AE 6042 Computational Fluid Dynamics: Final Project

2-D Finite Volume Euler Solver for supersonic flow over airfoil

Aishwerya Singh Gahlot

Instructor: Dr. Joseph Oefelein

Date: April 30, 2024

Contents

1 Introduction

Computational Fluid Dynamics is a fundamental topic in engineering, allowing us to simulate real-world scenarios computationally wit the help of numerical methods. This final project aims to implement a supersonic flow on a diamond-shaped airfoil surface using a 2D Euler solver. The 2D solver is developed using a Finite volume formulation with an Advection Upstream Splitting Method (AUSM^+ - up) FVS Scheme. The solver is developed in two stages, the first stage is a simple first-order scheme and later on, it is modified to a higher order, Total Variation Diminishing (TVD) Monotone Upstream Scheme for Conservation Laws (MUSCL) formulation. Since the shape of the diamond airfoil is like a wedge, the supersonic flow introduced from the inlet will change its direction causing an Oblique Shock wave. The solver aims to capture the oblique shock wave. The wave angle, Mach number, Pressure, and Temperature ratio across the first oblique shock that forms at the leading edge of the airfoil will also be compared to verification data from the given analytical solution data. Finally, a section is also added to address any problems that were encountered during this project.

2 Problem Formulation

2.1 Governing equation in the physical and transformed system

This section explains the numerical methodology involved in solving the 2D Euler Finite volume formulation on the airfoil. The 2D Euler equation in physical space is given by:

$$\frac{\partial Q}{\partial t} + \frac{\partial E}{\partial x} + \frac{\partial F}{\partial y} = 0 \tag{1}$$

Here, Q is the flow properties vector, and E and F are the fluxe vectors.

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho et \end{bmatrix}, \qquad E = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho h_t u \end{bmatrix}, \qquad F = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho h_t v \end{bmatrix} \tag{2}$$

The Finite Volume formulation of 1 is given by:

$$\hat{Q}^{q+1} - \hat{Q}^{q} = -\Delta\tau \left[\hat{E}_\xi S_\xi |_{i-1/2,j}^{i+1/2,j} + \hat{F}_\eta S_\eta |_{i,j-1/2}^{i,j+1/2} \right] \tag{3}$$

Where,

$$\hat{Q} = \Delta V \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_t \end{bmatrix}, \qquad \hat{E}_\xi = \begin{bmatrix} \rho U_\xi \\ \rho U_\xi + S_{\xi_x} P \\ \rho V_\xi + S_{\xi_y} P \\ \rho h_t U_\xi \end{bmatrix}, \qquad \hat{F}_\eta = \begin{bmatrix} \rho V_\eta \\ \rho V_\eta + S_{\eta_x} P \\ \rho U_\eta + S_{\eta_y} P \\ \rho h_t V_\eta \end{bmatrix} \tag{4}$$

The right-hand side spatial operator in (3) is called the residual as it goes to zero (or machine precision) when the solution approaches a steady state condition. Therefore, the governing equation is written in the form given by equation 5.

$$Q_{i,j}^{q+1} - Q_{i,j}^{q} = -\Delta\tau \, \mathrm{RHS}_{i,j}^{q} \tag{5}$$

$$\begin{gathered} U_\xi = \vec{u} \cdot \hat{\mathbf{n}}_\xi = \frac{1}{S_\xi}(S_{\xi_x} u + S_{\xi_y} v) \quad ; V_\eta = \vec{u} \cdot \hat{\mathbf{n}}_\eta = \frac{1}{S_\eta}(S_{\eta_x} u + S_{\eta_y} v) \\ S_\xi = \left(S_{\xi_x}^2 + S_{\xi_y}^2 \right)^{1/2} \quad ; S_\eta = \left(S_{\eta_x}^2 + S_{\eta_y}^2 \right)^{1/2} \\ h_t = e_t + \frac{P}{\rho} \end{gathered} \tag{6}$$

2.2 Computational domain

The computational domain is shown in Figure 2 . The airfoil geometry is diamond shaped and symmetric with a maximum thickness of 0.0882 at x = 0.5 and is defined by planar surfaces inclined at a 10-degree angle relative to the leading and trailing edges. Two primary grids

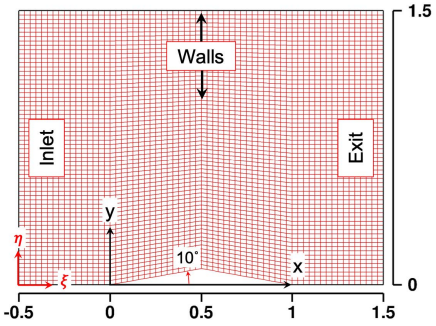

Figure 1: Computational grid

shown in Figure – have been provided to facilitate the development of the Euler solver. The first grid with dimensions of 33×25 and uniformly spaced cells, this grid serves as the basis for developing and debugging the initial Euler solver. The second grid has dimensions of 65×49 and uniformly spaced cells, this grid is employed for computing the "production" level inviscid Euler solver. Grid points within the primary grid are described by both (x, y) coordinates and (i, j) indices, where the i index follows the ξ direction and the j index follows the η direction. The primary grid has integer dimensions $1 \leq i \leq nx$ and $1 \leq j \leq ny$. The computational grid is augmented to include halo/ghost cells that facilitate the construction of fluxes on the boundary walls. The grid metrics such as the volume of the cell stored at cell centers and cell face areas and cell areas stored at the faces are then computed and stored for use in the Euler solver.

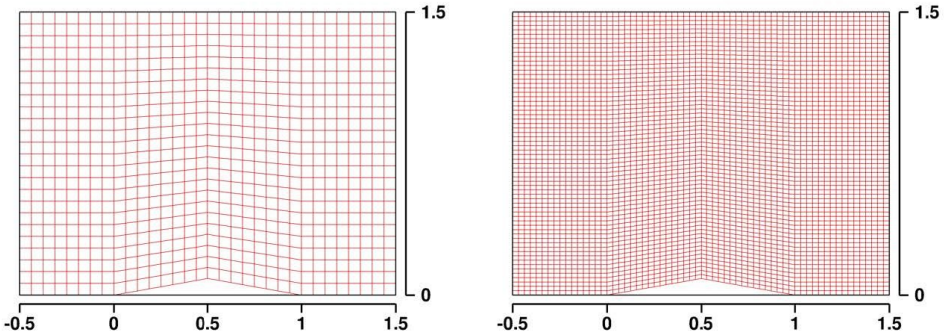

Figure 2: Grids (a) 33 x 25 uniformly spaced cells (b) 65 x 49 uniformly spaced cells

2.2.1 Reference Conditions

Property	Value
Ideal gas constant (R)	287.0 J/(kg·K)
Constant pressure specific heat (C_p)	1005 J/(kg·K)
Ratio of specific heats (γ)	1.400

2.2.2 Initial Conditions

The inlet as shown in Figure 1 is where the flow is initialized to enter the domain at time t=0. Flow direction is parallel to x-axis. These initial conditions are used to initialize the flow in the interior cells. Using these flow properties we can calulate u,v, P,T and therefore calculate the corresponding value of Q at those locations as follows:

$$\rho = \frac{P}{RT} \quad ; \quad \rho u \quad ; \quad \rho v \quad ; \quad \rho e_t = \frac{P}{\gamma - 1} + \frac{1}{2}\rho(u^2 + v^2) \tag{7}$$

Static pressure	101325 Pa
Temperature	300.0 K
Mach number (M)	2.000

2.2.3 Boundary Conditions

The initial conditions also act as boundary conditions for the inlet halo cells on the left inlet boundary and are held constant for each iteration as the flow progresses. The exit boundary on the right remains supersonic, for this boundary the information is extrapolated from the adjacent left interior cells after each iteration. In this manner, $Q_{1,j}^{q+1} = Q_{1,j}^{q}$ remains constant for all q along the inlet plane, while $Q_{nx+1,j}^{q+1} = Q_{nx,j}^{qn}$ is implemented along the exit plane after each time iteration. The bottom and top boundaries are treated as inviscid slip walls. The normal component of the velocity at the slip wall must be zero and the tangent component must be parallel for the slip condition. The following equations are solved to find the velocities in the halo cells to maintain the slip wall boundary condition:

$$\begin{aligned} U_{\xi_1||} &= U_{\xi_2||} \\ V_{\eta_1} &= -V_{\eta_2} \end{aligned} \tag{8}$$

$$\begin{aligned} \left(\frac{1}{S_\eta}\right) \begin{bmatrix} S_{\eta_y} & S_{\eta_x} \\ -S_{\eta_x} & S_{\eta_y} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \end{bmatrix} &= \left(\frac{1}{S_\eta}\right) \begin{bmatrix} S_{\eta_y} & -S_{\eta_x} \\ -S_{\eta_x} & -S_{\eta_y} \end{bmatrix} \begin{bmatrix} u_2 \\ v_2 \end{bmatrix} \\ P_{i,1} &= P_{i,2} \\ T_{i,1} &= T_{i,2} \end{aligned} \tag{9}$$

2.3 Time step selection

The project employs local time stepping as a strategy to converge the solution towards a steady state. Time accuracy is not a primary concern in this context. Instead, the focus is on identifying the most restrictive time step within each cell and direction, determined by a fixed value of CFLmax (Courant-Friedrichs-Lewy number). The CFLmax was set to

1 for the first case, and 0.5 for Case 2 and 3. This approach aims to maximize the rate of convergence to the steady-state solution. Therefore, the time step is given by equation 10.

$$\Delta\tau = \min\left[\mathrm{CFL}_{\max}\min\left(\frac{\Delta\xi}{(|U_\xi|+c)\left(\xi_x^2+\xi_y^2\right)^{\frac{1}{2}}}, \frac{\Delta\eta}{(|V_\eta|+c)\left(\eta_x^2+\eta_y^2\right)^{\frac{1}{2}}}\right)\right]_{i,j} \tag{10}$$

where, the analogy between the finite volume and finite difference metrics is used to define $\xi_x, \xi_y \text{and} \eta_x, \eta_y$ using equation 11.

$$\xi_x = \frac{S_{\xi_x}}{\Delta V}, \xi_y = \frac{S_{\xi_y}}{\Delta V}, \eta_x = \frac{S_{\eta_x}}{\Delta V}, \eta_y = \frac{S_{\eta_y}}{\Delta V} \tag{11}$$

2.4 AUSM$^+$ UP Scheme

The Advection Upstream Splitting Method (AUSM) scheme is the flux vector splitting scheme utilized in this project. The AUSM scheme was introduced by Liou and Steffen in the late 1990s [1] and has since undergone various modifications and improvements.For this project, the most recent paper [2] has been used to construct the AUSM flux splitting in the code. In this scheme the E and F fluxes in the equation 3 are decomposed into convective and pressure flux terms as follows:

$$\hat{\mathbf{E}}_\xi = \rho U_\xi \begin{bmatrix} 1 \\ u \\ v \\ h_t \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{S_{\xi_x}}{S_\xi}P \\ \frac{S_{\xi_y}}{S_\xi}P \\ 0 \end{bmatrix} \tag{12}$$

