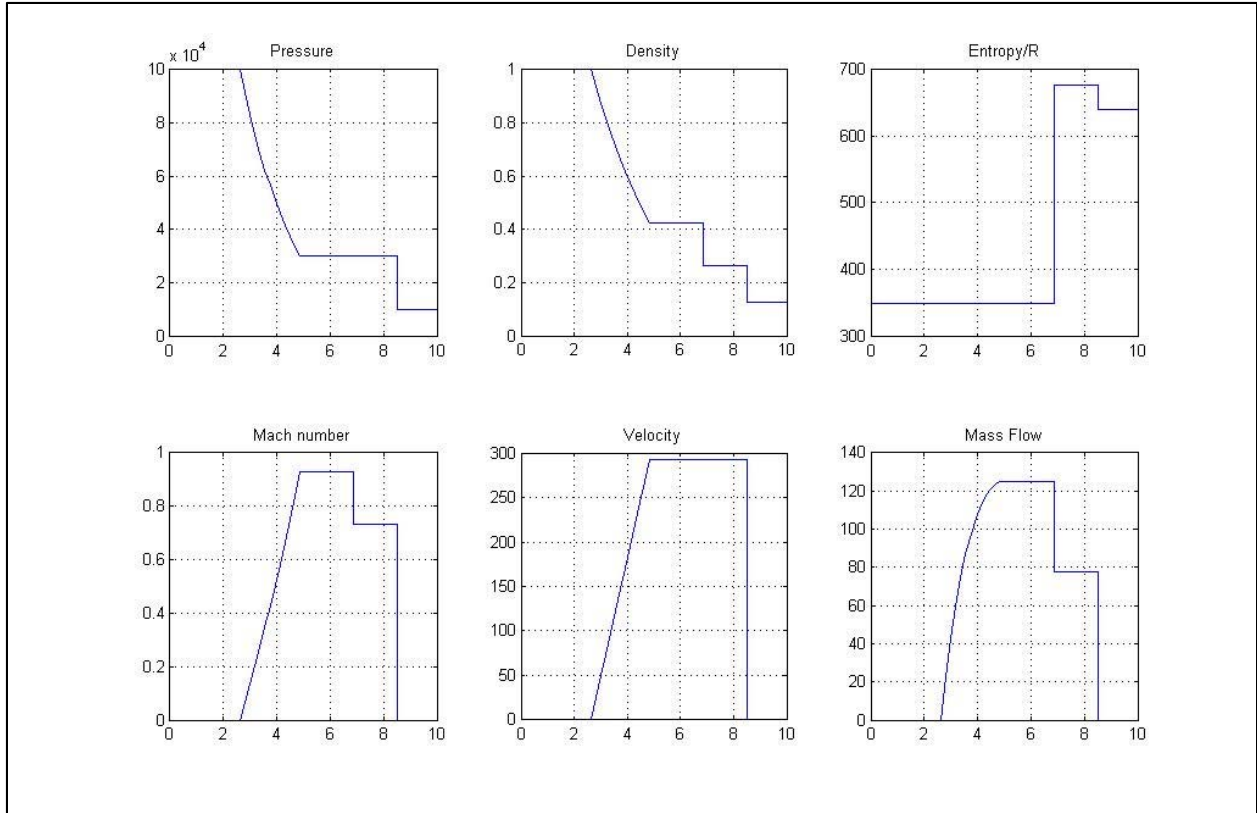


Figure 10. Shock tube analytical results at $t=6.1$ ms

The results of the code were verified watching how well the plots matched with the above plot. These results are discussed in the Results and Discussion section.

Shock Tube Results

First-order Euler Code:

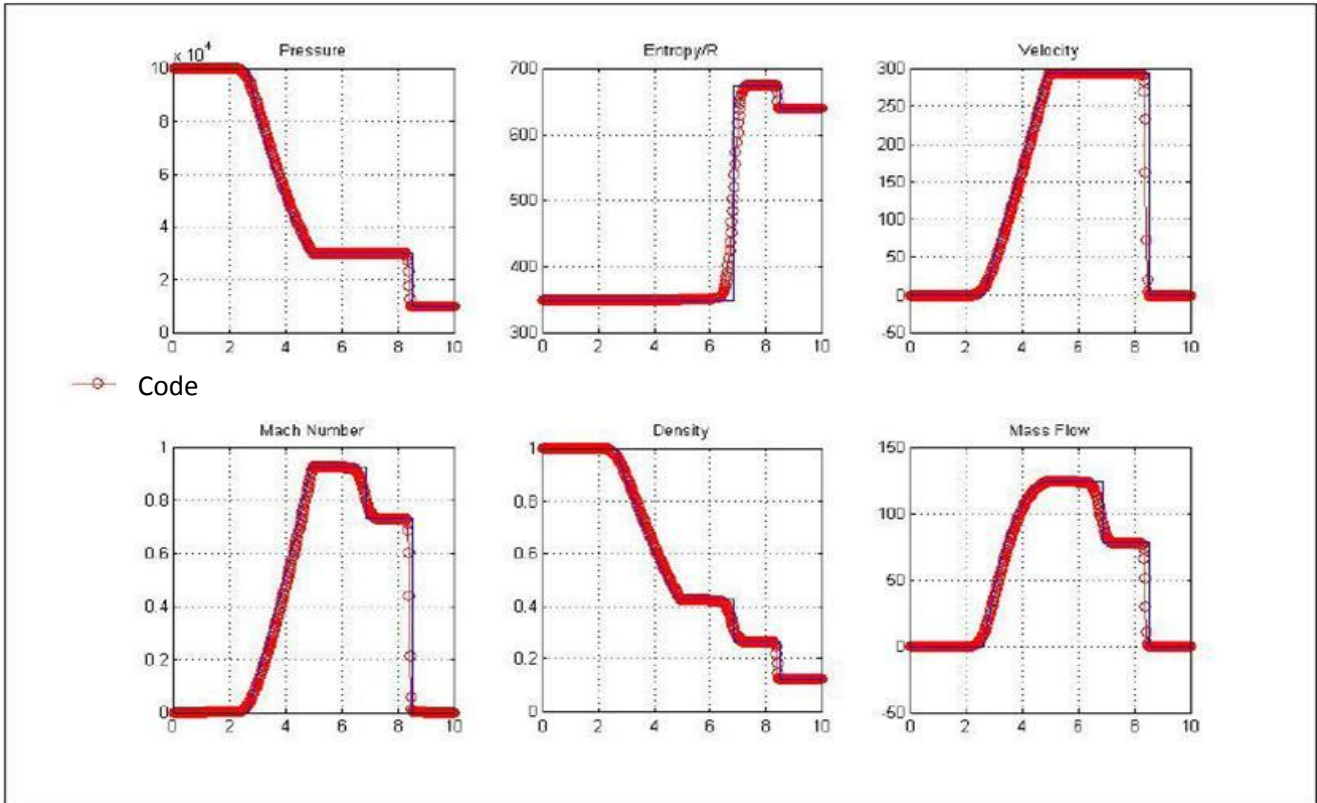


Figure 11. Results obtained using the first-order Euler code

The first order inviscid code produced excellent results and matched very well with the theoretical results. Since the first order accurate case is naturally a TVD scheme, we notice a smooth curve, without any undesirable oscillations and instabilities, and matches well with the theoretical results.

Next, the second order inviscid results are presented with and without the use of different limiters. A comparison between the limiters can be seen in Table 2.

Second-order Euler code without limiters:

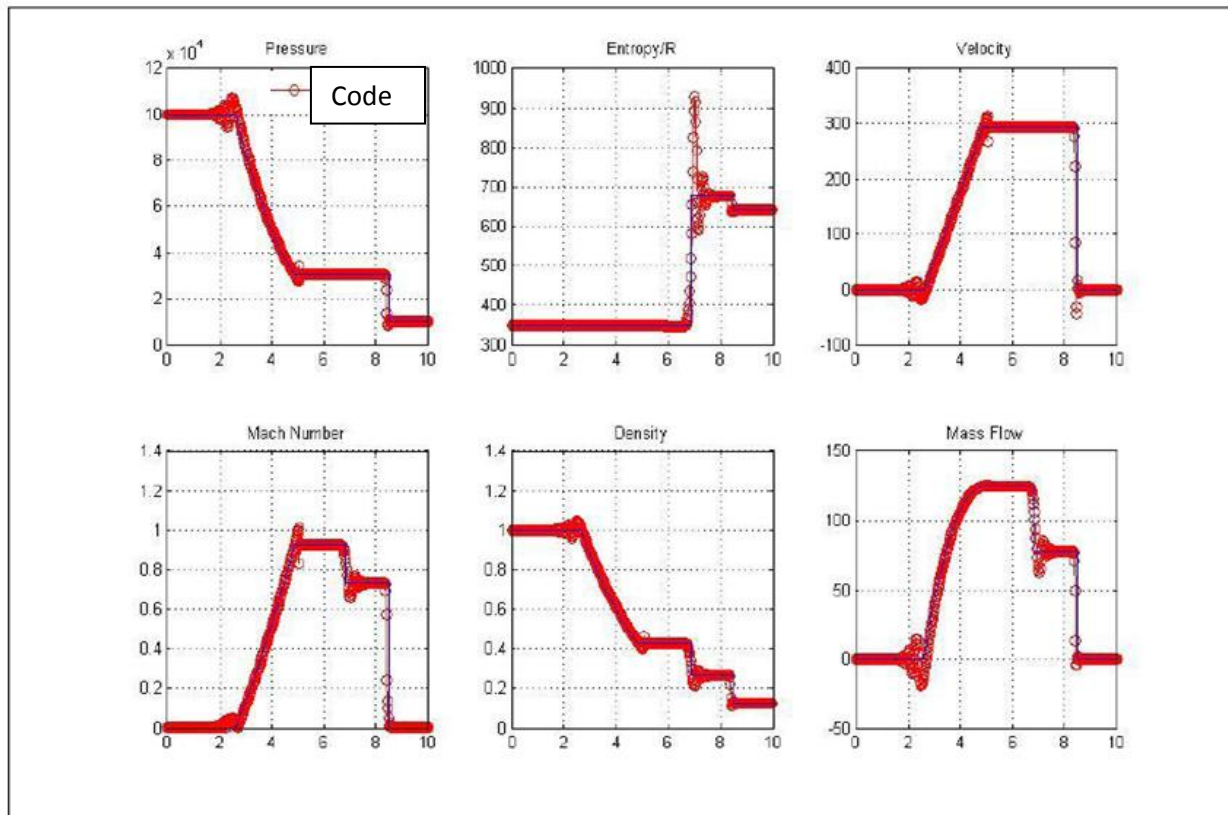


Figure 12. Results obtained from the second-order Euler code without limiters

We see a sharp capture of shock fronts in the second order code. An interesting feature to notice in the second order result is the undesirable oscillations. The reason behind this is that the second order scheme is not a TVD scheme and hence the monotonicity condition is violated. To control the magnitudes of these oscillations the limiters were used. Three different limiters were used in this present research and their pros and cons were compared.

Second-order Euler code using Van Leer limiter:

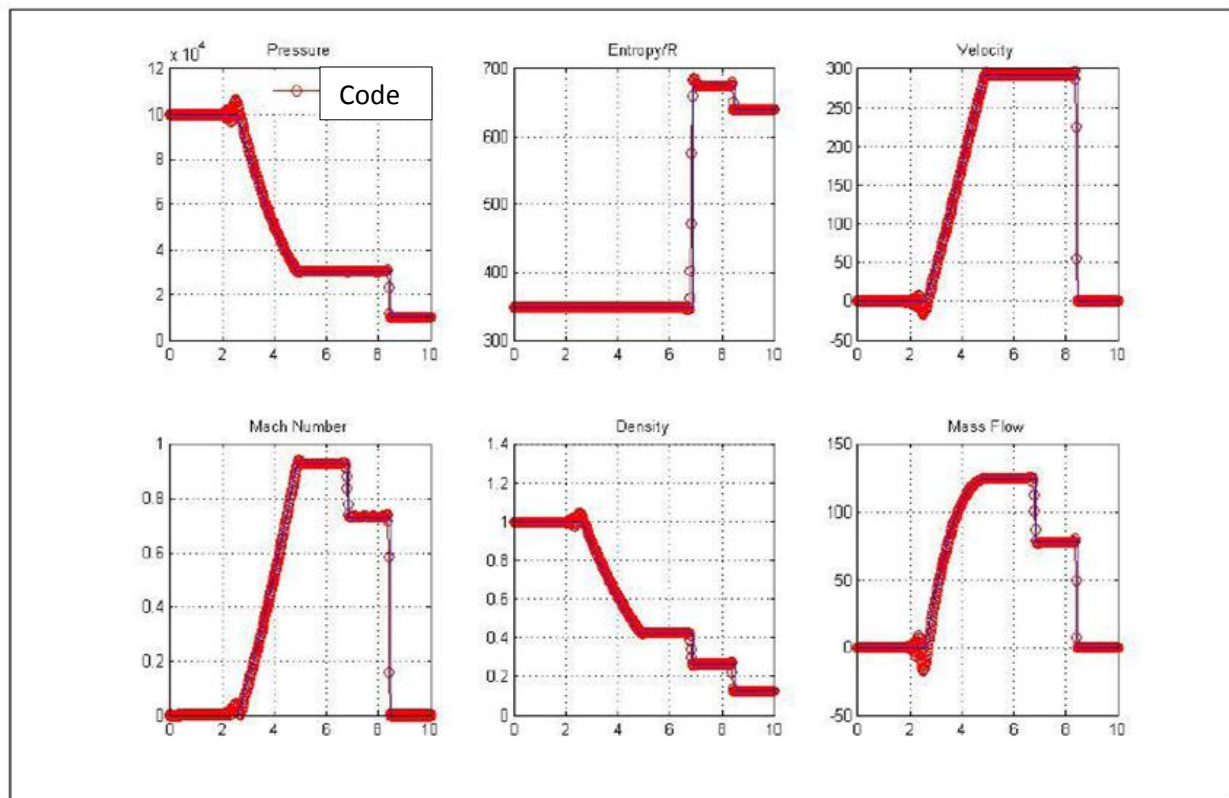


Figure 13. Results obtained using the second-order Euler code using Van Leer limiter

The Van leer limiter performed well in reducing the undesirable oscillations and maintaining the gradients. However we can still see little bumps in the curve which is not completely removed.

Second-order Euler code using Van Albada limiter:

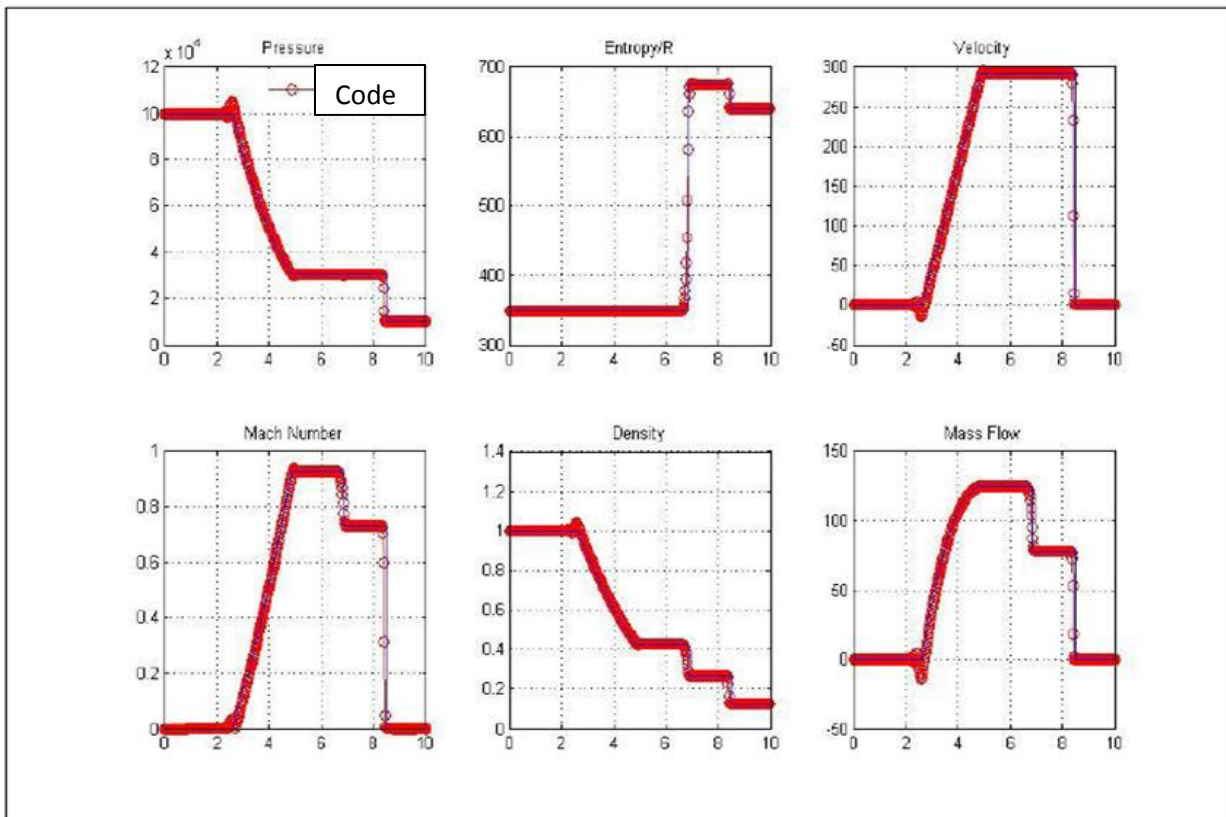


Figure 14. Results obtained from the second-order Euler code using Van Albada limiter

The Van Albada limiter performed better than the Van leer limiter in damping the oscillations. The peaks are lesser in magnitude compared to the ones using the Van leer limiter as seen in the above figure. The difference is not very clear from the figure, but from the values obtained, the above statement is justified.

Second-order Euler code using Min-mod limiter:

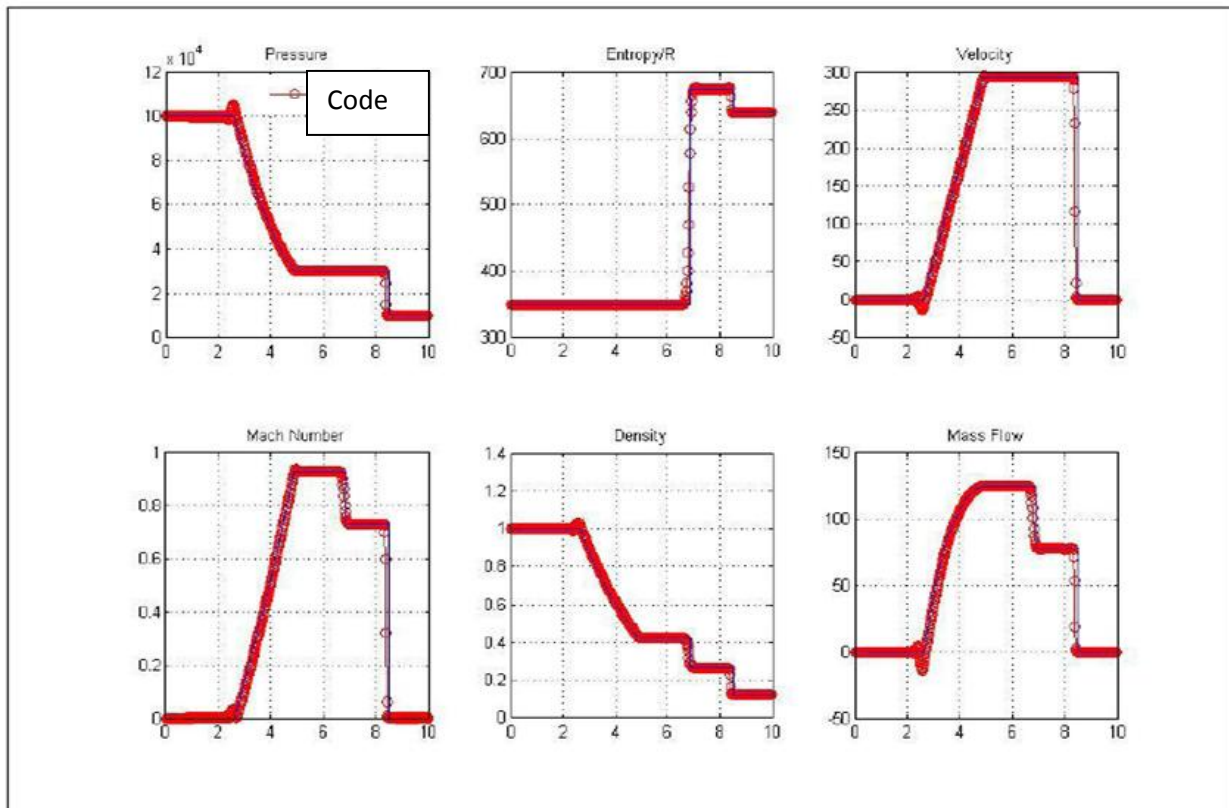


Figure 15. Results obtained from the second-order Euler code using Min-mod limiter

The min-mod limiter, as seen in the above figure, did not perform very well compared to the other two limiters in damping the oscillations and producing a smooth gradient. The contact discontinuities are resolved poorly. Thus from the results, it is seen that the Van Albada limiter has performed the best among the three limiters used in terms on damping the oscillations and resolving contact discontinuities.

Since the Van Albada limiter performed well, it was chosen to study the next capability of the code, again using the shock tube problem. The second-order Euler code using the Van Albada limiter was run till the shock waves were not seen and steady state was reached. An

interesting scenario was noted here, when the expansion wave and the compression shock wave met each other after bouncing back from the walls. The Euler code could not capture the instability effects that arose when the shocks met and the pressure plot had a huge spike. A reason for such a happening is unknown. A probable reason for this occurrence could be the instability caused by the huge velocity gradient near the wall ($u, v=0$) when the two waves, expansion and the compression wave, met each other. Lowering the time-step just before that instant could be a possible solution to overcome such an effect.

Converting the code to a Navier-Stokes code was suggested to check if the Navier-Stokes code to capture the physics well. Since the flow was basically a 1-D flow with no heat addition, only the “ du/dx ” and the “ du/dy ” terms were deemed important. The results of the Navier-Stokes code are discussed below.

First-order Navier-Stokes code using the Van Leer limiter

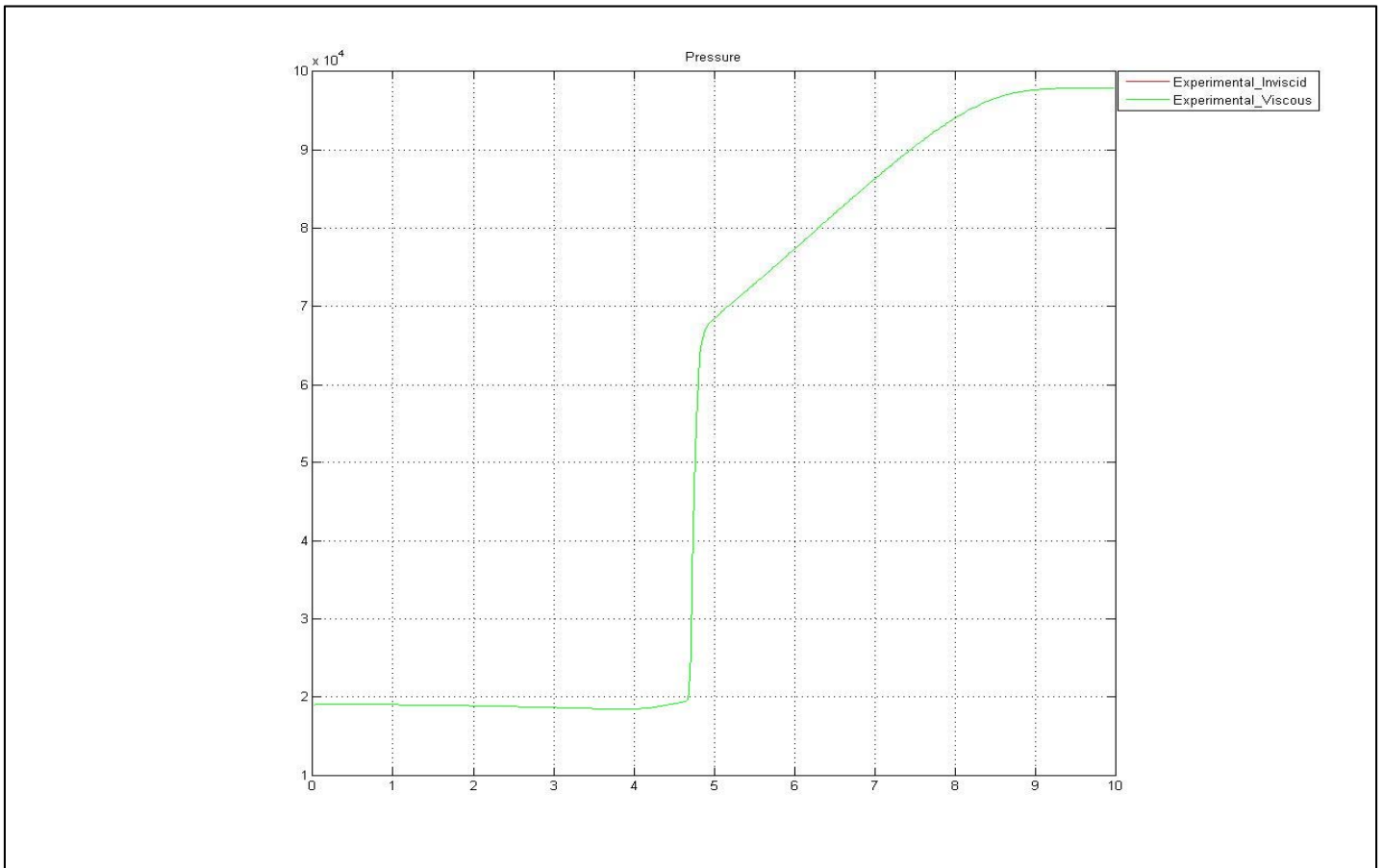


Figure 16. Comparison between the first-order viscous and inviscid solutions using Van leer limiter

No differences were noticed between the viscous and the inviscid solutions in the first order case because of the major domination of the artificial viscosity. Since the grid is coarse and the flow keeps oscillating between the walls, we really do not notice much of the boundary layer development. Thus, the magnitude of the “ du/dy ” term is very low. Also, because of the first order accuracy, the magnitude of the “ du/dx ” term, which is the major contributor for the viscous effect in our case, is very minimal.

Second-order Navier-Stokes code using the Van Leer limiter:

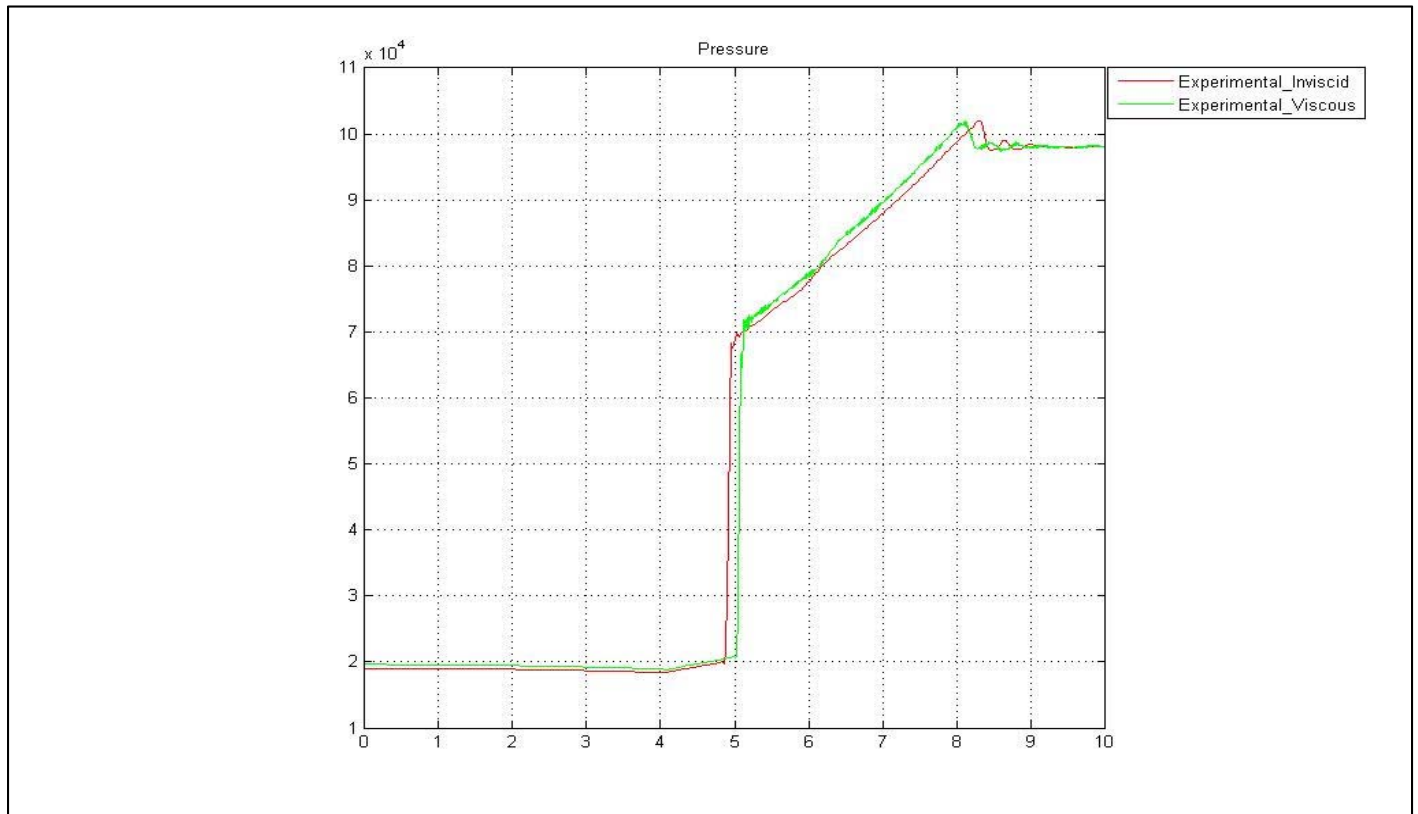


Figure 17. Comparison between the second-order viscous and inviscid codes using the Van leer limiter

In this case we start noticing difference in the viscous and the inviscid case after the waves reflect off the walls. We notice that the wave in the viscous case is moving slower than the inviscid counterpart. This alone does not show a confirmative viscous effect because the pressure wave in the viscous case shows a higher gradient than the inviscid case. An explanation for such an occurrence is unknown. A probable explanation could be that the artificial viscosity still dominates the viscous effects produced by the “ du/dx ” and the “ du/dy ” terms.

Raising the viscosity to a higher value could be a possible solution to clearly notice the viscous effects, but a higher viscosity value should be accompanied with a lower time step to maintain stability. Another solution could be to use a higher order code to increase the magnitude of the “du/dx” and the “du/dy” terms, and outdo the artificial viscosity.