A similar equation is used for the F fluxes. Th mass flux $\dot{m} = \rho U_\xi$ is a scalar quantity, the term multiplied to $\dot{m}$ is defined as ψ and the second term is expressed as $\mathbf{P}$. Therefore

equation 12 can be written in the form below:

$$\hat{\mathbf{E}}_{\xi} = \dot{m}\psi + \mathbf{P} \tag{13}$$

On the cell face, equation 13 is written as:

$$\hat{\mathbf{E}}_{\xi_{1/2}} = \dot{m}_{1/2}\vec{\psi}_{L/R} + \mathbf{P}_{1/2} \tag{14}$$

The value of $\vec{\psi}_{LR}$ is determined using a simple upwind approach:

$$\vec{\psi}_{L/R} = \begin{cases} \vec{\psi}_L & \text{for } \dot{m_{1/2}} > 0 \\ \vec{\psi}_R & \text{Otherwise} \end{cases} \tag{15}$$

The mass flux at the interface takes the form:

$$\dot{m_{1/2}} = \rho_{L/R} c_{1/2} M_{1/2} \tag{16}$$

Where:

$$\rho_{L/R} = \begin{cases} \rho_L & \text{if } M_{1/2} > 0 \\ \rho_R & \text{otherwise} \end{cases} \tag{17}$$

$$c_{1/2} = \frac{1}{2}(c_L + c_R) \tag{18}$$

In the current approach, Mach number at cell face is calculated using the 4th order polynomial as follows:

$$M_{1/2} = \mathcal{M}^{+}_{(4)}(M_L) + \mathcal{M}^{-}_{(4)}(M_R) + M_p \tag{19}$$

Where:

$$\mathcal{M}^{+}_{(4)}(M) = \begin{cases} \mathcal{M}^{+}_{(1)} & \text{if } |M| \geq 1 \\ \mathcal{M}^{+}_{(2)}(1 - 16\beta\mathcal{M}^{+}_{(2)}) & \text{Otherwise} \end{cases}$$

$$\mathcal{M}^{-}_{(4)}(M) = \begin{cases} \mathcal{M}^{-}_{(1)} & \text{if } |M| \geq 1 \\ \mathcal{M}^{-}_{(2)}(1 + 16\beta\mathcal{M}^{-}_{(2)}) & \text{Otherwise} \end{cases}$$

$$\mathcal{M}^{+}_{(1)}(M) = \frac{1}{2}(M + |M|)$$

$$\mathcal{M}^{-}_{(1)}(M) = \frac{1}{2}(M - |M|)$$

$$\mathcal{M}^{+}_{(2)}(M) = \frac{1}{4}(M + 1)^2$$

$$\mathcal{M}^{-}_{(2)}(M) = \frac{1}{4}(M - 1)^2$$

$$M_p = \frac{-k_p}{f_a}\max\left(1 - \sigma\overline{M}^2, 0\right)\frac{P_R - P_L}{\rho_{1/2}c^2_{1/2}}$$

Then finally the pressure flux is evaluated using 5th order polynomial as follows:

$$\begin{aligned} P_{1/2} &= \mathcal{P}^{+}_{(5)}(M_L)P_L + \mathcal{P}^{-}_{(5)}(M_R)P_R + P_u \\ \mathcal{P}^{+}_{(5)}(M) &= \begin{cases} \frac{1}{M}\mathcal{M}^{+}_{(1)} & \text{if } |M| \geq 1 \\ \mathcal{M}^{+}_{(2)}\left[(2 - M) - 16\alpha M\mathcal{M}^{-}_{(2)}\right] & \text{Otherwise} \end{cases} \\ \mathcal{P}^{-}_{(5)}(M) &= \begin{cases} \frac{1}{M}\mathcal{M}^{-}_{(1)} & \text{if } |M| \geq 1 \\ \mathcal{M}^{-}_{(2)}\left[(-2 - M) + 16\alpha M\mathcal{M}^{+}_{(2)}\right] & \text{Otherwise} \end{cases} \\ P_u &= -k_u\mathcal{P}^{+}_{(5)}(M_L)\mathcal{P}^{-}_{(5)}(M_R)(\rho_L + \rho_R)(f_a c_{1/2})(u_{\xi_R} - u_{\xi_L}) \end{aligned} \tag{20}$$

2.5 MUSCL Interpolation Scheme

The Monotone Upstream Scheme for Conservation Laws (MUSCL) formulation is used to achieve higher-order accuracy based on the Total Variation Diminishing (TVD) theory. The flux vectors in the AUSM scheme are computed at each cell face in the ξ and η directions as a function of the state vector Q. Therefore, we first need to find the state vector Q at the cell faces using the MUSCL interpolation scheme as follows:

$$\begin{aligned} Q^L_{i+1/2} &= Q_i + \frac{\epsilon}{4}(Q_i - Q_{i-1})[(1-k)\phi(r_L) + (1+k)r_L\phi(1/r_L)] \\ Q^R_{i+1/2} &= Q_{i+1} - \frac{\epsilon}{4}(Q_{i+2} - Q_{i+1})[(1+k)r_R\phi(1/r_R) + (1-k)\phi(r_R)] \end{aligned} \tag{21}$$

where,

$$r_L = \frac{\Delta Q_{i+1/2}}{\Delta Q_{i-1/2}} = \frac{Q_{i+1} - Q_i}{Q_i - Q_{i-1}} \quad \text{and} \tag{22}$$

$$r_R = \frac{\Delta Q_{i+1/2}}{\Delta Q_{i+3/2}} = \frac{Q_{i+1} - Q_i}{Q_{i+2} - Q_{i+1}} \tag{23}$$

In the equation 21, ϕ is the minmoid limiter function defined as follows:

$$\phi(r) = \begin{cases} 0 & \text{if } r \leq 0 \\ 1 & \text{if } r \geq 1 \\ r & \text{otherwise} \end{cases}$$

In this project, three cases were explored. The first case is a first-order accurate inviscid Euler solver with the ϵ set to 0. The second case uses a second-order accurate (fully upwind approach) without flux limiter, meaning ϕ is set to 1. The third case uses a minmoid limiter to implement the TVD scheme near the shock.

Case	Epsilon	Kappa
Case 1: 1st order fully upwind	0	-
Case 2: Second order accurate (fully upwind)	1	-1
Case 3: Second order accurate (fully upwind with minmoid limiter)	1	-1

3 Results and Discussion

The developed solver was used to test three cases with different levels of accuracy, comparing them against the 1D analytical shock equations for the oblique shock near the leading edge. This section presents the resultant Mach number, pressure, density, and temperature contours for all three cases, along with convergence plots demonstrating convergence to machine precision. Table 1 presents the results compared against the analytical data. It is to be noted that all three cases were run using the finer grid of 65 x 48 cells.

Analytical Data	**Case 1** ($\epsilon = 0$)	**Case 2** ($\epsilon = 1$, $\kappa = -1$)	**Case 3** ($\epsilon = 1$, $\kappa = -1$,with minmoid limiter)
Wave angle = 39.31°	39.86°	39.98°	39.5°
$M_2 = 1.641$	1.6372	1.6617	1.632
$P_2/P_1 = 1.707$	1.75	1.76	1.8122
$\rho_2/\rho_1 = 1.458$	1.45	1.462	1.4940
$T_2/T_1 = 1.170$	1.1752	1.1648	1.178
$P_{02}/P_{01} = 0.9846$	0.9824	0.9819	0.979

Table 1: Comparison of results with verification data.

The table above shows the comparison of the 1D analytical data with the 2D Euler solution using the three cases. The first case is a first order scheme with no switchin mechanism. Therefore it does well in the vicinity of the shock but is highly dissipative because of the low order of accuracy. Whereas the 2nd case is higher order scheme - third order accurate

upwind, but it doesn not do well near the vicinity of the shock as these are more prone to oscillations. The third scheme using a TVD scheme, the minmoid limiter function to switch to first order near the shock and higher order away from the shock. This allows it to capture the behavior of the shock with much less dissipation.

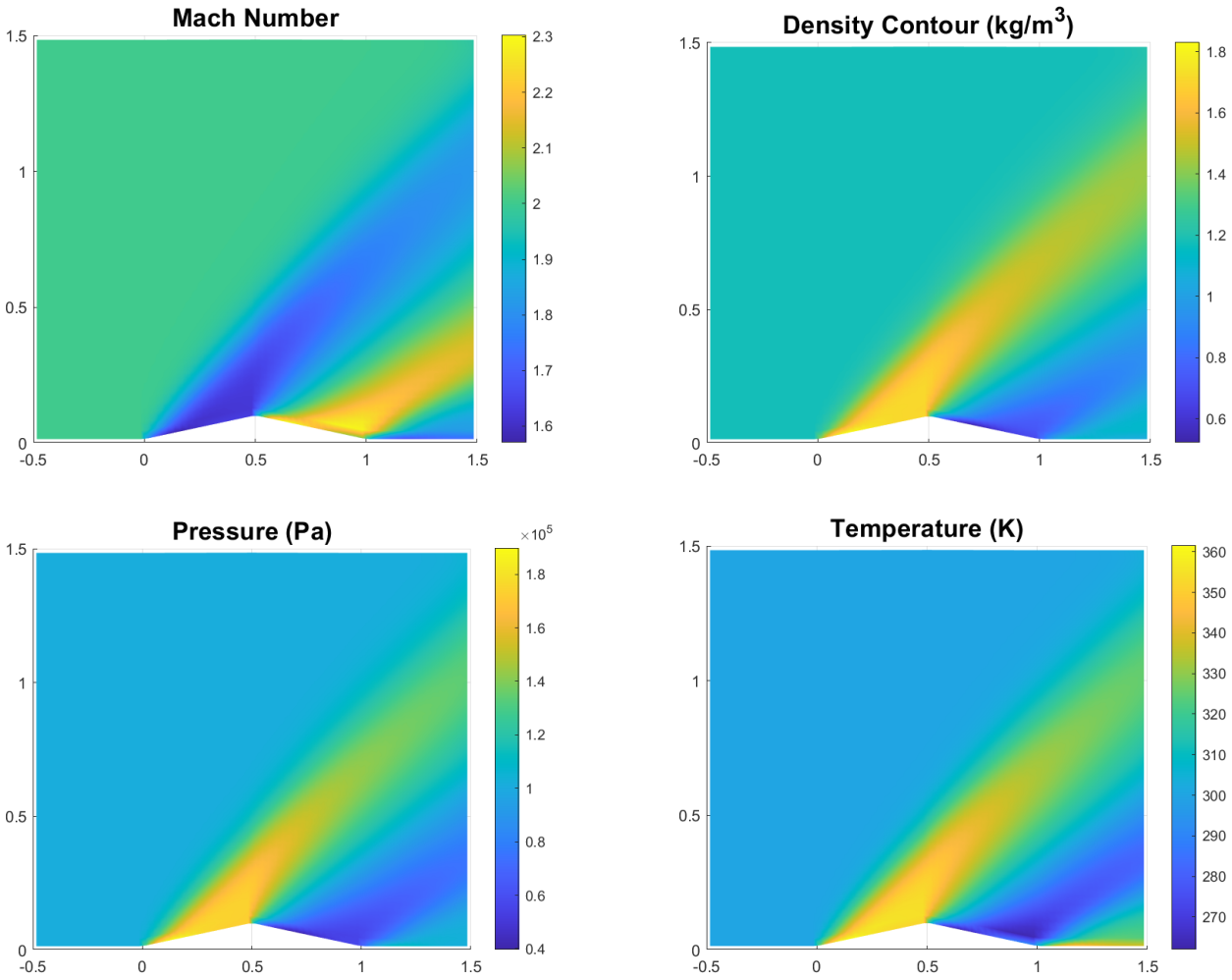

Figure 3: Case 1, Inviscid Euler, first-order accurate.

The contour plots for Cases 1 through 3 are shown in Figures 3, 4, 5 . All three cases can capture the oblique shock, but the first case is more dissipative, the second is less dissipative but more oscillatory and the third case is a good balance and is the best of all three. While

the three schemes show reasonably close values to the 1D data, it is important to note that the 1D data is an approximation for an infinitely thin shock. In the 2D Euler solver, artificial and numerical dissipation is observed, resulting in a smeared shock. Therefore, to determine the location of the shock, it was assumed that the location of the shock was at the maximum pressure gradient location.

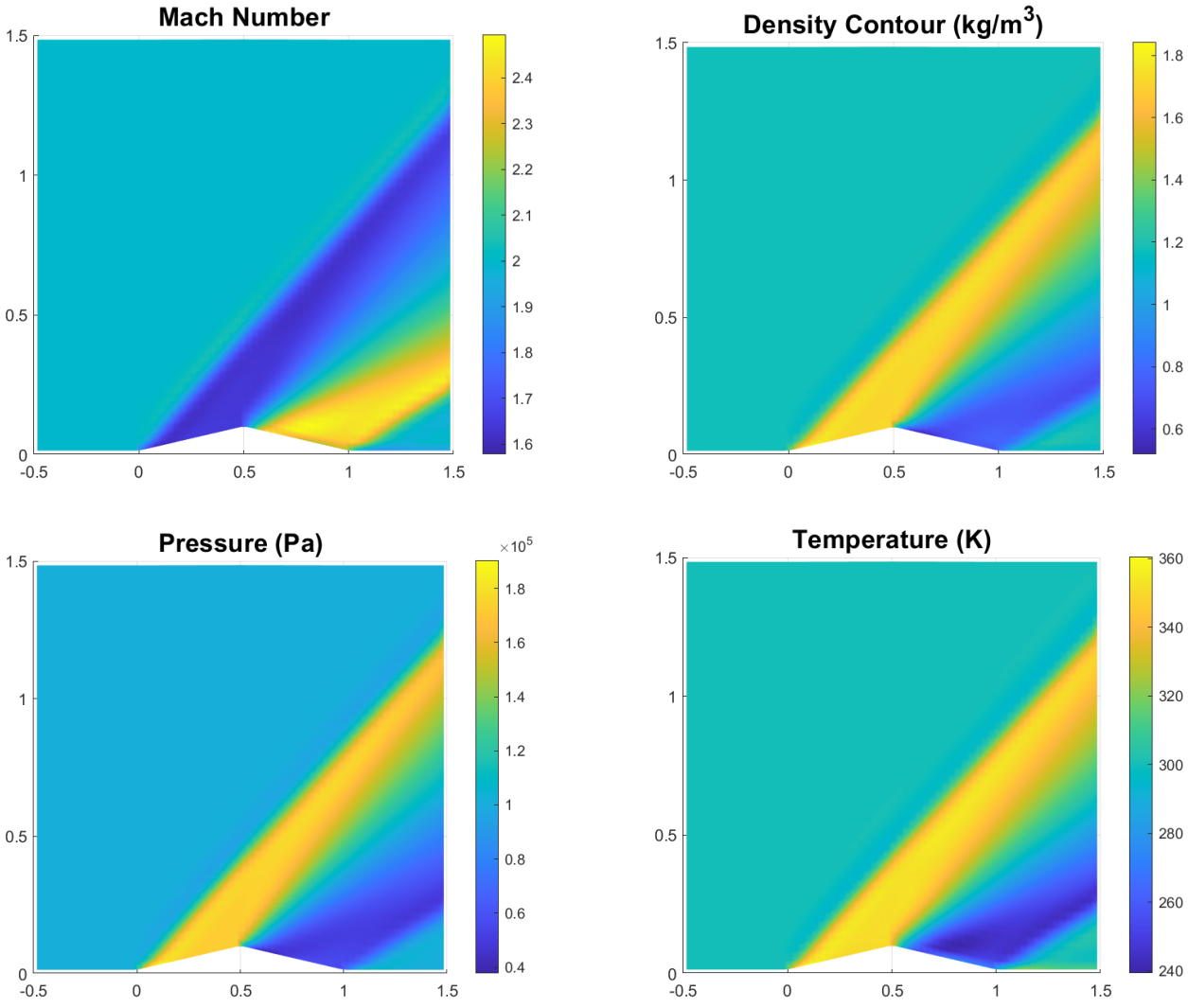

Figure 4: Case 2, Inviscid Euler, second-order accurate (fully upwind) without flux limiter.

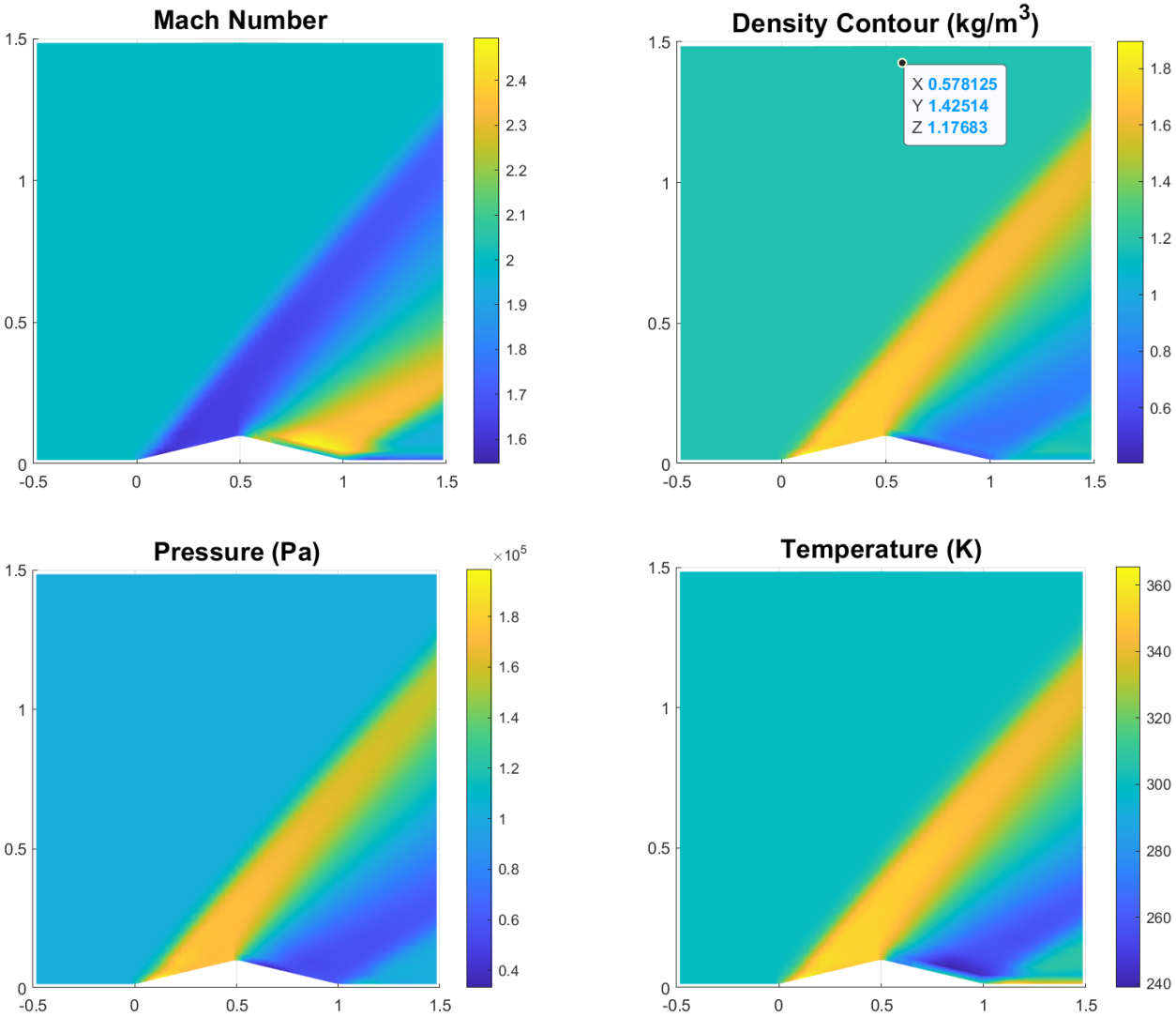

Figure 5: Case 3, Inviscid Euler, second-order accurate (fully upwind) with basic minmod limiter.

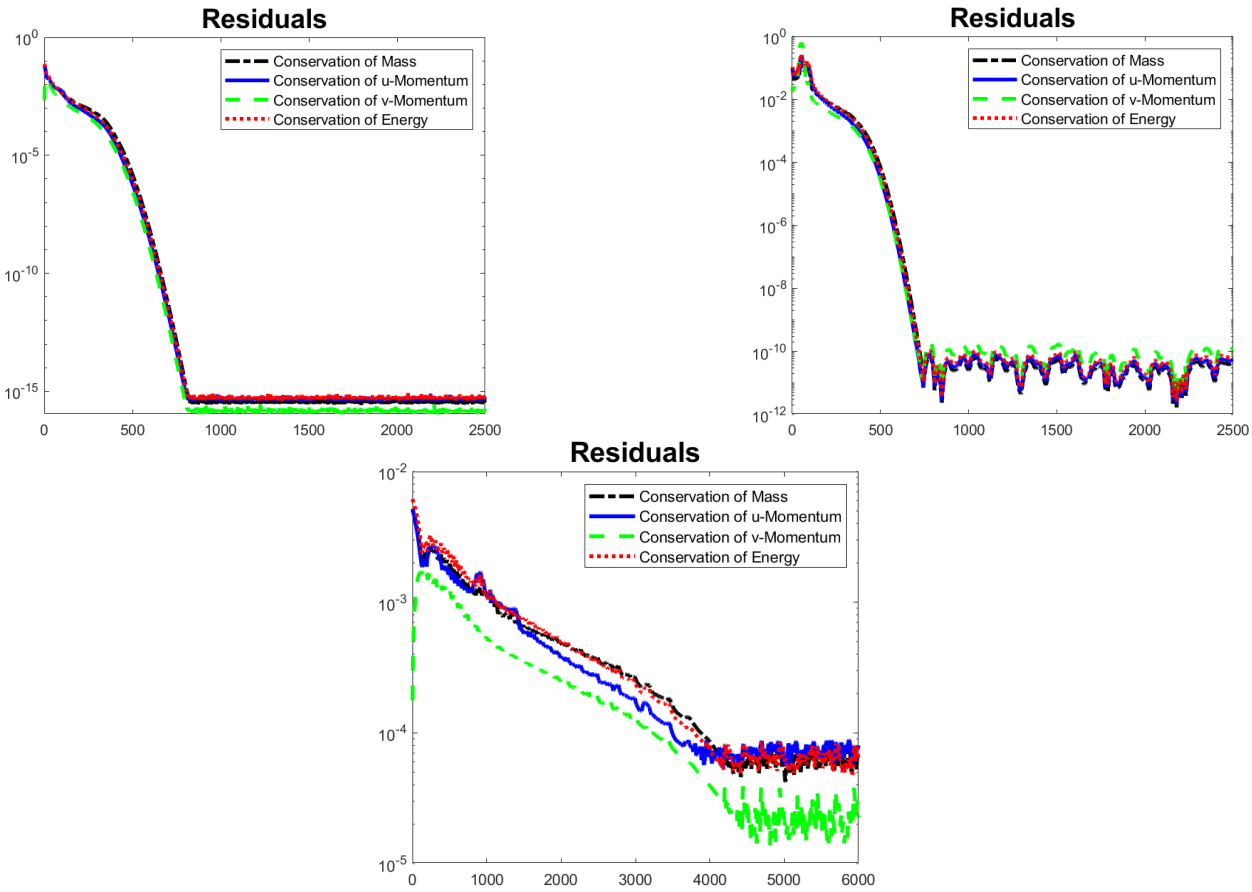

Figure 6: Residuals for Case 1, 2 and 3.