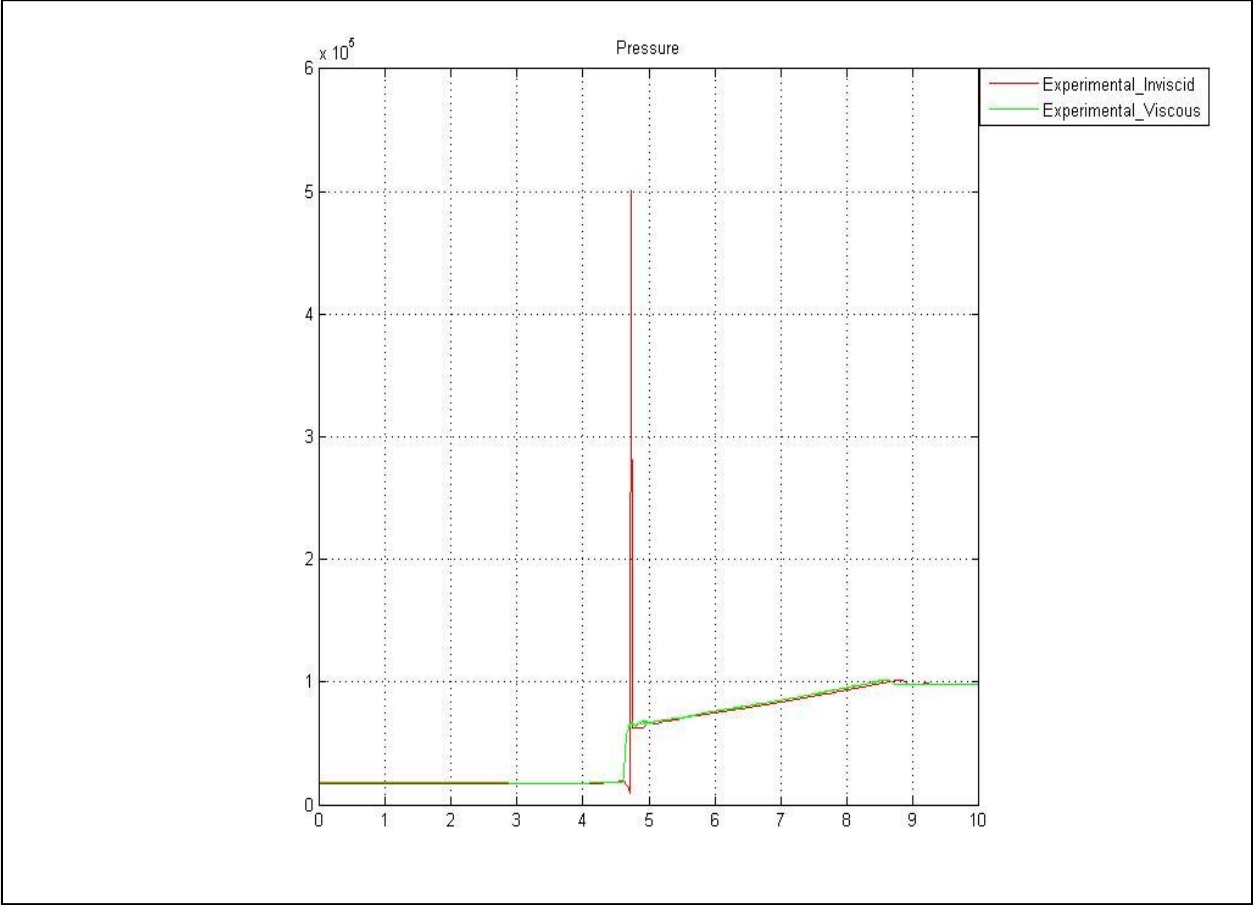


Figure 18. A second order viscous vs inviscid comparison

As expected, we see that the Navier-Stokes code dealt with the instability caused when the two shocks met and captured the physics well.

Suggestions for Future Work

An important suggestion for students who may use this code is to convert the code into FORTRAN for achieving increased computational speeds. An implicit scheme could be implemented to overcome some of the instability seen and to get better results. A higher order code can easily be formulated from this present code to get better results. Also the Roe scheme which is less dissipative than the Van leer scheme can be used carefully (without violating the entropy condition) along with the Superbee limiter to see how it fares in capturing the shocks and boundary layer.

To better understand the viscous effects and the boundary layer effects, the flat plate problem with a fine grid can be undertaken and coded with the appropriate boundary conditions.

Efforts should also be made to understand the instability effects when two shock waves meet in an unsteady flow problem. Understanding the temperature gradients across shocks in a transient flow case will prove useful in a lot of practical applications.

References

- Anderson, J. D. (1995). *Computational Fluid Dynamics: The Basics with Applications*. McGraw-Hill.
- Engblom, W. (2008). Introduction to CFD, Notes.
- Hirsch, C. (1988). *Numerical Computation of Internal and External Flows, Vol. II: Computational Methods for Inviscid and Viscous Flows*. New York: John Wiley & Sons.
- Leer, B. V. (1982). *Flux-Vector Splitting for the Euler Equations*. Hampton: Nasa Langley Research Center.
- Leer, B. V. (1974). Towards the Ultimate Conservative Difference Scheme,II. Monotonicity and Conservation combined in a second order scheme. *Journal of Computational Physics* , 361-370.
- Leer, B. V. (1979). Towards the Ultimate Conservative Difference Scheme. V. A Second-Order Sequel to Godunov's Method. *Journal of Computational Physics* , 101-136.
- Sod, G. A. (1978). A Survey of Several Finite Difference Methods for Systems of Non-Linear Hyperbolic Conservation Laws. *Journal of Computational Physics* , 1-31.
- Sweby, P. K. (1984). High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws. *SIAM Journal on Numerical Analysis* , 995-1011.

Appendix

The Matlab Code

```
% ARJUN VIJAYANARAYANAN
% AE 700: THESIS

% SECOND ORDER MATLAB CODE SOLVING THE INVISCID TWO-DIMENSIONAL EULER
EQUATIONS IMPLEMENTING THE VAN LEER SPLIT FLUXES SCHEME AND THE MUSCL
APPROACH

% STRUCTURED GRID
% THE GRID COORDINATES IS LOADED USING "grid_loader.m", WHICH READS THE
'.grd' OUTPUT GRIDGEN FILE.
% THE FREESTREAM CONDITIONS are called from "freestream.m".
% All three files should be in the same folder.

% GAME ON

clear all ;close all
clc;
format long
%%dbstop if error
% Adding File to path
addpath('/P:/ae699/arj code/arj_sod_sec/')
fname = 'sod3'; % set filename
[x,y,z,nx,ny,nz] = grid_loader(fname);
mesh(x,y,x*0);
%axis vis3d;
title ('Contour plot showing the mesh')

    for j = ny-2:ny-1;
    for i = nx-2:nx-1;
        pre_allocate
    end
end

    for j = 1:ny-1;
    for i = 1:nx-1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Change in x and y on each face
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        dx_ef(i,j) = x(i+1,j+1)-x(i+1,j);    % @ the East Face
        dy_ef(i,j) = y(i+1,j+1)-y(i+1,j);

        dx_nf(i,j) = x(i,j+1)-x(i+1,j+1);    % @ the North Face
        dy_nf(i,j) = y(i,j+1)-y(i+1,j+1);

        dx_wf(i,j) = x(i,j)-x(i,j+1);        % @ the West Face
        dy_wf(i,j) = y(i,j)-y(i,j+1);
```

```

dx_sf(i,j) = x(i+1,j)-x(i,j);           % @ the South Face
dy_sf(i,j) = y(i+1,j)-y(i,j);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Area for all the faces
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

area_ef(i,j) = sqrt(dx_ef(i,j)^2 + dy_ef(i,j)^2);
area_nf(i,j) = sqrt(dx_nf(i,j)^2 + dy_nf(i,j)^2);
area_wf(i,j) = sqrt(dx_wf(i,j)^2 + dy_wf(i,j)^2);
area_sf(i,j) = sqrt(dx_sf(i,j)^2 + dy_sf(i,j)^2);

A{i,j}=[area_ef(i,j);area_nf(i,j);area_wf(i,j);area_sf(i,j)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculating the unit normal vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

f = [0,0,1];
ne{i,j} = cross([dx_ef(i,j),dy_ef(i,j),0],f)...
/(area_ef(i,j));

nf{i,j} = cross([dx_nf(i,j),dy_nf(i,j),0],f)...
/(area_nf(i,j));

nw{i,j} = cross([dx_wf(i,j),dy_wf(i,j),0],f)...
/(area_wf(i,j));

ns{i,j} = cross([dx_sf(i,j),dy_sf(i,j),0],f) ...
/(area_sf(i,j));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculating the cell volumes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Vol(i,j) = 0.5*det([x(i+1,j+1)- x(i,j), y(i+1,j+1)- y(i,j);...
x(i,j+1)- x(i+1,j) , y(i,j+1)- y(i+1,j)]);

end
end

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Free stream conditions and constants
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

freestream           % Calling the freestream conditions
Vinf =[rho_ini;u_ini;v_ini;p_ini]; % Initialization V(infinity) vector
V{i,j} =[0;0;0;0];
U_c{i,j}=[0;0;0;0];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Primitive state vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j = 1:ny-1;
for i = 1:nx-1;

```



```

        rho_p(i,j) = Vinf(1);
        u_p(i,j)   = Vinf(2);
        v_p(i,j)   = Vinf(3);
        p_p(i,j)   = Vinf(4);

        V{i,j}     =[rho_p(i,j);u_p(i,j);v_p(i,j);p_p(i,j)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Adding in a point of perturbation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        if i > (nx/2) && j >= 1;

                V{i,j}(1) = .125 * V{i,j}(1);
                V{i,j}(4) = .1 * V{i,j}(4);
        end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Conservative state vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        rho_c(i,j)   = V{i,j}(1);
        rhou_c(i,j)  = V{i,j}(1)*V{i,j}(2);
        rhov_c(i,j)  = V{i,j}(1)*V{i,j}(3);
        rhoe_c(i,j)  = V{i,j}(1)*(cv*V{i,j}(4)/(V{i,j}(1)*R)+...
                    0.5*((V{i,j}(2))^2+(V{i,j}(3))^2));

        U_c{i,j}     =[rho_c(i,j);rhou_c(i,j);rhov_c(i,j);rhoe_c(i,j)];

        end
        end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Setting the fluxes to zero initially
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        F_e{i,j}=[0; 0; 0; 0];
        F_n{i,j}=[0; 0; 0; 0];
        F_w{i,j}=[0; 0; 0; 0];
        F_s{i,j}=[0; 0; 0; 0];
        F_plus_e{i,j}=[0; 0; 0; 0];
        F_plus_n{i,j}=[0; 0; 0; 0];
        F_minus_e{i,j}=[0; 0; 0; 0];
        F_minus_n{i,j}=[0; 0; 0; 0];
        F_ex{i,j}=[0; 0; 0; 0];
        F_ey{i,j}=[0; 0; 0; 0];
        F_nx{i,j}=[0; 0; 0; 0];
        F_ny{i,j}=[0; 0; 0; 0];
        F_wx{i,j}=[0; 0; 0; 0];
        F_wy{i,j}=[0; 0; 0; 0];
        F_sx{i,j}=[0; 0; 0; 0];
        F_sy{i,j}=[0; 0; 0; 0];
        Vinveb{i,j}= [0;0;0;0];
        Vinvnb{i,j}= [0;0;0;0];
        Vinvwb{i,j}= [0;0;0;0];
        Vinvsb{i,j}= [0;0;0;0];

```

```

Res{i,j}=[0; 0; 0; 0];

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Beginning of the time loop
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for n = 1:iter

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculating the contravariant velocities
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for j = 1:ny-1;
    for i = 1:nx-1;

        hob(i,j) = (cp*V{i,j}(4)/(V{i,j}(1)*R)) +
(1/2)*(V{i,j}(2)^2+V{i,j}(3)^2); % enthalpy

        a(i,j) = sqrt(gamma* V{i,j}(4)/ V{i,j}(1)); %speed of sound

        U_cone(i,j) = dot([ V{i,j}(2), V{i,j}(3),0],ne{i,j});
        U_conn(i,j) = dot([ V{i,j}(2), V{i,j}(3),0],nn{i,j});
        U_conw(i,j) = dot([ V{i,j}(2), V{i,j}(3),0],nw{i,j});
        U_cons(i,j) = dot([ V{i,j}(2), V{i,j}(3),0],ns{i,j});

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculating the contravariant mach numbers
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        M_cone(i,j) = U_cone(i,j)/a(i,j);
        M_conn(i,j) = U_conn(i,j)/a(i,j);
        M_conw(i,j) = U_conw(i,j)/a(i,j);
        M_cons(i,j) = U_cons(i,j)/a(i,j);

    end
end

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CFL step
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% for iter=1
dt_e = (area_ef(i,j)/(abs(U_cone(i,j))+a(i,j)))*CFL;
dt_n = (area_nf(i,j)/(abs(U_conn(i,j))+a(i,j)))*CFL;
dt_w = (area_wf(i,j)/(abs(U_conw(i,j))+a(i,j)))*CFL;
dt_s = (area_sf(i,j)/(abs(U_cons(i,j))+a(i,j)))*CFL;
dt_1 = min(dt_e,dt_n);
dt_2 = min(dt_w,dt_s);
dt_m1 = min(dt_1,dt_2);
if dt_m1 < dt
    dt=dt_m1;
end

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Calculating the supersonic inflow at the west boundary
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    i = 1;
    for j = 1:ny-1;

%       V_b_w{i,j}= Vinf;
%
%       ho_wb(i,j) = cp*V_b_w{i,j}(4)/(V_b_w{i,j}(1)*R) +
1/2*(V_b_w{i,j}(2)^2+V_b_w{i,j}(3)^2);
%
%       % F_w{i,j}=
[V_b_w{i,j}(1)*U_conw(i,j);(V_b_w{i,j}(1)*V_b_w{i,j}(2)*U_conw(i,j)) +
(V_b_w{i,j}(4)*nw{i,j}(1)); ...
%       % (V_b_w{i,j}(1)*U_conw(i,j)*V_b_w{i,j}(3)) +
(V_b_w{i,j}(4)*nw{i,j}(2));V_b_w{i,j}(1)*ho_wb(i,j)*U_conw(i,j)];

    Vinvwb{i,j}(1) = V{i,j}(1);

    Vinvwb{i,j}(2) = V{i,j}(2) - (U_conw(i,j)*nw{i,j}(1));

    Vinvwb{i,j}(3) = V{i,j}(3) - (U_conw(i,j)*nw{i,j}(2));

    Vinvwb{i,j}(4) = V{i,j}(4);

    hob_w(i,j) = cp*Vinvwb{i,j}(4)/(Vinvwb{i,j}(1)*R) +
0.5*(Vinvwb{i,j}(2)^2 + Vinvwb{i,j}(3)^2);

    F_wx{i,j} =
[Vinvwb{i,j}(1)*Vinvwb{i,j}(2);(Vinvwb{i,j}(1)*(Vinvwb{i,j}(2))^2)+(
Vinvwb{i,j}(4));...
    (Vinvwb{i,j}(1)*Vinvwb{i,j}(2)*
Vinvwb{i,j}(3));Vinvwb{i,j}(1)*hob_w(i,j)*Vinvwb{i,j}(2)];

    F_wy{i,j} =
[Vinvwb{i,j}(1)*Vinvwb{i,j}(3);(Vinvwb{i,j}(1)*Vinvwb{i,j}(2)*Vinvwb{i,j}(3))
;...
(Vinvwb{i,j}(1)*(Vinvwb{i,j}(3))^2)+(Vinvwb{i,j}(4));Vinvwb{i,j}(1)*hob_w(i,j)
)*Vinvwb{i,j}(3)];

%       F_n{i,j}=F_nx{i,j}*nn{i,j}(1)+F_ny{i,j}*nn{i,j}(2);

%
%       F_wx{i,j}=
[V_b_w{i,j}(1)*V_b_w{i,j}(2);(V_b_w{i,j}(1)*(V_b_w{i,j}(2))^2) +
(V_b_w{i,j}(4)); ...
%
(V_b_w{i,j}(1)*V_b_w{i,j}(2)*V_b_w{i,j}(3));V_b_w{i,j}(1)*ho_wb(i,j)*V_b_w{i,
j}(2)];
%