Representative convergence plots using the L_∞ norm are plotted for the continuity, u-momentum, v-momentum, and energy residuals for all three cases, as shown in Figure 6. The residuals converge to machine precision for the first and second cases but fail to go below 10^{-6} for the third case, which may be attributable to subtractive cancellation.

4 Conclusions

In conclusion, the developed two-dimensional finite-volume Euler solver underwent rigorous testing across three cases, beginning with a coarser grid and progressing to a finer grid. Each

case represents varying levels of spatial accuracy and numerical stability. The results indicate that while all cases reasonably capture the oblique shock, significant differences emerge in terms of dissipation, oscillations, and overall accuracy. The first case, employing a first-order accurate scheme, exhibits high dissipation near the shock but lacks accuracy away from it. Utilizing a second-order accurate scheme without a flux limiter, the second case reduces dissipation but introduces oscillations near the shock region. Finally, the third case, employing a second-order accurate scheme with a basic minmod limiter, strikes a balance between accuracy and stability. However, despite the advancements in accuracy, the solver struggles to achieve convergence to machine precision in the third case. Further refinement and optimization of the code may be necessary to overcome this limitation.

Overall, the results align with the verification data, and this project provided a solid foundation for future CFD projects.

References

[1] Meng-Sing Liou and Christopher J. Steffen. "A New Flux Splitting Scheme". In: *Journal of Computational Physics* 107.1 (1993), pp. 23–39. ISSN: 0021-9991. DOI: 10.1006/jcph.1993.1122. URL: https://www.sciencedirect.com/science/article/pii/S0021999183711228.

[2] M.-S. Liou. "A sequel to AUSM, part II: AUSM+-up for all speeds". In: *Journal of Computational Physics* 214 (2006), pp. 137–170.