```

```

%           F_wy{i,j}=
[V_b_w{i,j}(1)*V_b_w{i,j}(3);(V_b_w{i,j}(1)*(V_b_w{i,j}(2))*V_b_w{i,j}(3));
...
%           (V_b_w{i,j}(1)*(V_b_w{i,j}(3))^2) +
(V_b_w{i,j}(4));V_b_w{i,j}(1)*ho_wb(i,j)*V_b_w{i,j}(3)];

           F_w{i,j}=F_wx{i,j}*nw{i,j}(1)+F_wy{i,j}*nw{i,j}(2);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Calculating the supersonic outflow at the east boundary
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
           i = nx-1;
           for j = 1:ny-1;

%           V_b_e{i,j} = V{i,j};
%
%           ho_eb(i,j) = cp*V_b_e{i,j}(4)/(V_b_e{i,j}(1)*R) +
1/2*(V_b_e{i,j}(2)^2+V_b_e{i,j}(3)^2);

           Vinveb{i,j}(1) = V{i,j}(1);

           Vinveb{i,j}(2) = V{i,j}(2) - (U_cone(i,j)*ne{i,j}(1));

           Vinveb{i,j}(3) = V{i,j}(3) - (U_cone(i,j)*ne{i,j}(2));

           Vinveb{i,j}(4) = V{i,j}(4);

           hob_e(i,j) = cp*Vinveb{i,j}(4)/(Vinveb{i,j}(1)*R) +
0.5*(Vinveb{i,j}(2)^2 + Vinveb{i,j}(3)^2);

           F_ex{i,j} =
[Vinveb{i,j}(1)*Vinveb{i,j}(2);(Vinveb{i,j}(1)*(Vinveb{i,j}(2))^2)+(Vinveb{i,
j}(4));...
           (Vinveb{i,j}(1)*Vinveb{i,j}(2)*
Vinveb{i,j}(3));Vinveb{i,j}(1)*hob_e(i,j)*Vinveb{i,j}(2)];

           F_ey{i,j} =
[Vinveb{i,j}(1)*Vinveb{i,j}(3);(Vinveb{i,j}(1)*Vinveb{i,j}(2)*Vinveb{i,j}(3))
;...
(Vinveb{i,j}(1)*(Vinveb{i,j}(3))^2)+(Vinveb{i,j}(4));Vinveb{i,j}(1)*hob_e(i,j)
)*Vinveb{i,j}(3)];

%
%           F_ex{i,j}=
[V_b_e{i,j}(1)*V_b_e{i,j}(2);(V_b_e{i,j}(1)*(V_b_e{i,j}(2))^2) +
(V_b_e{i,j}(4));...
%
(V_b_e{i,j}(1)*V_b_e{i,j}(2)*V_b_e{i,j}(3));V_b_e{i,j}(1)*ho_eb(i,j)*V_b_e{i,
j}(2)];

```

```

%
%      F_ey{i,j}=
[V_b_e{i,j}(1)*V_b_e{i,j}(3);(V_b_e{i,j}(1)*(V_b_e{i,j}(2))*V_b_e{i,j}(3));...
.
%
(V_b_e{i,j}(1)*(V_b_e{i,j}(3))^2)+(V_b_e{i,j}(4));V_b_e{i,j}(1)*ho_eb(i,j)*V_
b_e{i,j}(3)];
%
      F_e{i,j}=F_ex{i,j}*ne{i,j}(1)+F_ey{i,j}*ne{i,j}(2);

end

```

```

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Calculating for the inviscid wall at the north boundary
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

      j = ny-1;
      for i = 1:nx-1;

          Vinvnb{i,j}(1) = V{i,j}(1);

          Vinvnb{i,j}(2) = V{i,j}(2) - (U_conn(i,j)*nn{i,j}(1));

          Vinvnb{i,j}(3) = V{i,j}(3) - (U_conn(i,j)*nn{i,j}(2));

          Vinvnb{i,j}(4) = V{i,j}(4);

          hob_n(i,j) = cp*Vinvnb{i,j}(4)/(Vinvnb{i,j}(1)*R) +
          0.5*(Vinvnb{i,j}(2)^2 + Vinvnb{i,j}(3)^2);

          F_nx{i,j} =
          [Vinvnb{i,j}(1)*Vinvnb{i,j}(2);(Vinvnb{i,j}(1)*(Vinvnb{i,j}(2))^2)+(
          Vinvnb{i,j}(4));...
          (Vinvnb{i,j}(1)*Vinvnb{i,j}(2)*
          Vinvnb{i,j}(3));Vinvnb{i,j}(1)*hob_n(i,j)*Vinvnb{i,j}(2)];

          F_ny{i,j} =
          [Vinvnb{i,j}(1)*Vinvnb{i,j}(3);(Vinvnb{i,j}(1)*Vinvnb{i,j}(2)*Vinvnb{i,j}(3))
          ;...
          (Vinvnb{i,j}(1)*(Vinvnb{i,j}(3))^2)+(Vinvnb{i,j}(4));Vinvnb{i,j}(1)*hob_n(i,j)
          )*Vinvnb{i,j}(3)];

          F_n{i,j}=F_nx{i,j}*nn{i,j}(1)+F_ny{i,j}*nn{i,j}(2);

      end

```

```

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Calculating for the inviscid wall at the south boundary
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

        j = 1;
    for i = 1:nx-1;

        Vinvsb{i,j}(1) = V{i,j}(1);

        Vinvsb{i,j}(2) = V{i,j}(2) - (U_cons(i,j)*ns{i,j}(1));

        Vinvsb{i,j}(3) = V{i,j}(3) - (U_cons(i,j)*ns{i,j}(2));

        Vinvsb{i,j}(4) = V{i,j}(4);

        hob_s(i,j) = (cp* Vinvsb{i,j}(4))/(Vinvsb{i,j}(1)*R) +
0.5*(Vinvsb{i,j}(2)^2 + Vinvsb{i,j}(3)^2);

        F_sx{i,j} =
[Vinvsb{i,j}(1)*Vinvsb{i,j}(2);(Vinvsb{i,j}(1)*(Vinvsb{i,j}(2))^2)+(
Vinvsb{i,j}(4));...
        (Vinvsb{i,j}(1)*Vinvsb{i,j}(2)*
Vinvsb{i,j}(3));Vinvsb{i,j}(1)*hob_s(i,j)*Vinvsb{i,j}(2)];

        F_sy{i,j} =
[Vinvsb{i,j}(1)*Vinvsb{i,j}(3);(Vinvsb{i,j}(1)*Vinvsb{i,j}(2)*Vinvsb{i,j}(3))
;...
(Vinvsb{i,j}(1)*(Vinvsb{i,j}(3))^2)+(Vinvsb{i,j}(4));Vinvsb{i,j}(1)*hob_s(i,j)
)*Vinvsb{i,j}(3)];

        F_s{i,j}=F_sx{i,j}*ns{i,j}(1)+F_sy{i,j}*ns{i,j}(2);

    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Second-order Extrapolations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    i = 1;
for j = 1:ny-1;

    rho_p_rl(i,j) = (V{i+1,j}(1)-V{i,j}(1))/(V{i,j}(1)-Vinvwb{i,j}(1));

    rho_p_rr(i,j) = (V{i+2,j}(1)-V{i+1,j}(1))/(V{i+1,j}(1)-V{i,j}(1));

    u_p_rl(i,j) = (V{i+1,j}(2)-V{i,j}(2))/(V{i,j}(2)-Vinvwb{i,j}(2));

    u_p_rr(i,j) = (V{i+2,j}(2)-V{i+1,j}(2))/(V{i+1,j}(2)-V{i,j}(2));

    v_p_rl(i,j) = (V{i+1,j}(3)-V{i,j}(3))/(V{i,j}(3)-Vinvwb{i,j}(3));

```

```

v_p_rr(i,j) = (V{i+2,j}(3)-V{i+1,j}(3))/(V{i+1,j}(3)-V{i,j}(3));
p_p_rl(i,j) = (V{i+1,j}(4)-V{i,j}(4))/(V{i,j}(4)-Vinwvb{i,j}(4));
p_p_rr(i,j) = (V{i+2,j}(4)-V{i+1,j}(4))/(V{i+1,j}(4)-V{i,j}(4));

if rho_p_rl(i,j)<= 0
    rho_p_psi_l(i,j)= 0;
% elseif rho_p_rl(i,j)>= 2
% rho_p_psi_l(i,j)= 2;
else
    rho_p_psi_l(i,j) = (rho_p_rl(i,j)+
abs(rho_p_rl(i,j)))/(1+rho_p_rl(i,j));
end

if rho_p_rr(i,j)<= 0
    rho_p_psi_r(i,j)= 0;
% elseif rho_p_rr(i,j)>= 2
% rho_p_psi_r(i,j)= 2;
else
    rho_p_psi_r(i,j) = (rho_p_rr(i,j)+
abs(rho_p_rr(i,j)))/(1+rho_p_rr(i,j));
end

if u_p_rl(i,j)<= 0
    u_p_psi_l(i,j)= 0;
% elseif u_p_rl(i,j)>= 2
% u_p_psi_l(i,j)= 2;
else
    u_p_psi_l(i,j) = (u_p_rl(i,j)+ abs(u_p_rl(i,j)))/(1+u_p_rl(i,j));
end

if u_p_rr(i,j)<= 0
    u_p_psi_r(i,j)= 0;
% elseif u_p_rr(i,j)>= 2
% u_p_psi_r(i,j)= 2;
else
    u_p_psi_r(i,j) = (u_p_rr(i,j)+ abs(u_p_rr(i,j)))/(1+u_p_rr(i,j));
end

if v_p_rl(i,j)<= 0
    v_p_psi_l(i,j)= 0;
% elseif v_p_rl(i,j)>= 2
% v_p_psi_l(i,j)= 2;
else
    v_p_psi_l(i,j) = (v_p_rl(i,j)+ abs(v_p_rl(i,j)))/(1+v_p_rl(i,j));
end

if v_p_rr(i,j)<= 0
    v_p_psi_r(i,j)= 0;
% elseif v_p_rr(i,j)>= 2
% v_p_psi_r(i,j)= 2;
else
    v_p_psi_r(i,j) = (v_p_rr(i,j)+ abs(v_p_rr(i,j)))/(1+v_p_rr(i,j));
end

```

```

    if p_p_rl(i,j)<= 0
        p_p_psi_l(i,j)= 0;
    %     elseif p_p_rl(i,j)>= 2
    %         p_p_psi_l(i,j)= 2;
    else
        p_p_psi_l(i,j) = (p_p_rl(i,j)+ abs(p_p_rl(i,j)))/(1+p_p_rl(i,j));
    end

    if p_p_rr(i,j)<= 0
        p_p_psi_r(i,j)= 0;
    %     elseif p_p_rr(i,j)>= 2
    %         p_p_psi_r(i,j)= 2;
    else
        p_p_psi_r(i,j) = (p_p_rr(i,j)+ abs(p_p_rr(i,j)))/(1+p_p_rr(i,j));
    end

    d_rho_p_l(i,j)= ((phi/4)*((1-k)*(V{i,j}(1)-Vinvwb{i,j}(1))));

    if d_rho_p_l(i,j) <= (V{i,j}(1)-Vinvwb{i,j}(1)) && d_rho_p_l(i,j) <=
(V{i+1,j}(1)-V{i,j}(1))

        d_rho_p_l(i,j)= d_rho_p_l(i,j);
    else
        d_rho_p_l(i,j)= (rho_p_psi_l(i,j))*(d_rho_p_l(i,j));
    end

    d_u_p_l(i,j)  = ((phi/4)*((1-k)*(V{i,j}(2)-Vinvwb{i,j}(2))));

    if d_u_p_l(i,j) <= (V{i,j}(2)-Vinvwb{i,j}(2)) && d_u_p_l(i,j) <=
(V{i+1,j}(2)-V{i,j}(2))

        d_u_p_l(i,j)= d_u_p_l(i,j);
    else
        d_u_p_l(i,j)= (u_p_psi_l(i,j))*(d_u_p_l(i,j));
    end

    d_v_p_l(i,j)  = ((phi/4)*((1-k)*(V{i,j}(3)-Vinvwb{i,j}(3))));

    if d_v_p_l(i,j) <= (V{i,j}(3)-Vinvwb{i,j}(3)) && d_v_p_l(i,j) <=
(V{i+1,j}(3)-V{i,j}(3))

        d_v_p_l(i,j)= d_v_p_l(i,j);
    else
        d_v_p_l(i,j)= (v_p_psi_l(i,j))*(d_v_p_l(i,j));
    end

    d_p_p_l(i,j)  = ((phi/4)*((1-k)*(V{i,j}(4)-Vinvwb{i,j}(4))));

    if d_p_p_l(i,j) <= (V{i,j}(4)-Vinvwb{i,j}(4)) && d_p_p_l(i,j) <=
(V{i+1,j}(4)-V{i,j}(4))

        d_p_p_l(i,j)= d_p_p_l(i,j);
    else

```



```

    d_p_p_l(i,j)= (p_p_psi_l(i,j))*(d_p_p_l(i,j));
end

    d_rho_p_r(i,j)= ((phi/4)*((1-k)*(V{i+2,j}(1)-V{i+1,j}(1))));

    if d_rho_p_r(i,j) <= (V{i+1,j}(1)-V{i,j}(1)) && d_rho_p_r(i,j) <=
(V{i+2,j}(1)-V{i+1,j}(1))
        d_rho_p_r(i,j)= d_rho_p_r(i,j);
    else
        d_rho_p_r(i,j)= (rho_p_psi_r(i,j))*(d_rho_p_r(i,j));
    end

    d_u_p_r(i,j)= ((phi/4)*((1-k)*(V{i+2,j}(2)-V{i+1,j}(2))));

    if d_u_p_r(i,j) <= (V{i+1,j}(2)-V{i,j}(2)) && d_u_p_r(i,j)<=
(V{i+2,j}(2)-V{i+1,j}(2))
        d_u_p_r(i,j) = d_u_p_r(i,j);
    else
        d_u_p_r(i,j) = (u_p_psi_r(i,j))*(d_u_p_r(i,j));
    end

    d_v_p_r(i,j)= ((phi/4)*((1-k)*(V{i+2,j}(3)-V{i+1,j}(3))));

    if d_v_p_r(i,j) <= (V{i+1,j}(3)-V{i,j}(3)) && d_v_p_r(i,j)<=
(V{i+2,j}(3)-V{i+1,j}(3))
        d_v_p_r(i,j) = d_v_p_r(i,j);
    else
        d_v_p_r(i,j) = (v_p_psi_r(i,j))*(d_v_p_r(i,j));
    end

    d_p_p_r(i,j)= ((phi/4)*((1-k)*(V{i+2,j}(4)-V{i+1,j}(4))));

    if d_p_p_r(i,j) <= (V{i+1,j}(4)-V{i,j}(4)) && d_p_p_r(i,j)<=
(V{i+2,j}(4)-V{i+1,j}(4))
        d_p_p_r(i,j) = d_p_p_r(i,j);
    else
        d_p_p_r(i,j) = (p_p_psi_r(i,j))*(d_p_p_r(i,j));
    end

end

for i = 2:nx-3;
    for j = 1:ny-1;

        rho_p_rl(i,j) = (V{i+1,j}(1)-V{i,j}(1))/(V{i,j}(1)-V{i-1,j}(1));

        rho_p_rr(i,j) = (V{i+2,j}(1)-V{i+1,j}(1))/(V{i+1,j}(1)-V{i,j}(1));

        u_p_rl(i,j) = (V{i+1,j}(2)-V{i,j}(2))/(V{i,j}(2)-V{i-1,j}(2));

        u_p_rr(i,j) = (V{i+2,j}(2)-V{i+1,j}(2))/(V{i+1,j}(2)-V{i,j}(2));
    end
end

```

```

v_p_rl(i,j) = (V{i+1,j}(3)-V{i,j}(3))/(V{i,j}(3)-V{i-1,j}(3));
v_p_rr(i,j) = (V{i+2,j}(3)-V{i+1,j}(3))/(V{i+1,j}(3)-V{i,j}(3));
p_p_rl(i,j) = (V{i+1,j}(4)-V{i,j}(4))/(V{i,j}(4)-V{i-1,j}(4));
p_p_rr(i,j) = (V{i+2,j}(4)-V{i+1,j}(4))/(V{i+1,j}(4)-V{i,j}(4));

if rho_p_rl(i,j)<= 0
    rho_p_psi_l(i,j)= 0;
% elseif rho_p_rl(i,j)>= 2
%     rho_p_psi_l(i,j)= 2;
else
    rho_p_psi_l(i,j) = (rho_p_rl(i,j)+
abs(rho_p_rl(i,j)))/(1+rho_p_rl(i,j));
end

if rho_p_rr(i,j)<= 0
    rho_p_psi_r(i,j)= 0;
% elseif rho_p_rr(i,j)>= 2
%     rho_p_psi_r(i,j)= 2;
else
    rho_p_psi_r(i,j) = (rho_p_rr(i,j)+
abs(rho_p_rr(i,j)))/(1+rho_p_rr(i,j));
end

if u_p_rl(i,j)<= 0
    u_p_psi_l(i,j)= 0;
% elseif u_p_rl(i,j)>= 2
%     u_p_psi_l(i,j)= 2;
else
    u_p_psi_l(i,j) = (u_p_rl(i,j)+ abs(u_p_rl(i,j)))/(1+u_p_rl(i,j));
end

if u_p_rr(i,j)<= 0
    u_p_psi_r(i,j)= 0;
% elseif u_p_rr(i,j)>= 2
%     u_p_psi_r(i,j)= 2;
else
    u_p_psi_r(i,j) = (u_p_rr(i,j)+ abs(u_p_rr(i,j)))/(1+u_p_rr(i,j));
end

if v_p_rl(i,j)<= 0
    v_p_psi_l(i,j)= 0;
% elseif v_p_rl(i,j)>= 2
%     v_p_psi_l(i,j)= 2;
else
    v_p_psi_l(i,j) = (v_p_rl(i,j)+ abs(v_p_rl(i,j)))/(1+v_p_rl(i,j));
end

if v_p_rr(i,j)<= 0
    v_p_psi_r(i,j)= 0;
% elseif v_p_rr(i,j)>= 2