5 Appendix

This section contains the code that has been developed to generate all the results and also the subroutines/functions involved.

```
%%% AE 6042 Final Project
% Name: Aishwerya Gahlot
% Instructor: Dr. Joseph Oefelein
% Date: 4/29/2024

clc;clear; clear all;

% Read the grid.
[nx,ny,x2d,y2d] = read_grid('g65x49u-1.dat');

% Compute Grid metrics - Cell face area, cell area, volume of cell and
% construct halo cells.
[x2d,y2d,xc,yc,xu,yu,xv,yv,Vc,S_eta,S_eta_x,S_eta_y,S_xi,S_xi_x,S_xi_y,yeta,xeta,yxi,
xxi] = gridcompute(x2d,y2d,nx,ny);

% Thermodynamic and Transport Properties
gamma = 1.4  ;          % ratio of specific heats
Cp    = 1005 ;          % J/kg-K
R     = 287  ;          % J/kg-K
P_ref = 101325;         %Pa
T_ref = 300   ;         %K
M_ref = 2   ;
c_ref  = 347.2 ;        %m/s
V_ref = M_ref*c_ref ;   %m/s based on Mach 2

% Pre-allocation
[Q, U, E, F, RHS, P, T] = pre(nx,ny);

%% Initialization
[Q,U] = Intialize(Q,U,P,T,
P_ref,V_ref,R,T_ref,gamma,nx,ny,S_eta,S_eta_x,S_eta_y,yxi,xxi,xu,yu);

iter = 2500;
CFLmax = 1;

for q = 1:iter

    [dt] = local_time(CFLmax, Q,nx,ny,yeta,xeta,yxi,xxi,Vc,gamma,R,x2d,y2d);

    [Q,U]   = MUSCL2(Q,U,nx,ny);

    % Turn this on when doing Case 1.
    % [E,F] = FLUX(Q,gamma,nx,ny,S_eta_x,S_eta_y, S_xi_x, S_xi_y, S_xi, S_eta);

    [E,F,U] = AUSM_UP(Q,U, nx,ny,gamma, R, S_xi_x,S_xi_y,S_xi,S_eta_x,S_eta_y,
S_eta);

    for j=2:ny
        for i=2:nx
            E.dE1(i,j) = E.E1(i+1,j)*S_xi(i+1,j) - E.E1(i,j)*S_xi(i,j);
            E.dE2(i,j) = E.E2(i+1,j)*S_xi(i+1,j) - E.E2(i,j)*S_xi(i,j);
            E.dE3(i,j) = E.E3(i+1,j)*S_xi(i+1,j) - E.E3(i,j)*S_xi(i,j);
            E.dE4(i,j) = E.E4(i+1,j)*S_xi(i+1,j) - E.E4(i,j)*S_xi(i,j);
        end
    end
```

```
for i=2:nx
    for j=2:ny
        F.dF1(i,j) =  F.F1(i,j+1)*S_eta(i,j+1) - F.F1(i,j)*S_eta(i,j);
        F.dF2(i,j) =  F.F2(i,j+1)*S_eta(i,j+1) - F.F2(i,j)*S_eta(i,j);
        F.dF3(i,j) =  F.F3(i,j+1)*S_eta(i,j+1) - F.F3(i,j)*S_eta(i,j);
        F.dF4(i,j) =  F.F4(i,j+1)*S_eta(i,j+1) - F.F4(i,j)*S_eta(i,j);
    end
end

for i= 2:nx
    for j= 2:ny
        RHS.rho(i,j) =  -  (dt(i,j)/Vc(i,j))*(E.dE1(i,j) + F.dF1(i,j));
        RHS.rhou(i,j) = -  (dt(i,j)/Vc(i,j))*(E.dE2(i,j) + F.dF2(i,j));
        RHS.rhov(i,j) = -  (dt(i,j)/Vc(i,j))*(E.dE3(i,j) + F.dF3(i,j));
        RHS.rhoet(i,j) = - (dt(i,j)/Vc(i,j))*(E.dE4(i,j) + F.dF4(i,j));
    end
end


% time-accurate (for debugging)
%dt = 1e-5; % Fixed Time step
% for i= 2:nx
%     for j= 2:ny
%         RHS.rho(i,j) =  -  (dt/Vc(i,j))*(E.dE1(i,j) + F.dF1(i,j));
%         RHS.rhou(i,j) = -  (dt/Vc(i,j))*(E.dE2(i,j) + F.dF2(i,j));
%         RHS.rhov(i,j) = -  (dt/Vc(i,j))*(E.dE3(i,j) + F.dF3(i,j));
%         RHS.rhoet(i,j) = - (dt/Vc(i,j))*(E.dE4(i,j) + F.dF4(i,j));
%     end
% end

%% Update the state vector
Q.rho   = Q.rho   + RHS.rho;
Q.rhou  = Q.rhou  + RHS.rhou;
Q.rhov  = Q.rhov  + RHS.rhov;
Q.rhoet = Q.rhoet + RHS.rhoet;

%plotVvector(Q,nx,ny,xc,yc,x2d,y2d, 1)

Q = bc(Q,P_ref,V_ref,R,T_ref,gamma,nx,ny,S_eta,S_eta_x,S_eta_y,yxi,xxi);

% Plot velocity vectors to ensure the BC is being applied correctly.
plotVvector(Q,nx,ny,xc,yc,x2d,y2d, 2)

%% Residuals

for i= 2:nx
    for j= 2:ny
        Residual.rho(i,j) = RHS.rho(i,j)/Q.ref.rho(i,j);
        Residual.rhou(i,j) = RHS.rhou(i,j)/Q.ref.rhou(i,j);
        Residual.rhov(i,j) = RHS.rhov(i,j)/Q.ref.rhov(i,j);
        Residual.rhoet(i,j) = RHS.rhoet(i,j)/Q.ref.rhoet(i,j);
    end
end
```

```
    % Infinity norm
    Continuity(q) = max(max(abs(Residual.rho)));
    U_mom(q)      = max(max(abs(Residual.rhou)));
    V_mom(q)      = max(max(abs(Residual.rhov)));
    Energy(q)     = max(max(abs(Residual.rhoet)));

    % Print out the residuals
    fprintf('%10d, %15.6e %15.6e %15.6e %15.6e\n', q, Continuity(q),
U_mom(q),V_mom(q) , Energy(q));

    if Continuity(q) < 1e-16 && U_mom(q) < 1e-16 && V_mom(q) < 1e-16 && Energy(q) <
1e-16
        break;
    end


end

%% Store Results for Post-Processing

for i = 2:nx
    for j = 2:ny
        Results.U.u(i,j) = Q.rhou(i,j)/Q.rho(i,j);
        Results.U.v(i,j) = Q.rhov(i,j)/Q.rho(i,j);
        Results.Temp(i,j) = ((gamma-1)/(2*((Q.rho(i,j))^2)*R)) *
((2*Q.rho(i,j)*Q.rhoet(i,j)) - ((Q.rhou(i,j))^2 + (Q.rhov(i,j))^2));
        c_center = sqrt(gamma*R*Results.Temp(i,j));
        Results.V_center(i,j) = sqrt(Results.U.u(i,j)^2 + (Results.U.v(i,j)^2));
        Results.Mach(i,j) = Results.V_center(i,j)/c_center;
        Results.Density(i,j) = Q.rho(i,j);
        Results.Pressure(i,j) = ((gamma-1)/(2*Q.rho(i,j))) *
((2*Q.rho(i,j)*Q.rhoet(i,j)) - ((Q.rhou(i,j)^2) + (Q.rhov(i,j)^2)));
        Results.StagPressure(i,j) = Results.Pressure(i,j) +
0.5*Results.Density(i,j)*(Results.U.u(i,j)^2 + (Results.U.v(i,j)^2)) ;
    end

end

% Plot the L infinity norms
figure(); clf;
semilogy(1:q,Continuity,'-.k','Linewidth',2.5)
hold on
semilogy(1:q,U_mom,'b','Linewidth',2.5)
semilogy(1:q,V_mom,'--g','Linewidth',2.5)
semilogy(1:q,Energy,':r','Linewidth',2.5)
title('Residuals','Fontsize',18)
lgd = legend ({'Conservation of Mass','Conservation of u-Momentum','Conservation of
v-Momentum','Conservation of Energy'},'Fontsize',10);

%%

figure(); clf
surf(xc(2:nx,2:ny),yc(2:nx,2:ny),(Results.Mach(2:nx,2:ny)));
az = 0;
el = 90;
```

```
view(az,el);
% clim([1.6 1.64])
colorbar
title('Mach Number','Fontsize',16);
shading interp

figure(10); clf
[C, h] = contour3(xc(2:nx,2:ny), yc(2:nx,2:ny), Results.Mach(2:nx,2:ny), 10);
clabel(C, h, 'FontSize', 10);
az = 0;
el = 90;
view(az,el);

figure(); clf
surf(xc(2:nx,2:ny),yc(2:nx,2:ny),(Results.Density(2:nx,2:ny)));
az = 0;
el = 90;
view(az,el);
colorbar
title('Density Contour','Fontsize',16);
shading interp
figure(2); clf
[C, h] = contour3(xc(2:nx,2:ny), yc(2:nx,2:ny), Results.Density(2:nx,2:ny), 10);
clabel(C, h, 'FontSize', 10); % set label font size
az = 0;
el = 90;
view(az,el);

figure(); clf
surf(xc(2:nx,2:ny),yc(2:nx,2:ny),(Results.V_center(2:nx,2:ny)));
az = 0;
el = 90;
view(az,el);
colorbar
title('Velocity Magnitude (m/s)','Fontsize',16);
shading interp

figure(1); clf
[C, h] = contour3(xc(2:nx,2:ny), yc(2:nx,2:ny), Results.V_center(2:nx,2:ny), 10);
clabel(C, h, 'FontSize', 10);
az = 0;
el = 90;
view(az,el);

figure(); clf
surf(xc(2:nx,2:ny),yc(2:nx,2:ny),(Results.Pressure(2:nx,2:ny)));
az = 0;
el = 90;
view(az,el);
colorbar
title('Pressure (Pa)','Fontsize',16);
shading interp

figure(3); clf
```

```
[C, h] = contour3(xc(2:nx,2:ny), yc(2:nx,2:ny), Results.Pressure(2:nx,2:ny)./100000,
10);
clabel(C, h, 'FontSize', 10);
az = 0;
el = 90;
view(az,el);

figure(); clf
surf(xc(2:nx,2:ny),yc(2:nx,2:ny),(Results.Temp(2:nx,2:ny)));
az = 0;
el = 90;
view(az,el);
colorbar
title('Temperature (K)','Fontsize',16);
shading interp

figure(); clf
[C, h] = contour3(xc(2:nx,2:ny), yc(2:nx,2:ny), Results.Temp(2:nx,2:ny), 10);
clabel(C, h, 'FontSize', 10);
az = 0;
el = 90;
view(az,el);

figure(); clf
streamslice(xc(2:nx,2:ny)',yc(2:nx,2:ny)',Results.U.u(2:nx,2:ny)',
Results.U.v(2:nx,2:ny)',2)
hold on;
plot(x2d(1:nx+2,2), y2d(1:nx+2,2), 'k-','LineWidth',1);


figure(); clf
surf(xc(2:nx,2:ny),yc(2:nx,2:ny),(Results.StagPressure(2:nx,2:ny)));
az = 0;
el = 90;
view(az,el);
colorbar
title('Stagnation pressure','Fontsize',16);
shading interp

figure(); clf