```

```

%         v_p_psi_r(i,j)= 2;
else
    v_p_psi_r(i,j) = (v_p_rr(i,j)+ abs(v_p_rr(i,j)))/(1+v_p_rr(i,j));
end

if p_p_rl(i,j)<= 0
    p_p_psi_l(i,j)= 0;
%     elseif p_p_rl(i,j)>= 2
%         p_p_psi_l(i,j)= 2;
else
    p_p_psi_l(i,j) = (p_p_rl(i,j)+ abs(p_p_rl(i,j)))/(1+p_p_rl(i,j));
end

if p_p_rr(i,j)<= 0
    p_p_psi_r(i,j)= 0;
%     elseif p_p_rr(i,j)>= 2
%         p_p_psi_r(i,j)= 2;
else
    p_p_psi_r(i,j) = (p_p_rr(i,j)+ abs(p_p_rr(i,j)))/(1+p_p_rr(i,j));
end

d_rho_p_l(i,j)= ((phi/4)*((1-k)*(V{i,j}(1)-V{i-1,j}(1))));

if d_rho_p_l(i,j) <= (V{i,j}(1)-V{i-1,j}(1)) && d_rho_p_l(i,j)<=
(V{i+1,j}(1)-V{i,j}(1))
    d_rho_p_l(i,j)= d_rho_p_l(i,j);
else
    d_rho_p_l(i,j)= (rho_p_psi_l(i,j))*(d_rho_p_l(i,j));
end

d_u_p_l(i,j)  = ((phi/4)*((1-k)*(V{i,j}(2)-V{i-1,j}(2))));

if d_u_p_l(i,j) <= (V{i,j}(2)-V{i-1,j}(2)) && d_u_p_l(i,j) <=
(V{i+1,j}(2)-V{i,j}(2))
    d_u_p_l(i,j)= d_u_p_l(i,j);
else
    d_u_p_l(i,j)= (u_p_psi_l(i,j))*(d_u_p_l(i,j));
end

d_v_p_l(i,j)  = ((phi/4)*((1-k)*(V{i,j}(3)-V{i-1,j}(3))));

if d_v_p_l(i,j) <= (V{i,j}(3)-V{i-1,j}(3)) && d_v_p_l(i,j) <=
(V{i+1,j}(3)-V{i,j}(3))
    d_v_p_l(i,j)= d_v_p_l(i,j);
else
    d_v_p_l(i,j)= (v_p_psi_l(i,j))*(d_v_p_l(i,j));
end

d_p_p_l(i,j)  = ((phi/4)*((1-k)*(V{i,j}(4)-V{i-1,j}(4))));

if d_p_p_l(i,j) <= (V{i,j}(4)-V{i-1,j}(4)) && d_p_p_l(i,j) <=
(V{i+1,j}(4)-V{i,j}(4))

```

```

    d_p_p_l(i,j)= d_p_p_l(i,j);
else
    d_p_p_l(i,j)= (p_p_psi_l(i,j))*(d_p_p_l(i,j));
end

d_rho_p_r(i,j)= ((phi/4)*((1-k)*(V{i+2,j}(1)-V{i+1,j}(1))));

if d_rho_p_r(i,j) <= (V{i+1,j}(1)-V{i,j}(1)) && d_rho_p_r(i,j)<=
(V{i+2,j}(1)-V{i+1,j}(1))
    d_rho_p_r(i,j)= d_rho_p_r(i,j);
else
    d_rho_p_r(i,j)= (rho_p_psi_r(i,j))*(d_rho_p_r(i,j));
end

d_u_p_r(i,j) = ((phi/4)*((1-k)*(V{i+2,j}(2)-V{i+1,j}(2))));

if d_u_p_r(i,j) <= (V{i+1,j}(2)-V{i,j}(2)) && d_u_p_r(i,j)<=
(V{i+2,j}(2)-V{i+1,j}(2))
    d_u_p_r(i,j)= d_u_p_r(i,j);
else
    d_u_p_r(i,j)= (u_p_psi_r(i,j))*(d_u_p_r(i,j));
end

d_v_p_r(i,j) = ((phi/4)*((1-k)*(V{i+2,j}(3)-V{i+1,j}(3))));

if d_v_p_r(i,j) <= (V{i+1,j}(3)-V{i,j}(3)) && d_v_p_r(i,j)<=
(V{i+2,j}(3)-V{i+1,j}(3))
    d_v_p_r(i,j)= d_v_p_r(i,j);
else
    d_v_p_r(i,j)= (v_p_psi_r(i,j))*(d_v_p_r(i,j));
end

d_p_p_r(i,j) = ((phi/4)*((1-k)*(V{i+2,j}(4)-V{i+1,j}(4))));

if d_p_p_r(i,j) <= (V{i+1,j}(4)-V{i,j}(4)) && d_p_p_r(i,j) <=
(V{i+2,j}(4)-V{i+1,j}(4))
    d_p_p_r(i,j)= d_p_p_r(i,j);
else
    d_p_p_r(i,j)= (p_p_psi_r(i,j))*(d_p_p_r(i,j));
end

end
end

i = nx-2;
for j = 1:ny-1;

rho_p_rl(i,j) = (V{i+1,j}(1)-V{i,j}(1))/(V{i,j}(1)-V{i-1,j}(1));

```

```

% rho_p_rr(i,j) = (Vinveb{i,j}(1)-V{i+1,j}(1))/(V{i+1,j}(1)-V{i,j}(1));
rho_p_rr(i,j) = 0;

u_p_rl(i,j) = (V{i+1,j}(2)-V{i,j}(2))/(V{i,j}(2)-V{i-1,j}(2));
u_p_rr(i,j) = -(U_cone(i+1,j)*ne{i+1,j}(1))/(V{i+1,j}(2)-V{i,j}(2));
v_p_rl(i,j) = (V{i+1,j}(3)-V{i,j}(3))/(V{i,j}(3)-V{i-1,j}(3));
v_p_rr(i,j) = -(U_cone(i+1,j)*ne{i+1,j}(2))/(V{i+1,j}(3)-V{i,j}(3));
p_p_rl(i,j) = (V{i+1,j}(4)-V{i,j}(4))/(V{i,j}(4)-V{i-1,j}(4));
% p_p_rr(i,j) = (Vinveb{i,j}(4)-V{i+1,j}(4))/(V{i+1,j}(4)-V{i,j}(4));
p_p_rr(i,j) = 0;

if rho_p_rl(i,j)<= 0
    rho_p_psi_l(i,j)= 0;
% elseif rho_p_rl(i,j)>= 2
%     rho_p_psi_l(i,j)= 2;
else
    rho_p_psi_l(i,j) = (rho_p_rl(i,j)+
abs(rho_p_rl(i,j)))/(1+rho_p_rl(i,j));
end

if rho_p_rr(i,j)<= 0
    rho_p_psi_r(i,j)= 0;
% elseif rho_p_rr(i,j)>= 2
%     rho_p_psi_r(i,j)= 2;
else
    rho_p_psi_r(i,j) = (rho_p_rr(i,j)+
abs(rho_p_rr(i,j)))/(1+rho_p_rr(i,j));
end

if u_p_rl(i,j)<= 0
    u_p_psi_l(i,j)= 0;
% elseif u_p_rl(i,j)>= 2
%     u_p_psi_l(i,j)= 2;
else
    u_p_psi_l(i,j) = (u_p_rl(i,j)+ abs(u_p_rl(i,j)))/(1+u_p_rl(i,j));
end

if u_p_rr(i,j)<= 0
    u_p_psi_r(i,j)= 0;
% elseif u_p_rr(i,j)>= 2
%     u_p_psi_r(i,j)= 2;
else
    u_p_psi_r(i,j) = (u_p_rr(i,j)+ abs(u_p_rr(i,j)))/(1+u_p_rr(i,j));
end

if v_p_rl(i,j)<= 0
    v_p_psi_l(i,j)= 0;

```

```

%     elseif v_p_rl(i,j)>= 2
%         v_p_psi_l(i,j)= 2;
else
    v_p_psi_l(i,j) = (v_p_rl(i,j)+ abs(v_p_rl(i,j)))/(1+v_p_rl(i,j));
end

if v_p_rr(i,j)<= 0
    v_p_psi_r(i,j)= 0;
%     elseif v_p_rr(i,j)>= 2
%         v_p_psi_r(i,j)= 2;
else
    v_p_psi_r(i,j) = (v_p_rr(i,j)+ abs(v_p_rr(i,j)))/(1+v_p_rr(i,j));
end

if p_p_rl(i,j)<= 0
    p_p_psi_l(i,j)= 0;
%     elseif p_p_rl(i,j)>= 2
%         p_p_psi_l(i,j)= 2;
else
    p_p_psi_l(i,j) = (p_p_rl(i,j)+ abs(p_p_rl(i,j)))/(1+p_p_rl(i,j));
end

if p_p_rr(i,j)<= 0
    p_p_psi_r(i,j)= 0;
%     elseif p_p_rr(i,j)>= 2
%         p_p_psi_r(i,j)= 2;
else
    p_p_psi_r(i,j) = (p_p_rr(i,j)+ abs(p_p_rr(i,j)))/(1+p_p_rr(i,j));
end

d_rho_p_l(i,j)= ((phi/4)*((1-k)*(V{i,j}(1)-V{i-1,j}(1))));

if d_rho_p_l(i,j) <= (V{i,j}(1)-V{i-1,j}(1)) && d_rho_p_l(i,j)<=
(V{i+1,j}(1)-V{i,j}(1))
    d_rho_p_l(i,j)= d_rho_p_l(i,j);
else
    d_rho_p_l(i,j)= (rho_p_psi_l(i,j))*(d_rho_p_l(i,j));
end

d_u_p_l(i,j) = ((phi/4)*((1-k)*(V{i,j}(2)-V{i-1,j}(2))));

if d_u_p_l(i,j) <= (V{i,j}(2)-V{i-1,j}(2)) && d_u_p_l(i,j)<=
(V{i+1,j}(2)-V{i,j}(2))
    d_u_p_l(i,j)= d_u_p_l(i,j);
else
    d_u_p_l(i,j)= (u_p_psi_l(i,j))*(d_u_p_l(i,j));
end

d_v_p_l(i,j) = ((phi/4)*((1-k)*(V{i,j}(3)-V{i-1,j}(3))));

if d_v_p_l(i,j) <= (V{i,j}(3)-V{i-1,j}(3)) && d_v_p_l(i,j)<=
(V{i+1,j}(3)-V{i,j}(3))
    d_v_p_l(i,j)= d_v_p_l(i,j);
else
    d_v_p_l(i,j)= (v_p_psi_l(i,j))*(d_v_p_l(i,j));
end

```

```

end

d_pp_l(i,j) = ((phi/4)*((1-k)*(V{i,j}(4)-V{i-1,j}(4))));

if d_pp_l(i,j) <= (V{i,j}(4)-V{i-1,j}(4)) && d_pp_l(i,j)<=
(V{i+1,j}(4)-V{i,j}(4))
    d_pp_l(i,j)= d_pp_l(i,j);
else
    d_pp_l(i,j)= (p_ppsi_l(i,j))*(d_pp_l(i,j));
end

%    d_rho_p_r(i,j)= ((phi/4)*((1-k)*(V{i+2,j}(2)-V{i+1,j}(1))));

d_rho_p_r(i,j)= 0;

if d_rho_p_r(i,j) <= (V{i+1,j}(1)-V{i,j}(1)) && d_rho_p_r(i,j)<= 0
    d_rho_p_r(i,j)= d_rho_p_r(i,j);
else
    d_rho_p_r(i,j)= (rho_ppsi_r(i,j))*(d_rho_p_r(i,j));
end

d_u_p_r(i,j) = ((phi/4)*((1-k)*(-(U_cone(i+1,j)*ne{i+1,j}(1)))));

if d_u_p_r(i,j) <= (V{i+1,j}(2)-V{i,j}(2)) && d_u_p_r(i,j)<= (-
(U_cone(i+1,j)*ne{i+1,j}(1)))
    d_u_p_r(i,j)= d_u_p_r(i,j);
else
    d_u_p_r(i,j)= (u_ppsi_r(i,j))*(d_u_p_r(i,j));
end

d_v_p_r(i,j) = ((phi/4)*((1-k)*(-(U_cone(i+1,j)*ne{i+1,j}(2)))));

if d_v_p_r(i,j) <= (V{i+1,j}(3)-V{i,j}(3)) && d_v_p_r(i,j)<= (-
(U_cone(i+1,j)*ne{i+1,j}(2)))
    d_v_p_r(i,j)= d_v_p_r(i,j);
else
    d_v_p_r(i,j)= (v_ppsi_r(i,j))*(d_v_p_r(i,j));
end

%    d_pp_r(i,j) = ((phi/4)*((1-k)*(V{i+2,j}(4)-V{i+1,j}(4))));

d_pp_r(i,j) = 0;

if d_pp_r(i,j) <= (V{i+1,j}(4)-V{i,j}(4)) && d_pp_r(i,j)<= 0
    d_pp_r(i,j)= d_pp_r(i,j);
else
    d_pp_r(i,j)= (p_ppsi_r(i,j))*(d_pp_r(i,j));
end

%    d_rho_p_r(i,j)= ((phi/4)*((1-k)*(Vinveb{i,j}(1)-V{i+1,j}(1))));

```

```

%      d_u_p_r(i,j) = ((phi/4)*((1-k)*(Vinveb{i,j}(2)-V{i+1,j}(2))));
%      d_v_p_r(i,j) = ((phi/4)*((1-k)*(Vinveb{i,j}(3)-V{i+1,j}(3))));
%      d_p_p_r(i,j) = ((phi/4)*((1-k)*(Vinveb{i,j}(4)-V{i+1,j}(4))));

%
%      d_rho_p_r(i,j)= 0;
%      d_u_p_r(i,j) = 0;
%      d_v_p_r(i,j) = 0;
%      d_p_p_r(i,j) = 0;

%
%      if d_rho_p_r(i,j) <= 2*(V{i+1,j}(1)-V{i,j}(1)) &&
d_rho_p_r(i,j) <= 2*(Vinveb{i,j}(1)-V{i+1,j}(1))
%      if d_u_p_r(i,j) <= 2*(V{i+1,j}(2)-V{i,j}(2)) && d_u_p_r(i,j)
<= 2*(Vinveb{i,j}(2)-V{i+1,j}(2))
%      if d_v_p_r(i,j) <= 2*(V{i+1,j}(3)-V{i,j}(3)) && d_v_p_r(i,j)
<= 2*(Vinveb{i,j}(3)-V{i+1,j}(3))
%      if d_p_p_r(i,j) <= 2*(V{i+1,j}(4)-V{i,j}(4)) && d_p_p_r(i,j)
<= 2*(Vinveb{i,j}(4)-V{i+1,j}(4))
%
%      d_rho_p_r(i,j)= d_rho_p_r(i,j);
%      d_u_p_r(i,j) = d_u_p_r(i,j);
%      d_v_p_r(i,j) = d_v_p_r(i,j);
%      d_p_p_r(i,j) = d_p_p_r(i,j);
%
%
%      else
%      d_rho_p_r(i,j)= (rho_p_psi_r(i,j))*(d_rho_p_r(i,j));
%      d_u_p_r(i,j) = (u_p_psi_r(i,j))*(d_u_p_r(i,j));
%      d_v_p_r(i,j) = (v_p_psi_r(i,j))*(d_v_p_r(i,j));
%      d_p_p_r(i,j) = (p_p_psi_r(i,j))*(d_p_p_r(i,j));
%
%      end
%      end
%      end
%      end