[C, h] = contour3(xc(2:nx,2:ny), yc(2:nx,2:ny), Results.StagPressure(2:nx,2:ny), 10);
% adjust number of contour levels as needed
clabel(C, h, 'FontSize', 10);
az = 0;
el = 90;
view(az,el);


function [nx,ny,x2d,y2d] = read_grid(grid_filename)
fileID = fopen(grid_filename, 'r'); %Open the file
dimensions = fscanf(fileID, '%d, %d', 2); % Read the dimensions
nx = dimensions(1);
ny = dimensions(2);
x2d = zeros(nx+2,ny+2); %+2 to account for halo cell allocation
y2d = zeros(nx+2,ny+2);
```

```
for j = 2:ny+1
for i = 2:nx+1
    temp = fscanf(fileID, '%f, %f', 2);
    x2d(i,j) = temp(1);
    y2d(i,j) = temp(2);
end
end
fclose(fileID);
end


function
[x2d,y2d,xc,yc,xu,yu,xv,yv,Vc,S_eta,S_eta_x,S_eta_y,S_xi,S_xi_x,S_xi_y,yeta,xeta,yxi,
xxi] = gridcompute(x2d,y2d,nx,ny)
%% Plotting  the grid

% figure(1); clf
% hold on;
% for i = 2:nx+1
%     plot(x2d(i,2:ny+1), y2d(i,2:ny+1), 'k-','LineWidth',1); % vertical grid lines
% end
%
% for j =2:ny+1
%     plot(x2d(2:nx+1,j), y2d(2:nx+1,j), 'b-','LineWidth',1); % horizontal grid lines
% end

% xlabel('x','FontSize',14);
% ylabel('y','FontSize',14);
% title('Computational grid','FontSize',14);
% axis equal;
% grid on;
% hold off;

%% Halo Cell Computation via Extrapolation

for j =2:ny+1
% Left wall
x2d(1,j) = x2d(2,j) -(x2d(3,j)-x2d(2,j));
y2d(1,j) = y2d(2,j) + ((y2d(3,j)-y2d(2,j))/(x2d(3,j)-x2d(2,j)))*(x2d(1,j)-x2d(2,j));
% Right wall
x2d(nx+2,j) = x2d(nx+1,j) +(x2d(nx+1,j)-x2d(nx,j));
y2d(nx+2,j) = y2d(nx+1,j) + ((y2d(nx+1,j)-y2d(nx,j))/(x2d(nx+1,j)-
x2d(nx,j)))*(x2d(nx+2,j)-x2d(nx+1,j));
end

for i =2:nx+1
% Top wall
%y2d(i,ny+2) = y2d(i,ny+1) +(y2d(i,ny+1)-y2d(i,ny));
y2d(i,ny+2) = y2d(i,ny+1) +(y2d(1,ny+1)-y2d(1,ny));%maining the same width
x2d(i,ny+2) = x2d(i,ny+1) + ((x2d(i,ny+1)-x2d(i,ny))/(y2d(i,ny+1)-
y2d(i,ny)))*(y2d(i,ny+2)-y2d(i,ny+1));
% Bottom wall
%y2d(i,1) = y2d(i,2) -(y2d(i,3)-y2d(i,2));
y2d(i,1) = y2d(i,2) -(y2d(1,3)-y2d(1,2)); %maintaining the same width
x2d(i,1) = x2d(i,2) + ((x2d(i,3)-x2d(i,2))/(y2d(i,3)-y2d(i,2)))*(y2d(i,1)-y2d(i,2));
```

```
end

%% Corner points

%Bottom Left
x2dbl(1,1) = x2d(2,1) -(x2d(3,1)-x2d(2,1)); % bottom left
y2dbl(1,1) = y2d(2,1) + ((y2d(3,1)-y2d(2,1))/(x2d(3,1)-x2d(2,1)))*(x2dbl(1,1)-
x2d(2,1));
y2dbd(1,1) = y2d(1,2) -(y2d(1,3)-y2d(1,2)); % bottom down
x2dbd(1,1) = x2d(1,2) + ((x2d(1,3)-x2d(1,2))/(y2d(1,3)-y2d(1,2)))*(y2dbd(1,1)-
y2d(1,2));
x2d(1,1) = 0.5*(x2dbl(1,1) + x2dbd(1,1));
y2d(1,1) = 0.5*(y2dbl(1,1) + y2dbd(1,1));

%Top Left
x2dtl(1,ny+2) = x2d(2,ny+2) -(x2d(3,ny+2)-x2d(2,ny+2)); % top left
y2dtl(1,ny+2) = y2d(2,ny+2) + ((y2d(3,ny+2)-y2d(2,ny+2))/(x2d(3,ny+2)-
x2d(2,ny+2)))*(x2dtl(1,ny+2)-x2d(2,ny+2));
y2dbu(1,ny+2) = y2d(1,ny+1) +(y2d(1,ny+1)-y2d(1,ny));   % bottom up
x2dbu(1,ny+2) = x2d(1,ny+1) + ((x2d(1,ny+1)-x2d(1,ny))/(y2d(1,ny+1)-
y2d(1,ny)))*(y2dbu(1,ny+2)-y2d(1,ny+1));
x2d(1,ny+2) = 0.5*(x2dtl(1,ny+2) + x2dbu(1,ny+2));
y2d(1,ny+2) = 0.5*(y2dtl(1,ny+2) + y2dbu(1,ny+2));

%Bottom Right
x2dbr(nx+2,1) = x2d(nx+1,1) +(x2d(nx+1,1)-x2d(nx,1)); % bottom right
y2dbr(nx+2,1) = y2d(nx+1,1) + ((y2d(nx+1,1)-y2d(nx,1))/(x2d(nx+1,1)-
x2d(nx,1)))*(x2dbr(nx+2,1)-x2d(nx+1,1));
y2dbrd(nx+2,1) = y2d(nx+2,2) -(y2d(nx+2,3)-y2d(nx+2,2)); % bottom down
x2dbrd(nx+2,1) = x2d(nx+2,2) + ((x2d(nx+2,3)-x2d(nx+2,2))/(y2d(nx+2,3)-
y2d(nx+2,2)))*(y2dbrd(nx+2,1)-y2d(nx+2,2));
x2d(nx+2,1) = 0.5*(x2dbr(nx+2,1)+x2dbrd(nx+2,1)) ;
y2d(nx+2,1) =0.5*(y2dbr(nx+2,1)+y2dbrd(nx+2,1)) ;

%Top Right
x2dtr(nx+2,ny+2) = x2d(nx+1,ny+2) +(x2d(nx+1,ny+2)-x2d(nx,ny+2)); % top right
y2dtr(nx+2,ny+2) = y2d(nx+1,ny+2) + ((y2d(nx+1,ny+2)-y2d(nx,ny+2))/(x2d(nx+1,ny+2)-
x2d(nx,ny+2)))*(x2dtr(nx+2,ny+2)-x2d(nx+1,ny+2));
y2dbu(nx+2,ny+2) = y2d(nx+2,ny+1) +(y2d(nx+2,ny+1)-y2d(nx+2,ny));   % bottom up
x2dbu(nx+2,ny+2) = x2d(nx+2,ny+1) + ((x2d(nx+2,ny+1)-x2d(nx+2,ny))/(y2d(nx+2,ny+1)-
y2d(nx+2,ny)))*(y2dbu(nx+2,ny+2)-y2d(nx+2,ny+1));
x2d(nx+2, ny+2)  = 0.5*(x2dtr(nx+2,ny+2) + x2dbu(nx+2,ny+2));
y2d(nx+2, ny+2)  = 0.5*(y2dtr(nx+2,ny+2) + y2dbu(nx+2,ny+2));


%% Plotting  the grid with Halo Cells

% figure(2);
% hold on;
% for i = 1:nx+2
%     plot(x2d(i,1:ny+2), y2d(i,1:ny+2), 'b-','LineWidth',1); % vertical grid lines
% end
%
% for j =1:ny+2
%     plot(x2d(1:nx+2,j), y2d(1:nx+2,j), 'b-','LineWidth',1); % horizontal grid lines
```

```
% end
%
% xlabel('x','FontSize',14);
% ylabel('y','FontSize',14);
% title('Computational grid with Halo Cells','FontSize',14);
% axis equal;
% grid on;
%
% plot(x2d(1,1:ny+2), y2d(1,1:ny+2), 'r-','LineWidth',2); % Left Wall points
extrapolated
% plot(x2d(nx+2,1:ny+2), y2d(nx+2,1:ny+2), 'r-','LineWidth',2); % Right Wall points
extrapolated
%
% plot(x2d(1:nx+2,ny+2), y2d(1:nx+2,ny+2), 'r-','LineWidth',2); % Top Wall points
extrapolated
% plot(x2d(1:nx+2,1), y2d(1:nx+2,1), 'r-','LineWidth',2); % Bottom Wall points
extrapolated
%
% plot(x2d(2,2:ny+1), y2d(2,2:ny+1), 'r-','LineWidth',2); % Left Wall points
extrapolated
% plot(x2d(nx+1,2:ny+1), y2d(nx+1,2:ny+1), 'r-','LineWidth',2); % Right Wall points
extrapolated
%
% plot(x2d(2:nx+1,ny+1), y2d(2:nx+1,ny+1), 'r-','LineWidth',2); % Top Wall points
extrapolated
% plot(x2d(2:nx+1,2), y2d(2:nx+1,2), 'r-','LineWidth',2); % Bottom Wall points
extrapolated
% hold off;

%% Part #4 Cell Volume and cell face area computations

xc = zeros(nx+1,ny+1);
yc = zeros(nx+1,ny+1);

% Define cell centered cooordinates. These are 1/4 of surrounding cells.

for j = 1:ny+1
for i = 1:nx+1
xc(i,j) = (1/4) * (x2d(i,j) + x2d(i+1,j) + x2d(i,j+1) + x2d(i+1,j+1));
yc(i,j) = (1/4) * (y2d(i,j) + y2d(i+1,j) + y2d(i,j+1) + y2d(i+1,j+1));
end
end

% Cell Volume stored at cell centeroid

Vc = zeros(nx+1,ny+1);
for j = 1:ny+1
for i = 1:nx+1
Vc(i,j) = 0.5*( (x2d(i+1,j+1)-x2d(i,j)) * (y2d(i,j+1)-y2d(i+1,j))...
              - (x2d(i,j+1)-x2d(i+1,j)) * (y2d(i+1,j+1)-y2d(i,j)) );
end
end

% figure(3); clf;
% contourf(xc,yc,Vc)
```

```
% h = colorbar;
% xlabel('x');
% ylabel('y');
% title('Contours of Cell Volume');
% title(h, 'Cell Volume');

% Cell Face area computations

xu = zeros(nx+2,ny+1); % Cell face center normal to xi direction
yu = zeros(nx+2,ny+1);
xv = zeros(nx+1,ny+2); % Cell face center normal to eta direction
yv = zeros(nx+1,ny+2);

% Cell Face center coordinates

% face normal to xi direction: horizontal direction
for j = 1:ny+1
for i = 1:nx+2
xu(i,j) = 0.5*(x2d(i,j)+x2d(i,j+1));
yu(i,j) = 0.5*(y2d(i,j)+y2d(i,j+1));
end
end
% face normal to eta direction: vertical direction
for j = 1:ny+2
for i = 1:nx+1
xv(i,j) = 0.5*(x2d(i,j)+x2d(i+1,j));
yv(i,j) = 0.5*(y2d(i,j)+y2d(i+1,j));
end
end

%Cell face areas

%%
yeta = zeros(nx+2,ny+1);
xeta = zeros(nx+2,ny+1);
S_xi = zeros(nx+2,ny+1); % faces normal to xi

for j = 1:ny+1
for i = 1:nx+2
yeta(i,j)  =  y2d(i,j+1) - y2d(i,j);
xeta(i,j)  =  x2d(i,j+1) - x2d(i,j);
S_xi(i,j) = sqrt(yeta(i,j)^2 + xeta(i,j)^2);
end
end

%% Components of S_xi
S_xi_x = yeta;
S_xi_y = - xeta;

yxi = zeros(nx+1,ny+2);
xxi = zeros(nx+1,ny+2);
S_eta = zeros(nx+1,ny+2);   % faces normal to eta

for j = 1:ny+2
for i = 1:nx+1
```

```
yxi(i,j) = y2d(i+1,j) - y2d(i,j);
xxi(i,j) = x2d(i+1,j) - x2d(i,j);
S_eta(i,j) = sqrt(yxi(i,j)^2 + xxi(i,j)^2);
end
end

%% Components of S_eta

S_eta_x = - yxi ;
S_eta_y =   xxi ;

% Plot surface area normal to \xi direction
% figure(4); clf;
% contourf(xu,yu,S_xi)
% h = colorbar;