```

end

```

j = 1;
for i = 1: nx-1;

```

```

rho_p_rn(i,j) = (V{i,j+2}(1)-V{i,j+1}(1))/(V{i,j+1}(1)-V{i,j}(1));

```

```

u_p_rn(i,j) = (V{i,j+2}(2)-V{i,j+1}(2))/(V{i,j+1}(2)-V{i,j}(2));

```

```

v_p_rn(i,j) = (V{i,j+2}(3)-V{i,j+1}(3))/(V{i,j+1}(3)-V{i,j}(3));

```

```

p_p_rn(i,j) = (V{i,j+2}(4)-V{i,j+1}(4))/(V{i,j+1}(4)-V{i,j}(4));

```

```

rho_p_rs(i,j) = (V{i,j+1}(1)-V{i,j}(1))/(V{i,j}(1)-Vinvsb{i,j}(1));

```

```

u_p_rs(i,j) = (V{i,j+1}(2)-V{i,j}(2))/(V{i,j}(2)-Vinvsb{i,j}(2));

```

```

v_p_rs(i,j) = (V{i,j+1}(3)-V{i,j}(3))/(V{i,j}(3)-Vinvsb{i,j}(3));

```



```

p_p_rs(i,j) = (V{i,j+1}(4)-V{i,j}(4))/(V{i,j}(4)-Vinvsb{i,j}(4));

if rho_p_rn(i,j)<= 0
    rho_p_psi_n(i,j)= 0;
% elseif rho_p_rn(i,j)>= 2
%     rho_p_psi_n(i,j)= 2;
else
    rho_p_psi_n(i,j) = (rho_p_rn(i,j)+
abs(rho_p_rn(i,j)))/(1+rho_p_rn(i,j));
end

if rho_p_rs(i,j)<= 0
    rho_p_psi_s(i,j)= 0;
% elseif rho_p_rs(i,j)>= 2
%     rho_p_psi_s(i,j)= 2;
else
    rho_p_psi_s(i,j) = (rho_p_rs(i,j)+
abs(rho_p_rs(i,j)))/(1+rho_p_rs(i,j));
end

if u_p_rn(i,j)<= 0
    u_p_psi_n(i,j)= 0;
% elseif u_p_rn(i,j)>= 2
%     u_p_psi_n(i,j)= 2;
else
    u_p_psi_n(i,j) = (u_p_rn(i,j)+ abs(u_p_rn(i,j)))/(1+u_p_rn(i,j));
end

if u_p_rs(i,j)<= 0
    u_p_psi_s(i,j)= 0;
% elseif u_p_rs(i,j)>= 2
%     u_p_psi_s(i,j)= 2;
else
    u_p_psi_s(i,j) = (u_p_rs(i,j)+ abs(u_p_rs(i,j)))/(1+u_p_rs(i,j));
end

if v_p_rn(i,j)<= 0
    v_p_psi_n(i,j)= 0;
% elseif v_p_rn(i,j)>= 2
%     v_p_psi_n(i,j)= 2;
else
    v_p_psi_n(i,j) = (v_p_rn(i,j)+ abs(v_p_rn(i,j)))/(1+v_p_rn(i,j));
end

if v_p_rs(i,j)<= 0
    v_p_psi_s(i,j)= 0;
% elseif v_p_rs(i,j)>= 2
%     v_p_psi_s(i,j)= 2;
else
    v_p_psi_s(i,j) = (v_p_rs(i,j)+ abs(v_p_rs(i,j)))/(1+v_p_rs(i,j));
end

if p_p_rn(i,j)<= 0
    p_p_psi_n(i,j)= 0;

```

```

%     elseif p_p_rn(i,j)>= 2
%         p_p_psi_n(i,j)= 2;
else
    p_p_psi_n(i,j) = (p_p_rn(i,j)+ abs(p_p_rn(i,j)))/(1+p_p_rn(i,j));
end

if p_p_rs(i,j)<= 0
    p_p_psi_s(i,j)= 0;
%     elseif p_p_rs(i,j)>= 2
%         p_p_psi_s(i,j)= 2;
else
    p_p_psi_s(i,j) = (p_p_rs(i,j)+ abs(p_p_rs(i,j)))/(1+p_p_rs(i,j));
end

d_rho_p_n(i,j)= ((phi/4)*((1-k)*(V{i,j+2}{1}-V{i,j+1}{1})));

if d_rho_p_n(i,j) <= (V{i,j+1}{1}-V{i,j}{1}) && d_rho_p_n(i,j) <=
(V{i,j+2}{1}-V{i,j+1}{1})
    d_rho_p_n(i,j)= d_rho_p_n(i,j);
else
    d_rho_p_n(i,j)= (rho_p_psi_n(i,j))*(d_rho_p_n(i,j));
end

d_u_p_n(i,j) = ((phi/4)*((1-k)*(V{i,j+2}{2}-V{i,j+1}{2})));

if d_u_p_n(i,j) <= (V{i,j+1}{2}-V{i,j}{2}) && d_u_p_n(i,j) <=
(V{i,j+2}{2}-V{i,j+1}{2})
    d_u_p_n(i,j)= d_u_p_n(i,j);
else
    d_u_p_n(i,j)= (u_p_psi_n(i,j))*(d_u_p_n(i,j));
end

d_v_p_n(i,j) = ((phi/4)*((1-k)*(V{i,j+2}{3}-V{i,j+1}{3})));

if d_v_p_n(i,j) <= (V{i,j+1}{3}-V{i,j}{3}) && d_v_p_n(i,j) <=
(V{i,j+2}{3}-V{i,j+1}{3})
    d_v_p_n(i,j)= d_v_p_n(i,j);
else
    d_v_p_n(i,j)= (v_p_psi_n(i,j))*(d_v_p_n(i,j));
end

d_p_p_n(i,j) = ((phi/4)*((1-k)*(V{i,j+2}{4}-V{i,j+1}{4})));

if d_p_p_n(i,j) <= (V{i,j+1}{4}-V{i,j}{4}) && d_p_p_n(i,j) <=
(V{i,j+2}{4}-V{i,j+1}{4})
    d_p_p_n(i,j)= d_p_p_n(i,j);
else
    d_p_p_n(i,j)= (p_p_psi_n(i,j))*(d_p_p_n(i,j));
end

d_rho_p_s(i,j)= ((phi/4)*((1-k)*(V{i,j}{1}-Vinvsb{i,j}{1})));

```

```

    if d_rho_p_s(i,j) <= (V{i,j}(1)-Vinvsb{i,j}(1)) && d_rho_p_s(i,j)<=
(V{i,j+1}(1)-V{i,j}(1))
        d_rho_p_s(i,j)= d_rho_p_s(i,j);
    else
        d_rho_p_s(i,j)= (rho_p_psi_s(i,j))*(d_rho_p_s(i,j));
    end

    d_u_p_s(i,j) = ((phi/4)*((1-k)*(V{i,j}(2)-Vinvsb{i,j}(2))));

    if d_u_p_s(i,j) <= (V{i,j}(2)-Vinvsb{i,j}(2)) && d_u_p_s(i,j)<=
(V{i,j+1}(2)-V{i,j}(2))
        d_u_p_s(i,j) = d_u_p_s(i,j);
    else
        d_u_p_s(i,j) = (u_p_psi_s(i,j))*(d_u_p_s(i,j));
    end

    d_v_p_s(i,j) = ((phi/4)*((1-k)*(V{i,j}(3)-Vinvsb{i,j}(3))));

    if d_v_p_s(i,j) <= (V{i,j}(3)-Vinvsb{i,j}(3)) && d_v_p_s(i,j)<=
(V{i,j+1}(3)-V{i,j}(3))
        d_v_p_s(i,j) = d_v_p_s(i,j);
    else
        d_v_p_s(i,j) = (v_p_psi_s(i,j))*(d_v_p_s(i,j));
    end

    d_p_p_s(i,j) = ((phi/4)*((1-k)*(V{i,j}(4)-Vinvsb{i,j}(4))));

    if d_p_p_s(i,j) <= (V{i,j}(4)-Vinvsb{i,j}(4)) && d_p_p_s(i,j)<=
(V{i,j+1}(4)-V{i,j}(4))
        d_p_p_s(i,j) = d_p_p_s(i,j);
    else
        d_p_p_s(i,j) = (p_p_psi_s(i,j))*(d_p_p_s(i,j));
    end

end

for j = 2:ny-3;
    for i = 1:nx-1;

        rho_p_rn(i,j) = (V{i,j+2}(1)-V{i,j+1}(1))/(V{i,j+1}(1)-V{i,j}(1));

        u_p_rn(i,j) = (V{i,j+2}(2)-V{i,j+1}(2))/(V{i,j+1}(2)-V{i,j}(2));

        v_p_rn(i,j) = (V{i,j+2}(3)-V{i,j+1}(3))/(V{i,j+1}(3)-V{i,j}(3));

        p_p_rn(i,j) = (V{i,j+2}(4)-V{i,j+1}(4))/(V{i,j+1}(4)-V{i,j}(4));

        rho_p_rs(i,j) = (V{i,j+1}(1)-V{i,j}(1))/(V{i,j}(1)-V{i,j-1}(1));

        u_p_rs(i,j) = (V{i,j+1}(2)-V{i,j}(2))/(V{i,j}(2)-V{i,j-1}(2));

        v_p_rs(i,j) = (V{i,j+1}(3)-V{i,j}(3))/(V{i,j}(3)-V{i,j-1}(3));

```

```

p_p_rs(i,j) = (V{i,j+1}{4})-V{i,j}{4})/(V{i,j}{4})-V{i,j-1}{4});

if rho_p_rn(i,j)<= 0
    rho_p_psi_n(i,j)= 0;
%   elseif rho_p_rn(i,j)>= 2
%       rho_p_psi_n(i,j)= 2;
else
    rho_p_psi_n(i,j) = (rho_p_rn(i,j)+
abs(rho_p_rn(i,j)))/(1+rho_p_rn(i,j));
end

if rho_p_rs(i,j)<= 0
    rho_p_psi_s(i,j)= 0;
%   elseif rho_p_rs(i,j)>= 2
%       rho_p_psi_s(i,j)= 2;
else
    rho_p_psi_s(i,j) = (rho_p_rs(i,j)+
abs(rho_p_rs(i,j)))/(1+rho_p_rs(i,j));
end

if u_p_rn(i,j)<= 0
    u_p_psi_n(i,j)= 0;
%   elseif u_p_rn(i,j)>= 2
%       u_p_psi_n(i,j)= 2;
else
    u_p_psi_n(i,j) = (u_p_rn(i,j)+ abs(u_p_rn(i,j)))/(1+u_p_rn(i,j));
end

if u_p_rs(i,j)<= 0
    u_p_psi_s(i,j)= 0;
%   elseif u_p_rs(i,j)>= 2
%       u_p_psi_s(i,j)= 2;
else
    u_p_psi_s(i,j) = (u_p_rs(i,j)+ abs(u_p_rs(i,j)))/(1+u_p_rs(i,j));
end

if v_p_rn(i,j)<= 0
    v_p_psi_n(i,j)= 0;
%   elseif v_p_rn(i,j)>= 2
%       v_p_psi_n(i,j)= 2;
else
    v_p_psi_n(i,j) = (v_p_rn(i,j)+ abs(v_p_rn(i,j)))/(1+v_p_rn(i,j));
end

if v_p_rs(i,j)<= 0
    v_p_psi_s(i,j)= 0;
%   elseif v_p_rs(i,j)>= 2
%       v_p_psi_s(i,j)= 2;
else
    v_p_psi_s(i,j) = (v_p_rs(i,j)+ abs(v_p_rs(i,j)))/(1+v_p_rs(i,j));
end

if p_p_rn(i,j)<= 0

```

```

        p_p_psi_n(i,j)= 0;
%       elseif p_p_rn(i,j)>= 2
%           p_p_psi_n(i,j)= 2;
    else
        p_p_psi_n(i,j) = (p_p_rn(i,j)+ abs(p_p_rn(i,j)))/(1+p_p_rn(i,j));
    end

    if p_p_rs(i,j)<= 0
        p_p_psi_s(i,j)= 0;
%       elseif p_p_rs(i,j)>= 2
%           p_p_psi_s(i,j)= 2;
    else
        p_p_psi_s(i,j) = (p_p_rs(i,j)+ abs(p_p_rs(i,j)))/(1+p_p_rs(i,j));
    end

    d_rho_p_n(i,j)= ((phi/4)*((1-k)*(V{i,j+2}(1)-V{i,j+1}(1))));

    if d_rho_p_n(i,j) <= (V{i,j+1}(1)-V{i,j}(1)) && d_rho_p_n(i,j)<=
(V{i,j+2}(1)-V{i,j+1}(1))
        d_rho_p_n(i,j)= d_rho_p_n(i,j);
    else
        d_rho_p_n(i,j)= (rho_p_psi_n(i,j))*(d_rho_p_n(i,j));
    end

    d_u_p_n(i,j) = ((phi/4)*((1-k)*(V{i,j+2}(2)-V{i,j+1}(2))));

    if d_u_p_n(i,j) <= (V{i,j+1}(2)-V{i,j}(2)) && d_u_p_n(i,j) <=
(V{i,j+2}(2)-V{i,j+1}(2))
        d_u_p_n(i,j)= d_u_p_n(i,j);
    else
        d_u_p_n(i,j)= (u_p_psi_n(i,j))*(d_u_p_n(i,j));
    end

    d_v_p_n(i,j) = ((phi/4)*((1-k)*(V{i,j+2}(3)-V{i,j+1}(3))));

    if d_v_p_n(i,j) <= (V{i,j+1}(3)-V{i,j}(3)) && d_v_p_n(i,j)<=
(V{i,j+2}(3)-V{i,j+1}(3))
        d_v_p_n(i,j)= d_v_p_n(i,j);
    else
        d_v_p_n(i,j)= (v_p_psi_n(i,j))*(d_v_p_n(i,j));
    end

    d_p_p_n(i,j) = ((phi/4)*((1-k)*(V{i,j+2}(4)-V{i,j+1}(4))));

    if d_p_p_n(i,j) <= (V{i,j+1}(4)-V{i,j}(4)) && d_p_p_n(i,j)<=
(V{i,j+2}(4)-V{i,j+1}(4))
        d_p_p_n(i,j)= d_p_p_n(i,j);
    else
        d_p_p_n(i,j)= (p_p_psi_n(i,j))*(d_p_p_n(i,j));
    end

    d_rho_p_s(i,j)= ((phi/4)*((1-k)*(V{i,j}(1)-V{i,j-1}(1))));

```

```

    if d_rho_p_s(i,j) <= (V{i,j}(1)-V{i,j-1}(1)) && d_rho_p_s(i,j)<=
(V{i,j+1}(1)-V{i,j}(1))
        d_rho_p_s(i,j)= d_rho_p_s(i,j);
    else
        d_rho_p_s(i,j)= (rho_p_psi_s(i,j))*(d_rho_p_s(i,j));
    end

    d_u_p_s(i,j) = ((phi/4)*((1-k)*(V{i,j}(2)-V{i,j-1}(2))));

    if d_u_p_s(i,j) <= (V{i,j}(2)-V{i,j-1}(2)) && d_u_p_s(i,j)<=
(V{i,j+1}(2)-V{i,j}(2))
        d_u_p_s(i,j)= d_u_p_s(i,j);
    else
        d_u_p_s(i,j)= (u_p_psi_s(i,j))*(d_u_p_s(i,j));
    end

    d_v_p_s(i,j) = ((phi/4)*((1-k)*(V{i,j}(3)-V{i,j-1}(3))));

    if d_v_p_s(i,j) <= (V{i,j}(3)-V{i,j-1}(3)) && d_v_p_s(i,j)<=
(V{i,j+1}(3)-V{i,j}(3))
        d_v_p_s(i,j)= d_v_p_s(i,j);
    else
        d_v_p_s(i,j)= (v_p_psi_s(i,j))*(d_v_p_s(i,j));
    end

    d_p_p_s(i,j) = ((phi/4)*((1-k)*(V{i,j}(4)-V{i,j-1}(4))));

    if d_p_p_s(i,j) <= (V{i,j}(4)-V{i,j-1}(4)) && d_p_p_s(i,j)<=
(V{i,j+1}(4)-V{i,j}(4))
        d_p_p_s(i,j)= d_p_p_s(i,j);
    else
        d_p_p_s(i,j)= (p_p_psi_s(i,j))*(d_p_p_s(i,j));
    end

end
end

j = ny-2;
for i = 1:nx-1;

    rho_p_rs(i,j) = (V{i,j+1}(1)-V{i,j}(1))/(V{i,j}(1)-V{i,j-1}(1));
%    rho_p_rr(i,j) = (Vinveb{i,j}(1)-V{i+1,j}(1))/(V{i+1,j}(1)-V{i,j}(1));

    rho_p_rn(i,j) = 0;

    u_p_rs(i,j) = (V{i,j+1}(2)-V{i,j}(2))/(V{i,j}(2)-V{i,j-1}(2));

    u_p_rn(i,j) = -(U_conn(i,j+1)*nn{i,j+1}(1))/(V{i,j+1}(2)-V{i,j}(2));

    v_p_rs(i,j) = (V{i,j+1}(3)-V{i,j}(3))/(V{i,j}(3)-V{i,j-1}(3));

    v_p_rn(i,j) = -(U_conn(i,j+1)*nn{i,j+1}(2))/(V{i,j+1}(3)-V{i,j}(3));

```

```

p_p_rs(i,j) = (V{i,j+1}(4)-V{i,j}(4))/(V{i,j}(4)-V{i,j-1}(4));
%
p_p_rr(i,j) = (Vinveb{i,j}(4)-V{i+1,j}(4))/(V{i+1,j}(4)-V{i,j}(4));
p_p_rn(i,j) = 0;

if rho_p_rn(i,j)<= 0
    rho_p_psi_n(i,j)= 0;
%
% elseif rho_p_rn(i,j)>= 2
%     rho_p_psi_n(i,j)= 2;
else
    rho_p_psi_n(i,j) = (rho_p_rn(i,j)+
abs(rho_p_rn(i,j)))/(1+rho_p_rn(i,j));
end

if rho_p_rs(i,j)<= 0
    rho_p_psi_s(i,j)= 0;
%
% elseif rho_p_rs(i,j)>= 2
%     rho_p_psi_s(i,j)= 2;
else
    rho_p_psi_s(i,j) = (rho_p_rs(i,j)+
abs(rho_p_rs(i,j)))/(1+rho_p_rs(i,j));
end

if u_p_rn(i,j)<= 0
    u_p_psi_n(i,j)= 0;
%
% elseif u_p_rn(i,j)>= 2
%     u_p_psi_n(i,j)= 2;
else
    u_p_psi_n(i,j) = (u_p_rn(i,j)+ abs(u_p_rn(i,j)))/(1+u_p_rn(i,j));
end

if u_p_rs(i,j)<= 0
    u_p_psi_s(i,j)= 0;
%
% elseif u_p_rs(i,j)>= 2
%     u_p_psi_s(i,j)= 2;
else
    u_p_psi_s(i,j) = (u_p_rs(i,j)+ abs(u_p_rs(i,j)))/(1+u_p_rs(i,j));
end

if v_p_rn(i,j)<= 0
    v_p_psi_n(i,j)= 0;
%
% elseif v_p_rn(i,j)>= 2
%     v_p_psi_n(i,j)= 2;
else
    v_p_psi_n(i,j) = (v_p_rn(i,j)+ abs(v_p_rn(i,j)))/(1+v_p_rn(i,j));
end

if v_p_rs(i,j)<= 0
    v_p_psi_s(i,j)= 0;
%
% elseif v_p_rs(i,j)>= 2
%     v_p_psi_s(i,j)= 2;

```

```

else
    v_p_psi_s(i,j) = (v_p_rs(i,j)+ abs(v_p_rs(i,j)))/(1+v_p_rs(i,j));
end

if p_p_rn(i,j)<= 0
    p_p_psi_n(i,j)= 0;
% elseif p_p_rn(i,j)>= 2
%     p_p_psi_n(i,j)= 2;
else
    p_p_psi_n(i,j) = (p_p_rn(i,j)+ abs(p_p_rn(i,j)))/(1+p_p_rn(i,j));
end

if p_p_rs(i,j)<= 0
    p_p_psi_s(i,j)= 0;
% elseif p_p_rs(i,j)>= 2
%     p_p_psi_s(i,j)= 2;
else
    p_p_psi_s(i,j) = (p_p_rs(i,j)+ abs(p_p_rs(i,j)))/(1+p_p_rs(i,j));
end

% d_rho_p_n(i,j)= ((phi/4)*((1-k)*(Vinvnb{i,j}{1}-V{i,j+1}{1})));
% d_u_p_n(i,j) = ((phi/4)*((1-k)*(Vinvnb{i,j}{2}-V{i,j+1}{2})));
% d_v_p_n(i,j) = ((phi/4)*((1-k)*(Vinvnb{i,j}{3}-V{i,j+1}{3})));
% d_p_p_n(i,j) = ((phi/4)*((1-k)*(Vinvnb{i,j}{4}-V{i,j+1}{4})));

    d_rho_p_n(i,j)= 0;

    if d_rho_p_n(i,j) <= (V{i,j+1}{1}-V{i,j}{1}) && d_rho_p_n(i,j)
<= 0
        d_rho_p_n(i,j)= d_rho_p_n(i,j);
    else
        d_rho_p_n(i,j)= (rho_p_psi_n(i,j))*(d_rho_p_n(i,j));
    end

    d_u_p_n(i,j) = ((phi/4)*((1-k)*(-
(U_conn(i,j+1)*nn{i,j+1}{1})))));

    if d_u_p_n(i,j) <= (V{i,j+1}{2}-V{i,j}{2}) && d_u_p_n(i,j)<=
(-(U_conn(i,j+1)*nn{i,j+1}{1}))
        d_u_p_n(i,j) = d_u_p_n(i,j);
    else
        d_u_p_n(i,j) = (u_p_psi_n(i,j))*(d_u_p_n(i,j));
    end

    d_v_p_n(i,j) = ((phi/4)*((1-k)*(-
(U_conn(i,j+1)*nn{i,j+1}{2})))));

    if d_v_p_n(i,j) <= (V{i,j+1}{3}-V{i,j}{3}) && d_v_p_n(i,j)<=
(-(U_conn(i,j+1)*nn{i,j+1}{2}))
        d_v_p_n(i,j) = d_v_p_n(i,j);
    else
        d_v_p_n(i,j) = (v_p_psi_n(i,j))*(d_v_p_n(i,j));
    end
end

```



```

d_p_p_n(i,j) = 0;

if d_p_p_n(i,j) <= (V{i,j+1}(4)-V{i,j}(4)) && d_p_p_n(i,j) <= 0
    d_p_p_n(i,j)= d_p_p_n(i,j);
else
    d_p_p_n(i,j)= (p_p_psi_n(i,j))*(d_p_p_n(i,j));
end

d_rho_p_s(i,j)= ((phi/4)*((1-k)*(V{i,j}(1)-V{i,j-1}(1))));

if d_rho_p_s(i,j) <= (V{i,j}(1)-V{i,j-1}(1)) &&
d_rho_p_s(i,j)<= (V{i,j+1}(1)-V{i,j}(1))
    d_rho_p_s(i,j)= d_rho_p_s(i,j);
else
    d_rho_p_s(i,j)= (rho_p_psi_s(i,j))*(d_rho_p_s(i,j));
end

d_u_p_s(i,j) = ((phi/4)*((1-k)*(V{i,j}(2)-V{i,j-1}(2))));

if d_u_p_s(i,j) <= (V{i,j}(2)-V{i,j-1}(2)) && d_u_p_s(i,j)<=
(V{i,j+1}(2)-V{i,j}(2))
    d_u_p_s(i,j)= d_u_p_s(i,j);
else
    d_u_p_s(i,j)= (u_p_psi_s(i,j))*(d_u_p_s(i,j));
end

d_v_p_s(i,j) = ((phi/4)*((1-k)*(V{i,j}(3)-V{i,j-1}(3))));

if d_v_p_s(i,j) <= (V{i,j}(3)-V{i,j-1}(3)) && d_v_p_s(i,j) <=
(V{i,j+1}(3)-V{i,j}(3))
    d_v_p_s(i,j)= d_v_p_s(i,j);
else
    d_v_p_s(i,j)= (v_p_psi_s(i,j))*(d_v_p_s(i,j));
end

d_p_p_s(i,j) = ((phi/4)*((1-k)*(V{i,j}(4)-V{i,j-1}(4))));

if d_p_p_s(i,j) <= (V{i,j}(4)-V{i,j-1}(4)) && d_p_p_s(i,j) <=
(V{i,j+1}(4)-V{i,j}(4))
    d_p_p_s(i,j)= d_p_p_s(i,j);
else
    d_p_p_s(i,j)= (p_p_psi_s(i,j))*(d_p_p_s(i,j));
end

end

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Adding the second order extrapolation
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j=1:ny-1
for i=1:nx-2

rho_p_l(i,j)= V{i,j}(1)+ d_rho_p_l(i,j);
u_p_l(i,j) = V{i,j}(2)+ d_u_p_l(i,j);
v_p_l(i,j) = V{i,j}(3)+ d_v_p_l(i,j);

```

```

p_p_l(i,j) = V{i,j}(4)+ d_p_p_l(i,j);

rho_p_r(i,j)= V{i+1,j}(1)- d_rho_p_r(i,j);
u_p_r(i,j) = V{i+1,j}(2)- d_u_p_r(i,j);
v_p_r(i,j) = V{i+1,j}(3)- d_v_p_r(i,j);
p_p_r(i,j) = V{i+1,j}(4)- d_p_p_r(i,j);

V_l{i,j} = [rho_p_l(i,j);u_p_l(i,j);v_p_l(i,j);p_p_l(i,j)];
V_r{i,j} = [rho_p_r(i,j);u_p_r(i,j);v_p_r(i,j);p_p_r(i,j)];

hob_l(i,j) = (cp*V_l{i,j}(4)/(V_l{i,j}(1)*R)) +
(1/2)*(V_l{i,j}(2)^2+V_l{i,j}(3)^2);% enthalpy
hob_r(i,j) = (cp*V_r{i,j}(4)/(V_r{i,j}(1)*R)) +
(1/2)*(V_r{i,j}(2)^2+V_r{i,j}(3)^2);% enthalpy

a_l(i,j) = sqrt(gamma* abs(V_l{i,j}(4))/ abs(V_l{i,j}(1)));
%speed of sound
a_r(i,j) = sqrt(gamma* abs(V_r{i,j}(4))/ abs(V_r{i,j}(1)));
%speed of sound

end
end

for i=1:nx-1
for j=1:ny-2

rho_p_n(i,j)= V{i,j+1}(1)- d_rho_p_n(i,j);
u_p_n(i,j) = V{i,j+1}(2)- d_u_p_n(i,j);
v_p_n(i,j) = V{i,j+1}(3)- d_v_p_n(i,j);
p_p_n(i,j) = V{i,j+1}(4)- d_p_p_n(i,j);

rho_p_s(i,j)= V{i,j}(1)+ d_rho_p_s(i,j);
u_p_s(i,j) = V{i,j}(2)+ d_u_p_s(i,j);
v_p_s(i,j) = V{i,j}(3)+ d_v_p_s(i,j);
p_p_s(i,j) = V{i,j}(4)+ d_p_p_s(i,j);

V_n{i,j} = [rho_p_n(i,j);u_p_n(i,j);v_p_n(i,j);p_p_n(i,j)];
V_s{i,j} = [rho_p_s(i,j);u_p_s(i,j);v_p_s(i,j);p_p_s(i,j)];

hob_n(i,j) = (cp*V_n{i,j}(4)/(V_n{i,j}(1)*R)) +
(1/2)*(V_n{i,j}(2)^2+V_n{i,j}(3)^2); % enthalpy
hob_s(i,j) = (cp*V_s{i,j}(4)/(V_s{i,j}(1)*R)) +
(1/2)*(V_s{i,j}(2)^2+V_s{i,j}(3)^2); % enthalpy

a_n(i,j) = sqrt(gamma* abs(V_n{i,j}(4))/ abs(V_n{i,j}(1)));
%speed of sound
a_s(i,j) = sqrt(gamma* abs(V_s{i,j}(4))/ abs(V_s{i,j}(1)));
%speed of sound

end
end

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculating the contravariant velocities for all r,l,n,s states
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j = 1:ny-1;
for i = 1:nx-2;

    U_cone_l(i,j) = dot([ V_l{i,j}(2), V_l{i,j}(3),0],ne{i,j});
    U_cone_r(i,j) = dot([ V_r{i,j}(2), V_r{i,j}(3),0],ne{i,j});

    if i == 1
        U_conw_l(i,j)=0;
    else
        U_conw_l(i,j) = dot([ V_l{i,j}(2), V_l{i,j}(3),0],nw{i,j});
    end

    U_conw_r(i,j) = dot([ V_r{i,j}(2), V_r{i,j}(3),0],nw{i,j});

    M_cone_l(i,j) = U_cone_l(i,j)/a_l(i,j);
    M_cone_r(i,j) = U_cone_r(i,j)/a_r(i,j);

end
end

for i=1:nx-1
for j=1:ny-2
    U_conn_n(i,j) = dot([ V_n{i,j}(2), V_n{i,j}(3),0],nn{i,j});
    U_conn_s(i,j) = dot([ V_s{i,j}(2), V_s{i,j}(3),0],nn{i,j});

    U_cons_n(i,j) = dot([ V_n{i,j}(2), V_n{i,j}(3),0],ns{i,j});

    if j==1
        U_cons_s(i,j)=0;
    else
        U_cons_s(i,j) = dot([ V_s{i,j}(2), V_s{i,j}(3),0],ns{i,j});
    end

    M_conn_n(i,j) = U_conn_n(i,j)/a_n(i,j);
    M_conn_s(i,j) = U_conn_s(i,j)/a_s(i,j);

end
end

```

```

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculating the contravariant mach numbers for all r,l,n,s states
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%      M_cone_l(i,j) = U_cone_l(i,j)/a_l(i,j);
%      M_cone_r(i,j) = U_cone_r(i,j)/a_r(i,j);

%
%      M_conn_n(i,j) = U_conn_n(i,j)/a_n(i,j);
%      M_conn_s(i,j) = U_conn_s(i,j)/a_s(i,j);
%
%      if i==1
%          M_conw_l(i,j)=0;
%      else
%          M_conw_l(i,j) = U_conw_l(i,j)/a_l(i,j);
%      end
%
%      M_conw_r(i,j) = U_conw_r(i,j)/a_r(i,j);
%
%      M_cons_n(i,j) = U_cons_n(i,j)/a_n(i,j);
%
%      if j==1
%          M_cons_s(i,j)=0;
%      else
%          M_cons_s(i,j) = U_cons_s(i,j)/a_s(i,j);
%      end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculating Time step for stability purposes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%for iter=1
  for i=1:nx-2
    for j=1:ny-1

      dt_e1 = (area_ef(i,j)/(abs(U_cone_l(i,j))+a_l(i,j)))*CFL;
      dt_w1 = (area_wf(i,j)/(abs(U_conw_r(i,j))+a_r(i,j)))*CFL;
      dt_a1 = min(dt_e1,dt_w1);
    end
  end
  for i=1:nx-1
    for j=1:ny-2
      dt_n1 = (area_nf(i,j)/(abs(U_conn_s(i,j))+a_s(i,j)))*CFL;
      dt_s1 = (area_sf(i,j)/(abs(U_cons_n(i,j))+a_n(i,j)))*CFL;

      dt_a2 = min(dt_n1,dt_s1);
    end
  end

  dt_m2 = min (dt_a1,dt_a2);
%end
  if dt_m2 < dt
    dt=dt_m2;

```

end

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
% %Calculating the flux vectors for subsonic and supersonic conditions
```

```
% %Calculating the Van Leer split fluxes for 2-D Euler  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
for j = 1:ny-1;  
for i = 1:nx-2;
```

```
%           F_plus_e{i,j} =  
((V{i,j}(1)*a(i,j))/4)*((M_cone(i,j)+1)^2)*[1;...  
%           V{i,j}(2)+((ne{i,j}(1))*(-  
U_cone(i,j)+2*a(i,j)))/gamma];...  
%           V{i,j}(3)+((ne{i,j}(2))*(-  
U_cone(i,j)+2*a(i,j)))/gamma];...  
%           hob(i,j)-a(i,j)^2*(M_cone(i,j)-  
1)^2/(gamma+1)];
```