% xlabel('x');
% ylabel('y');
% title('Cell face area normal to \xi direction');
% title(h, 'Cell face Area');

%% Plot surface vectors normal to \xi direction
% figure(5);clf;
% quiver(xu,yu,S_xi_x,S_xi_y)
% hold on;
% for i = 1:nx+2
%     plot(x2d(i,1:ny+2), y2d(i,1:ny+2), 'k-','LineWidth',1); % vertical grid lines
% end
%
% for j =1:ny+2
%     plot(x2d(1:nx+2,j), y2d(1:nx+2,j), 'b-','LineWidth',1); % horizontal grid lines
% end

% figure(6); clf;
% contourf(xv,yv,S_eta)
% h = colorbar;
% xlabel('x');
% ylabel('y');
% title('Cell face area normal to \eta direction');
% title(h, 'Cell face Area');

%% Plot surface vectors normal to \eta direction
% figure(7);clf;
% quiver(xv,yv,S_eta_x,S_eta_y)
% hold on;
% for i = 1:nx+2
%     plot(x2d(i,1:ny+2), y2d(i,1:ny+2), 'k-','LineWidth',1); % vertical grid lines
% end
%
% for j =1:ny+2
%     plot(x2d(1:nx+2,j), y2d(1:nx+2,j), 'k-','LineWidth',1); % horizontal grid lines
% end

function [Q, U, E, F, RHS, P, T] = pre(nx,ny)
% Preallocation
Q.LQj.Qrho = zeros(nx+1,ny+2);
```

```
Q.LQi.Qrho = zeros(nx+2,ny+1);
Q.LQj.Qrhou = zeros(nx+1,ny+2);
Q.LQi.Qrhou = zeros(nx+2,ny+1);
Q.LQj.Qrhov = zeros(nx+1,ny+2);
Q.LQi.Qrhov = zeros(nx+2,ny+1);
Q.LQj.Qrhoet = zeros(nx+1,ny+2);
Q.LQi.Qrhoet = zeros(nx+2,ny+1);
U.Lui = zeros(nx+2,ny+1);
U.Rui = zeros(nx+2,ny+1);
U.Lvi = zeros(nx+2,ny+1);
U.Rvi = zeros(nx+2,ny+1);

Q.RQj.Qrho = zeros(nx+1,ny+2);
Q.RQi.Qrho = zeros(nx+2,ny+1);
Q.RQj.Qrhou = zeros(nx+1,ny+2);
Q.RQi.Qrhou = zeros(nx+2,ny+1);
Q.RQj.Qrhov = zeros(nx+1,ny+2);
Q.RQi.Qrhov = zeros(nx+2,ny+1);
Q.RQj.Qrhoet = zeros(nx+1,ny+2);
Q.RQi.Qrhoet = zeros(nx+2,ny+1);
U.Luj =  zeros(nx+1,ny+2);
U.Ruj = zeros(nx+1,ny+2);
U.Lvj =zeros(nx+1,ny+2);
U.Rvj =zeros(nx+1,ny+2);

E.E1 = zeros(nx+2,ny+1);
E.E2 = zeros(nx+2,ny+1);
E.E3 = zeros(nx+2,ny+1);
E.E4 = zeros(nx+2,ny+1);

F.F1 = zeros(nx+1,ny+2);
F.F2 = zeros(nx+1,ny+2);
F.F3 = zeros(nx+1,ny+2);
F.F4 = zeros(nx+1,ny+2);

RHS.rho = zeros(nx+1,ny+1);
RHS.rhou = zeros(nx+1,ny+1);
RHS.rhov = zeros(nx+1,ny+1);
RHS.rhoet = zeros(nx+1,ny+1);

Q.rho = zeros(nx+1, ny+1);
Q.rhou = zeros(nx+1, ny+1);
Q.rhov = zeros(nx+1, ny+1);
Q.rhoet = zeros(nx+1, ny+1);
U.u = zeros(nx+1,ny+1);
U.v = zeros(nx+1,ny+1);
U.V_eta = zeros(nx+1,ny+1);
P = zeros(nx+1,ny+1);
T = zeros(nx+1,ny+1);
end


function [Q,U] =
Intialize(Q,U,P,T,P_ref,V_ref,R,T_ref,gamma,nx,ny,S_eta,S_eta_x,S_eta_y,yxi,xxi,xu,yu
)
```

```
%Constructing the normalizing Q state vector
for i = 1:nx+1
for j = 1:ny+1
Q.ref.rho(i,j) = P_ref/(R*T_ref);
Q.ref.rhou(i,j) = (P_ref/(R*T_ref))*V_ref;
Q.ref.rhov(i,j) = (P_ref/(R*T_ref))*V_ref;
Q.ref.rhoet(i,j) = (P_ref/(gamma-1)) + 0.5*(P_ref/(R*T_ref))*(V_ref^2);
end
end

% Initialize flow


% 1. Constant Inflow
U.u(1,:) = V_ref;
U.v(1,:) = 0.0;
P(1,:) = P_ref;
T(1,:) = T_ref;
Q.rho(1,:) = P_ref/(R*T_ref);
Q.rhou(1,:) = U.u(1,:).*P_ref/(R*T_ref);
Q.rhov(1,:) = U.v(1,:).*P_ref/(R*T_ref);
Q.rhoet(1,:) = (P_ref/(gamma-1)) + 0.5.*Q.rho(1,:).*(U.u(1,:).^2 + U.v(1,:).^2);


% 2. All internal gridpoints 2,2 to nx,ny
for i=2:nx
    for j =2:ny
        U.u(i,j) = V_ref;
        U.v(i,j) = 0;
        P(i,j) = P_ref;
        T(i,j) = T_ref;
        Q.rho(i,j) = P(i,j)/(R*T(i,j));
        Q.rhou(i,j) = U.u(i,j)*P(i,j)/(R*T(i,j));
        Q.rhov(i,j) = U.v(i,j)*P(i,j)/(R*T(i,j));
        Q.rhoet(i,j) = (P(i,j)/(gamma-1)) + 0.5*Q.rho(i,j)*(U.u(i,j)^2 + U.v(i,j)^2);
    end
end

% 3. For Bottom Wall
for i=1:nx+1
uv0 = ((1/(S_eta(i,2)))*[S_eta_y(i,2), -S_eta_x(i,2); S_eta_x(i,2),
S_eta_y(i,2)])\((1/(S_eta(i,2)))*[S_eta_y(i,2)*U.u(i,2)-S_eta_x(i,2)*U.v(i,2);-
S_eta_x(i,2)*U.u(i,2)-S_eta_y(i,2)*U.v(i,2)]);
U.u(i,1) = uv0(1);
U.v(i,1) = uv0(2);
U.V_eta(i,1) = ((-yxi(i,1))/(S_eta(i,1))) * U.u(i,1) + (xxi(i,1)/S_eta(i,1)) *
U.v(i,1);
U.V_eta(i,2) = ((-yxi(i,2))/S_eta(i,2)) * U.u(i,2) + (xxi(i,2)/S_eta(i,2)) *
U.v(i,2);
P(i,1)       =  P(i,2);
T(i,1)       =  T(i,2) ;

Q.rho(i,1) = P(i,1)/(R*T(i,1));
Q.rhou(i,1) = U.u(i,1)*P(i,1)/(R*T(i,1));
Q.rhov(i,1) = U.v(i,1)*P(i,1)/(R*T(i,1));
```

```
Q.rhoet(i,1) = (P(i,1)/(gamma-1)) + 0.5*Q.rho(i,1)*(U.u(i,1)^2 + U.v(i,1)^2);

end

% 4. For Top Wall
for i=1:nx+1
uvny2 = ((1/(S_eta(i,end)))*[S_eta_y(i,end), -S_eta_x(i,end); S_eta_x(i,end),
S_eta_y(i,end)])\((1/(S_eta(i,end-1)))*[S_eta_y(i,end-1)*U.u(i,end-1)-S_eta_x(i,end-
1)*U.v(i,end-1);-S_eta_x(i,end-1)*U.u(i,end-1)-S_eta_y(i,end-1)*U.v(i,end-1)]);
U.u(i,end) = uvny2(1);
U.v(i,end) = uvny2(2);
U.V_eta(i,end-1) = ((-yxi(i,end-1))/(S_eta(i,end-1))) * U.u(i,end-1) + (xxi(i,end-
1)/S_eta(i,end-1)) * U.v(i,end-1);
U.V_eta(i,end) = ((-yxi(i,end))/S_eta(i,end)) * U.u(i,end) +
(xxi(i,end)/S_eta(i,end)) * U.v(i,end);
P(i,end)       =  P(i,end-1);
T(i,end)       =  T(i,end-1) ;

Q.rho(i,end) = P(i,end)/(R*T(i,end));
Q.rhou(i,end) = U.u(i,end)*P(i,end)/(R*T(i,end));
Q.rhov(i,end) = U.v(i,end)*P(i,end)/(R*T(i,end));
Q.rhoet(i,end) = (P(i,end)/(gamma-1)) + 0.5*Q.rho(i,end)*(U.u(i,end)^2 +
U.v(i,end)^2);

end

% 5. Outflow

for j = 1:ny+1
Q.rho(nx+1,j) = P(nx,j)/(R*T(nx,j));
Q.rhou(nx+1,j) = U.u(nx,j)*P(nx,j)/(R*T(nx,j));
Q.rhov(nx+1,j) = U.v(nx,j)*P(nx,j)/(R*T(nx,j));
Q.rhoet(nx+1,j) = (P(nx,j)/(gamma-1)) + 0.5*Q.rho(nx,j)*(U.u(nx,j)^2 + U.v(nx,j)^2);
end


end


function [dt] = local_time(CFLmax, Q,nx,ny,yeta,xeta,yxi,xxi,Vc,gamma,R,x2d,y2d)

dt = zeros(nx+1,ny+1);
xi_x =zeros(nx+1, ny+1);
xi_y=zeros(nx+1, ny+1);
eta_x=zeros(nx+1, ny+1);
eta_y=zeros(nx+1, ny+1);
u_xi = zeros(nx+1, ny+1);
v_eta = zeros(nx+1, ny+1);
c_center = zeros(nx+1,ny+1);
P = zeros(nx+1,ny+1);
T = zeros(nx+1,ny+1);
rholambda_xi = zeros(nx+1,ny+1);
rholambda_eta = zeros(nx+1,ny+1);

% Calculate u_xi , v_eta
```

```
for i = 2:nx
for j = 2:ny
Sxix = 0.5*(yeta(i,j) + yeta(i+1,j));
Sxiy = 0.5*((-xeta(i,j)) + (-xeta(i+1,j)));
Setax = 0.5*((-yxi(i,j)) + (-yxi(i,j+1)));
Setay = 0.5*(xxi(i,j) + xxi(i,j+1));
Sxi = sqrt(Sxix^2 + Sxiy^2);
Seta = sqrt(Setax^2 + Setay^2);
xi_x(i,j) = Sxix/Vc(i,j);
xi_y(i,j) = Sxiy/Vc(i,j);
eta_x(i,j) = Setax/Vc(i,j);
eta_y(i,j) = Setay/Vc(i,j);
u_xi(i,j) =  (Sxix * (Q.rhou(i,j) / Q.rho(i,j)) + Sxiy * (Q.rhov(i,j) /
Q.rho(i,j)))/Sxi ;
v_eta(i,j) =  (Setax* (Q.rhou(i,j) / Q.rho(i,j)) + Setay * (Q.rhov(i,j) /
Q.rho(i,j)))/Seta ;
end
end

for i =2:nx
for j = 2:ny
P(i,j) = (gamma -1) * ( Q.rhoet(i,j) - 0.5 * (Q.rhou(i,j)^2 +
Q.rhov(i,j)^2)/Q.rho(i,j) );
T(i,j) = P(i,j)/(Q.rho(i,j)*R);
c_center(i,j) = sqrt(gamma*R*T(i,j));
rholambda_xi(i,j) = abs(u_xi(i,j)) + c_center(i,j);
rholambda_eta(i,j) = abs(v_eta(i,j)) + c_center(i,j);
dt(i,j) = CFLmax * (min((1/(rholambda_xi(i,j)*(sqrt(((xi_x(i,j)^2) +
(xi_y(i,j)^2))))))),(1/(rholambda_eta(i,j)*(sqrt(((eta_x(i,j)^2) +
(eta_y(i,j)^2)))))));
end
end

end

function [E,F] = FLUX(Q,gamma,nx,ny,S_eta_x,S_eta_y,S_xi_x, S_xi_y, S_xi, S_eta)
%% Construct Fluxes E and F at the cell faces. Construct Q at cell faces first.

% 1. Construct Q in top and bottom faces - wall.
%    Q@face will be average of Q in adjacent cell centeres for walls.


% 2. Flux on interior cell faces - upwind

% xi face
for j = 2:ny
    for i = 3:nx
        Q.Qi.Qrho(i,j) = Q.rho(i-1,j);
        Q.Qi.Qrhou(i,j) = Q.rhou(i-1,j);
        Q.Qi.Qrhov(i,j) = Q.rhov(i-1,j);
        Q.Qi.Qrhoet(i,j) = Q.rhoet(i-1,j);
    end
end
```

```
% Fill i=2 and nx+1 xi faces by averaging Q

for j=2:ny