```
%           F_minus_e{i,j} =((-V{i+1,j}(1)*a(i+1,j))/4)*((M_cone(i+1,j)-  
1)^2)*[1;...  
%           V{i+1,j}(2)+((ne{i,j}(1))*(-U_cone(i+1,j)-  
2*a(i+1,j)))/gamma];...  
%           V{i+1,j}(3)+((ne{i,j}(2))*(-U_cone(i+1,j)-  
2*a(i+1,j)))/gamma];...  
%           hob(i+1,j)-  
a(i+1,j)^2*(M_cone(i+1,j)+1)^2/(gamma+1)];
```

```
%           F_plus_e_l{i,j} =  
((V_l{i,j}(1)*a_l(i,j))/4)*((M_cone_l(i,j)+1)^2)*[1;...  
%           V_l{i,j}(2)+((ne_l{i,j}(1))*(-  
U_cone_l(i,j)+2*a_l(i,j)))/gamma];...  
%           V_l{i,j}(3)+((ne_l{i,j}(2))*(-  
U_cone_l(i,j)+2*a_l(i,j)))/gamma];...  
%           hob_l(i,j)-a_l(i,j)^2*(M_cone_l(i,j)-  
1)^2/(gamma+1)];
```

```
%           F_minus_e_r{i,j} =((-V_r{i,j}(1)*a_r(i,j))/4)*((M_cone_r(i,j)-  
1)^2)*[1;...  
%           V_r{i,j}(2)+((ne_r{i,j}(1))*(-U_cone_r(i,j)-  
2*a_r(i,j)))/gamma];...  
%           V_r{i,j}(3)+((ne_r{i,j}(2))*(-U_cone_r(i,j)-  
2*a_r(i,j)))/gamma];...  
%           hob_r(i,j)-  
a_r(i,j)^2*(M_cone_r(i,j)+1)^2/(gamma+1)];
```

```
%
```

```
if M_cone_l(i,j)<1
```

```

F_e{i,j}=F_plus_e{i,j}+F_minus_e{i,j};

elseif M_cone_l(i,j)>= 1

    F_e{i,j}=[V_l{i,j}(1)*U_cone_l(i,j) ; ...
              V_l{i,j}(1)*U_cone_l(i,j)*V_l{i,j}(2) +
V_l{i,j}(4)*ne{i,j}(1) ; ...
              V_l{i,j}(1)*U_cone_l(i,j)*V_l{i,j}(3) +
V_l{i,j}(4)*ne{i,j}(2) ; ...
              V_l{i,j}(1)*U_cone_l(i,j)*hob_l(i,j)];

    end

F_w{i+1,j} = -F_e{i,j};

end
end

for j= 1:ny-2;
for i= 1:nx-1;

% F_plus_n{i,j} = (V{i,j}(1)*a(i,j))/4*(M_conn(i,j)+1)^2*[1;...
% V{i,j}(2)+nn{i,j}(1)*(-U_conn(i,j)+2*a(i,j))/gamma;...
% V{i,j}(3)+nn{i,j}(2)*(-U_conn(i,j)+2*a(i,j))/gamma;...
% hob(i,j)-a(i,j)^2*(M_conn(i,j)-1)^2/(gamma+1)];
%
% F_minus_n{i,j} = (-V{i,j+1}(1)*a(i,j+1))/4*(M_conn(i,j+1)-1)^2*[1;...
% V{i,j+1}(2)+(nn{i,j}(1))*(-U_conn(i,j+1)-
2*a(i,j+1))/gamma;...
% V{i,j+1}(3)+(nn{i,j}(2))*(-U_conn(i,j+1)-
2*a(i,j+1))/gamma;...
% hob(i,j+1)-a(i,j+1)^2*(M_conn(i,j+1)+1)^2/(gamma+1)];

F_plus_n{i,j} = (V_s{i,j}(1)*a_s(i,j))/4*(M_conn_s(i,j)+1)^2*[1;...
V_s{i,j}(2)+nn{i,j}(1)*(-U_conn_s(i,j)+2*a_s(i,j))/gamma;...
V_s{i,j}(3)+nn{i,j}(2)*(-U_conn_s(i,j)+2*a_s(i,j))/gamma;...
hob_s(i,j)-a_s(i,j)^2*(M_conn_s(i,j)-1)^2/(gamma+1)];

F_minus_n{i,j} = (-V_n{i,j}(1)*a_n(i,j))/4*(M_conn_n(i,j)-1)^2*[1;...
V_n{i,j}(2)+(nn{i,j}(1))*(-U_conn_n(i,j)-2*a_n(i,j))/gamma;...
V_n{i,j}(3)+(nn{i,j}(2))*(-U_conn_n(i,j)-2*a_n(i,j))/gamma;...
hob_n(i,j)-a_n(i,j)^2*(M_conn_n(i,j)+1)^2/(gamma+1)];

%
%

if M_conn_s(i,j)<1

F_n{i,j}=F_plus_n{i,j}+F_minus_n{i,j};

```

```

elseif M_conn_s(i,j)>= 1

    F_n{i,j}=[V_s{i,j}(1)*U_conn_s(i,j) ; ...
             V_s{i,j}(1)*U_conn_s(i,j)*V_s{i,j}(2) +
V_s{i,j}(4)*nn{i,j}(1) ; ...
             V_s{i,j}(1)*U_conn_s(i,j)*V_s{i,j}(3) +
V_s{i,j}(4)*nn{i,j}(2) ; ...
             V_s{i,j}(1)*U_conn_s(i,j)*hob_s(i,j)];

    end

    F_s{i,j+1} = -F_n{i,j};

end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %Calculating the flux vectors for subsonic and supersonic conditions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j = 1:ny-1;
    for i = 1:nx-1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %Calculating the residuals for each of the faces
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    Res{i,j} =
[(F_e{i,j}(1)*A{i,j}(1)+F_n{i,j}(1)*A{i,j}(2)+F_w{i,j}(1)*A{i,j}(3)+F_s{i,j}(
1)*A{i,j}(4));...

(F_e{i,j}(2)*A{i,j}(1)+F_n{i,j}(2)*A{i,j}(2)+F_w{i,j}(2)*A{i,j}(3)+F_s{i,j}(2
)*A{i,j}(4));...

(F_e{i,j}(3)*A{i,j}(1)+F_n{i,j}(3)*A{i,j}(2)+F_w{i,j}(3)*A{i,j}(3)+F_s{i,j}(3
)*A{i,j}(4));...

(F_e{i,j}(4)*A{i,j}(1)+F_n{i,j}(4)*A{i,j}(2)+F_w{i,j}(4)*A{i,j}(3)+F_s{i,j}(4
)*A{i,j}(4))];

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %Updating the state vectors
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    U_c{i,j} = U_c{i,j} - (dt/Vol(i,j))*(Res{i,j}); % Euler Step

    V{i,j} = [U_c{i,j}(1);U_c{i,j}(2)/U_c{i,j}(1);...
             U_c{i,j}(3)/U_c{i,j}(1);(U_c{i,j}(4) - 0.5*U_c{i,j}(1)*...
             ((U_c{i,j}(2)/U_c{i,j}(1))^2 + (U_c{i,j}(3)/U_c{i,j}(1))^2 )
)*R/cv];

    a(i,j) = sqrt(gamma* V{i,j}(4)/ V{i,j}(1)); %speed of sound

    P_out(i,j) = V{i,j}(4);

```

```

M_out(i,j) = sqrt(V{i,j}(2)^2+V{i,j}(3)^2)/a(i,j);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Calculating the residuals for the L2 norms
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    R1(i,j) = Res{i,j}(1);
    R2(i,j) = Res{i,j}(2);
    R3(i,j) = Res{i,j}(3);
    R4(i,j) = Res{i,j}(4);

fprintf('%g : %g : %g : %g : %g\n',n,R1,R2,R3,R4);

fprintf('%g\n',R4);

    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Calculating L2 norms
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    L1(n) = sqrt(sum(sum(R1.^2)))/((nx-1)*(ny-1));
    L2(n) = sqrt(sum(sum(R2.^2)))/((nx-1)*(ny-1));
    L3(n) = sqrt(sum(sum(R3.^2)))/((nx-1)*(ny-1));
    L4(n) = sqrt(sum(sum(R4.^2)))/((nx-1)*(ny-1));

fprintf('%g : %g : %g : %g : %g\n',n,L1(n),L2(n),L3(n),L4(n));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Plotting the L2 norms
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(3)
subplot(2,2,1)
plot((L1)/max(L1))
title('L1')

subplot(2,2,2)
plot((L2)/max(L2))
title('L2')

subplot(2,2,3)
plot((L3)/max(L3))
title('L3')

subplot(2,2,4)
plot((L4)/max(L4))
title('L4')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Plotting the Pressure/Mach Contours
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if output == 1

```



```

figure(2);

contourf(x(1:end-1,1:end-1,:),'y(1:end-1,1:end-1,:)','transpose(P_out));

title('Pressure Dissipation from the point of perturbation')

else

figure (4);

contourf(x(1:end-1,1:end-1,:),'y(1:end-1,1:end-1,:)','transpose(M_out));

title('Mach Dissipation from the point of perturbation')

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if n==1 || rem(n,q)==0

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for j =1:ny-1;
for i = 1:nx-1;

rho_disp(i,j) = V{i,j}(1);
u_disp(i,j) = V{i,j}(2);
v_disp(i,j) = V{i,j}(3);
p_disp(i,j) = V{i,j}(4);
mf(i,j) = V{i,j}(1)* V{i,j}(2);
en(i,j) = (V{i,j}(4)/(V{i,j}(1))^(gamma));
enr(i,j) = en(i,j)/R;
M_disp(i,j) = sqrt(V{i,j}(2)^2+V{i,j}(3)^2)/a(i,j);
nt(i,j) = i;
nxy(i,j) = nt(i,j)/45;
end
end

rho_disp
u_disp
v_disp
p_disp

fid_1 = fopen(['rho_disp', num2str(floor(n/r), '%.4i') , '.dat'], 'w');
fid_2 = fopen(['u_disp', num2str(floor(n/r), '%.4i'), '.dat'], 'w');
fid_3 = fopen(['v_disp', num2str(floor(n/r), '%.4i'), '.dat'], 'w');
fid_4 = fopen(['p_disp', num2str(floor(n/r), '%.4i'), '.dat'], 'w');
fid_5 = fopen(['mf', num2str(floor(n/r), '%.4i'), '.dat'], 'w');
fid_6 = fopen(['enr', num2str(floor(n/r), '%.4i'), '.dat'], 'w');

```

```

fid_7 = fopen(['nxy',num2str(floor(n/r),'%.4i'),'.dat'],'w');
fid_a = fopen(['M_disp',num2str(floor(n/r),'%.4i'),'.dat'],'w');

for i = 1:nx-1
    for j = 1:ny-1

        fprintf(fid_1,'%f ',rho_disp(i,j));
        fprintf(fid_2,'%f ',u_disp(i,j));
        fprintf(fid_3,'%f ',v_disp(i,j));
        fprintf(fid_4,'%f ',p_disp(i,j));
        fprintf(fid_5,'%f ',mf(i,j));
        fprintf(fid_6,'%f ',enr(i,j));
        fprintf(fid_7,'%f ',nxy(i,j));
        fprintf(fid_a,'%f ',M_disp(i,j));

        if j == ny-1
            fprintf(fid_1, '\n');
            fprintf(fid_2, '\n');
            fprintf(fid_3, '\n');
            fprintf(fid_4, '\n');
            fprintf(fid_5, '\n');
            fprintf(fid_6, '\n');
            fprintf(fid_7, '\n');
            fprintf(fid_a, '\n');
        end

    end

end

fclose(fid_1);
fclose(fid_2);
fclose(fid_3);
fclose(fid_4);
fclose(fid_5);
fclose(fid_6);
fclose(fid_7);
fclose(fid_a);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Sod Analytical Results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x1=10*[0.000000;0.263357;0.281906; 0.300455; 0.319004; 0.337553; 0.356102;
0.374651;0.393200;...
0.411749; 0.430298; 0.448847; 0.467396; 0.485945; 0.685490; 0.685490
;0.850431; 0.850431; 1.000000];

p_th=10^5*[1.000000;1.000000;0.912059;0.830832;0.755886;0.686811;0.623219;0.5
64742;0.511032;0.461762;...

0.416621;0.375316;0.337572;0.303130;0.303130;0.303130;0.303130;0.100000;0.100
000];

```

```

enr_th=[348.432055749129;348.432055749129;348.432329923741;348.431920848723;3
48.431982355768;348.431902726631;...

348.431981057585;348.432349291094;348.431787748745;348.431937741368;348.43238
0688576;348.431992707131;...

348.431595830625;348.432341021727;348.432341021727;675.907469652912;675.90746
9652912;640.389326827615;...
    640.389326827615];

rho_th=[1.000000;1.000000;0.936364;0.876011;0.818810;0.764637;0.713370;0.6648
90;0.619083;0.575836;0.535040;...

0.496590;0.460383;0.426319;0.426319;0.265574;0.265574;0.125000;0.125000];

m_th=[0.000000;0.000000;0.066185;0.134145;0.203953;0.275686;0.349425;0.425253
;0.503263;0.583548;0.666211;...

0.751358;0.839103;0.929567;0.929567;0.733678;0.733678;0.000000;0.000000];

u_th=[0;0;24.4406472842199;48.8810285254502;73.3213913489630;97.7619141786265
;122.202731499605;...

146.643071015574;171.083697855818;195.52409830508;219.964841359251;244.405225
659930;268.845555208870;...

293.286336701323;293.286336701323;293.285908350364;293.285908350364;0;0];

mf_th=[0;0;22.8853422536412;42.8203186796081;60.0362884504444;74.752376771802
5;87.1757625698729;...

97.5015114875451;105.915008919673;112.589814671608;117.689988720854;121.36919
1010465;...

123.771923243725;125.033537776171;125.033537776171;77.8891118242395;77.889111
8242395;0;0];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plotting the results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(3)
subplot(2,3,1)
plot(nxy,(p_disp),'-ro',x1,p_th);

    q=legend('Experimental',1);
    set(q,'Interpreter','none');
% hold on
% plot(x1,p_th,'-.b');

grid on

```

```

axis square

% set(h,'Interpreter','none')
title('Pressure')

subplot(2,3,2)
plot(nxy,(enr), '-ro')
% q=legend('Experimental',1);
% set(q,'Interpreter','none');
grid on
hold on
plot(x1,enr_th)
% h=legend('Analytical',1);
% set(h,'Interpreter','none');
axis square
title('Entropy/R')

subplot(2,3,3)
plot(nxy,(u_disp), '-ro')
grid on
hold on
plot(x1,u_th)
axis square
title('Velocity')

subplot(2,3,4)
plot(nxy,(M_disp), '-ro')
grid on
hold on
plot(x1,m_th)
axis square
title('Mach Number')

subplot(2,3,5)
plot(nxy,(rho_disp), '-ro')
grid on
hold on
plot(x1,rho_th)
axis square
title('Density')

subplot(2,3,6)
plot(nxy,(mf), '-ro')
grid on
hold on
plot(x1,mf_th)
axis square
title('Mass Flow')

end
end

```

Gridloader:

```
function [x,y,z,ni,nj,nk] = grid_loader(filename)
clc;

addpath('/P:/ae699/arj code/arj_sod_sec/')
%addpath('/home/students/vijayana/Desktop/arj code/arj_sod/')

% addpath('/C:/Users/BharadwajPopuri/Desktop/arj code/')
filename = 'sod3.grd';
% Reading the output (.grd) file
I = importdata(filename);
zones = I(1);
ni = I(2);
nj = I(3);
nk = I(4);
tot = ni*nj*nk;

X = I(5:4+tot);
Y = I(5+tot:4+tot*2);
Z = I(5+tot*2:end);
% Obtaining the co-ordinates
n = 1;
for j = 1:nj
    for i = 1:ni
        x(i,j) = X(n);
        y(i,j) = Y(n);
        z(i,j) = Z(n);

        n = n + 1;
    end
end
end
```