    Q.Qi.Qrho(2,j) = 0.5*(Q.rho(1,j)+Q.rho(2,j));
    Q.Qi.Qrhou(2,j) = 0.5*(Q.rhou(1,j)+Q.rhou(2,j));
    Q.Qi.Qrhov(2,j) = 0.5*(Q.rhov(1,j)+Q.rhov(2,j));
    Q.Qi.Qrhoet(2,j) = 0.5*(Q.rhoet(1,j)+Q.rhoet(2,j));

    Q.Qi.Qrho(nx+1,j) = 0.5*(Q.rho(nx,j)+Q.rho(nx+1,j));
    Q.Qi.Qrhou(nx+1,j) = 0.5*(Q.rhou(nx,j)+Q.rhou(nx+1,j));
    Q.Qi.Qrhov(nx+1,j) = 0.5*(Q.rhov(nx,j)+Q.rhov(nx+1,j));
    Q.Qi.Qrhoet(nx+1,j) = 0.5*(Q.rhoet(nx,j)+Q.rhoet(nx+1,j));
end


% Eta face

for i = 1:17
    for j = 3:ny
    Q.Qj.Qrho(i,j) = Q.rho(i,j-1);
    Q.Qj.Qrhou(i,j) = Q.rhou(i,j-1);
    Q.Qj.Qrhov(i,j) = Q.rhov(i,j-1);
    Q.Qj.Qrhoet(i,j) = Q.rhoet(i,j-1);
    end
end

for i = 18:nx+1       %18 to 25  ramp down
   for j = 3:ny
    Q.Qj.Qrho(i,j) = Q.rho(i,j);
    Q.Qj.Qrhou(i,j) = Q.rhou(i,j);
    Q.Qj.Qrhov(i,j) = Q.rhov(i,j);
    Q.Qj.Qrhoet(i,j) = Q.rhoet(i,j);
    end
end

% Fill j = 2 and ny+1 by averaging

for i = 1:nx+1
Q.Qj.Qrho(i,2) = 0.5*(Q.rho(i,1) + Q.rho(i,2)) ;
Q.Qj.Qrhou(i,2) = 0.5*(Q.rhou(i,1) + Q.rhou(i,2)) ;
Q.Qj.Qrhov(i,2) = 0.5*(Q.rhov(i,1) + Q.rhov(i,2)) ;
Q.Qj.Qrhoet(i,2) = 0.5*(Q.rhoet(i,1) + Q.rhoet(i,2)) ;

Q.Qj.Qrho(i,ny+1) = 0.5*(Q.rho(i,ny) + Q.rho(i,ny+1)) ;
Q.Qj.Qrhou(i,ny+1) = 0.5*(Q.rhou(i,ny) + Q.rhou(i,ny+1)) ;
Q.Qj.Qrhov(i,ny+1) = 0.5*(Q.rhov(i,ny) + Q.rhov(i,ny+1)) ;
Q.Qj.Qrhoet(i,ny+1) = 0.5*(Q.rhoet(i,ny) + Q.rhoet(i,ny+1)) ;
end


%% E flux on face, Note: Divide the below expressions with S_xi to get the actual
expressions that were used in class.
U_xi =zeros(nx+2,ny+1);
P_xi = zeros(nx+2,ny+1);
E.E1 = zeros(nx+2,ny+1);
```

```
E.E2 = zeros(nx+2,ny+1);
E.E3 = zeros(nx+2,ny+1);
E.E4 = zeros(nx+2,ny+1);

for j = 2:ny
for i = 2:nx+1
U_xi(i,j)= (S_xi_x(i,j)*(Q.Qi.Qrhou(i,j)/Q.Qi.Qrho(i,j)) +
S_xi_y(i,j)*(Q.Qi.Qrhov(i,j)/Q.Qi.Qrho(i,j)))/S_xi(i,j);
P_xi(i,j)= (gamma-1)*(Q.Qi.Qrhoet(i,j) - (0.5*(Q.Qi.Qrhou(i,j)^2 +
Q.Qi.Qrhov(i,j)^2)/Q.Qi.Qrho(i,j)));
E.E1(i,j) = Q.Qi.Qrho(i,j) * U_xi(i,j);
E.E2(i,j) = (E.E1(i,j) * Q.Qi.Qrhou(i,j)/Q.Qi.Qrho(i,j)) +
P_xi(i,j)*S_xi_x(i,j)/S_xi(i,j);
E.E3(i,j) = (E.E1(i,j) * Q.Qi.Qrhov(i,j)/Q.Qi.Qrho(i,j)) +
P_xi(i,j)*S_xi_y(i,j)/S_xi(i,j);
E.E4(i,j) = Q.Qi.Qrhoet(i,j)*U_xi(i,j) + P_xi(i,j)*U_xi(i,j);
end
end


%% F flux on face, Note: Divide the below expressions with S_eta to get the actual
expressions that were used in class.

V_eta =zeros(nx+1,ny+2);
P_eta = zeros(nx+1,ny+2);
F.F1 = zeros(nx+1,ny+2);
F.F2 = zeros(nx+1,ny+2);
F.F3 = zeros(nx+1,ny+2);
F.F4 = zeros(nx+1,ny+2);

for i = 1:nx+1
for j = 2:ny+1
V_eta(i,j)= (S_eta_x(i,j)*(Q.Qj.Qrhou(i,j)/Q.Qj.Qrho(i,j)) +
S_eta_y(i,j)*(Q.Qj.Qrhov(i,j)/Q.Qj.Qrho(i,j)))/S_eta(i,j);
P_eta(i,j)= (gamma-1)*(Q.Qj.Qrhoet(i,j) - (0.5*(Q.Qj.Qrhou(i,j)^2 +
Q.Qj.Qrhov(i,j)^2)/Q.Qj.Qrho(i,j)));
F.F1(i,j) = Q.Qj.Qrho(i,j) * V_eta(i,j);
F.F2(i,j) = (F.F1(i,j) * Q.Qj.Qrhou(i,j)/Q.Qj.Qrho(i,j)) +
P_eta(i,j)*S_eta_x(i,j)/S_eta(i,j);
F.F3(i,j) = (F.F1(i,j) * Q.Qj.Qrhov(i,j)/Q.Qj.Qrho(i,j)) +
P_eta(i,j)*S_eta_y(i,j)/S_eta(i,j);
F.F4(i,j) = Q.Qj.Qrhoet(i,j)*V_eta(i,j) + P_eta(i,j)*V_eta(i,j);
end
end

% Flux Difference

E.dE1 = zeros(nx+1,ny+1);
E.dE2 = zeros(nx+1,ny+1);
E.dE3 = zeros(nx+1,ny+1);
E.dE4 = zeros(nx+1,ny+1);

for j=2:ny
    for i=2:nx
   E.dE1(i,j) = E.E1(i+1,j)*S_xi(i+1,j) - E.E1(i,j)*S_xi(i,j);
```

```
   E.dE2(i,j) = E.E2(i+1,j)*S_xi(i+1,j) - E.E2(i,j)*S_xi(i,j);
   E.dE3(i,j) = E.E3(i+1,j)*S_xi(i+1,j) - E.E3(i,j)*S_xi(i,j);
   E.dE4(i,j) = E.E4(i+1,j)*S_xi(i+1,j) - E.E4(i,j)*S_xi(i,j);
    end
end

F.dF1 = zeros(nx+1,ny+1);
F.dF2 = zeros(nx+1,ny+1);
F.dF3 = zeros(nx+1,ny+1);
F.dF4 = zeros(nx+1,ny+1);

for i=2:nx
    for j=2:ny
     F.dF1(i,j) =  F.F1(i,j+1)*S_eta(i,j+1) - F.F1(i,j)*S_eta(i,j);
     F.dF2(i,j) =  F.F2(i,j+1)*S_eta(i,j+1) - F.F2(i,j)*S_eta(i,j);
     F.dF3(i,j) =  F.F3(i,j+1)*S_eta(i,j+1) - F.F3(i,j)*S_eta(i,j);
     F.dF4(i,j) =  F.F4(i,j+1)*S_eta(i,j+1) - F.F4(i,j)*S_eta(i,j);
    end
end

end


function [Q, U] = MUSCL2(Q,U,nx,ny)

% MUSCL 2nd order accurate epsilon = 1 and k =-1 without flux limiter i.e phi = 1.
epsilon = 1; k = -1;

% xi face

for j = 2:ny
    for i = 3:nx
        Q.LQi.Qrho(i,j)   = Q.rho(i-1,j)  + (epsilon/4)*( (1-k)*(Q.rho(i-1,j)   -
Q.rho(i-2,j))   + (1+k)*(Q.rho(i,j)   - Q.rho(i-1,j)) );
        Q.LQi.Qrhou(i,j)  = Q.rhou(i-1,j) + (epsilon/4)*( (1-k)*(Q.rhou(i-1,j)  -
Q.rhou(i-2,j))  + (1+k)*(Q.rhou(i,j)  - Q.rhou(i-1,j)) );
        Q.LQi.Qrhov(i,j)  = Q.rhov(i-1,j) + (epsilon/4)*( (1-k)*(Q.rhov(i-1,j)  -
Q.rhov(i-2,j))  + (1+k)*(Q.rhov(i,j)  - Q.rhov(i-1,j)) );
        Q.LQi.Qrhoet(i,j) = Q.rhoet(i-1,j)+ (epsilon/4)*( (1-k)*(Q.rhoet(i-1,j) -
Q.rhoet(i-2,j)) + (1+k)*(Q.rhoet(i,j) - Q.rhoet(i-1,j)) );

        Q.RQi.Qrho(i,j)    = Q.rho(i,j)   - (epsilon/4)*( (1+k)*((Q.rho(i,j)    -
Q.rho(i-1,j)))   + (1-k)*(Q.rho(i+1,j)  - Q.rho(i,j)) );
        Q.RQi.Qrhou(i,j)   = Q.rhou(i,j)  - (epsilon/4)*( (1+k)*((Q.rhou(i,j)   -
Q.rhou(i-1,j)))  + (1-k)*(Q.rhou(i+1,j) - Q.rhou(i,j)) );
        Q.RQi.Qrhov(i,j)   = Q.rhov(i,j)  - (epsilon/4)*( (1+k)*((Q.rhov(i,j)   -
Q.rhov(i-1,j)))  + (1-k)*(Q.rhov(i+1,j) - Q.rhov(i,j)) );
        Q.RQi.Qrhoet(i,j)  = Q.rhoet(i,j) - (epsilon/4)*( (1+k)*((Q.rhoet(i,j)  -
Q.rhoet(i-1,j))) + (1-k)*(Q.rhoet(i+1,j)- Q.rhoet(i,j)) );
    end
end

% Fill i=2 and nx+1 xi faces by averaging Q

for j=2:ny
```

```
    Q.LQi.Qrho(2,j) = 0.5*(Q.rho(1,j)+Q.rho(2,j));
    Q.LQi.Qrhou(2,j) = 0.5*(Q.rhou(1,j)+Q.rhou(2,j));
    Q.LQi.Qrhov(2,j) = 0.5*(Q.rhov(1,j)+Q.rhov(2,j));
    Q.LQi.Qrhoet(2,j) = 0.5*(Q.rhoet(1,j)+Q.rhoet(2,j));

    Q.LQi.Qrho(nx+1,j) = 0.5*(Q.rho(nx,j)+Q.rho(nx+1,j));
    Q.LQi.Qrhou(nx+1,j) = 0.5*(Q.rhou(nx,j)+Q.rhou(nx+1,j));
    Q.LQi.Qrhov(nx+1,j) = 0.5*(Q.rhov(nx,j)+Q.rhov(nx+1,j));
    Q.LQi.Qrhoet(nx+1,j) = 0.5*(Q.rhoet(nx,j)+Q.rhoet(nx+1,j));


    Q.RQi.Qrho(2,j) = 0.5*(Q.rho(1,j)+Q.rho(2,j));
    Q.RQi.Qrhou(2,j) = 0.5*(Q.rhou(1,j)+Q.rhou(2,j));
    Q.RQi.Qrhov(2,j) = 0.5*(Q.rhov(1,j)+Q.rhov(2,j));
    Q.RQi.Qrhoet(2,j) = 0.5*(Q.rhoet(1,j)+Q.rhoet(2,j));

    Q.RQi.Qrho(nx+1,j) = 0.5*(Q.rho(nx,j)+Q.rho(nx+1,j));
    Q.RQi.Qrhou(nx+1,j) = 0.5*(Q.rhou(nx,j)+Q.rhou(nx+1,j));
    Q.RQi.Qrhov(nx+1,j) = 0.5*(Q.rhov(nx,j)+Q.rhov(nx+1,j));
    Q.RQi.Qrhoet(nx+1,j) = 0.5*(Q.rhoet(nx,j)+Q.rhoet(nx+1,j));

end


for i = 1:nx+1
    for j = 3:ny
    Q.LQj.Qrho(i,j)    = Q.rho(i,j-1)  + (epsilon/4)*( (1-k)*(Q.rho(i,j-1)   -
Q.rho(i,j-2))   + (1+k)*(Q.rho(i,j)   - Q.rho(i,j-1)));
    Q.LQj.Qrhou(i,j)   = Q.rhou(i,j-1) + (epsilon/4)*( (1-k)*(Q.rhou(i,j-1)  -
Q.rhou(i,j-2))  + (1+k)*(Q.rhou(i,j)  - Q.rhou(i,j-1)));
    Q.LQj.Qrhov(i,j)   = Q.rhov(i,j-1) + (epsilon/4)*( (1-k)*(Q.rhov(i,j-1)  -
Q.rhov(i,j-2))  + (1+k)*(Q.rhov(i,j)  - Q.rhov(i,j-1)));
    Q.LQj.Qrhoet(i,j)  = Q.rhoet(i,j-1)+ (epsilon/4)*( (1-k)*(Q.rhoet(i,j-1) -
Q.rhoet(i,j-2)) + (1+k)*(Q.rhoet(i,j) - Q.rhoet(i,j-1)));

    Q.RQj.Qrho(i,j)    = Q.rho(i,j)   - (epsilon/4)*( (1+k)*(Q.rho(i,j)      -
Q.rho(i,j-1))   + (1-k)*(Q.rho(i,j+1)  - Q.rho(i,j)) ) ;
    Q.RQj.Qrhou(i,j)   = Q.rhou(i,j)  - (epsilon/4)*( (1+k)*(Q.rhou(i,j)     -
Q.rhou(i,j-1))  + (1-k)*(Q.rhou(i,j+1) - Q.rhou(i,j)) ) ;
    Q.RQj.Qrhov(i,j)   = Q.rhov(i,j)  - (epsilon/4)*( (1+k)*(Q.rhov(i,j)     -
Q.rhov(i,j-1))  + (1-k)*(Q.rhov(i,j+1) - Q.rhov(i,j)) ) ;
    Q.RQj.Qrhoet(i,j)  = Q.rhoet(i,j) - (epsilon/4)*( (1+k)*(Q.rhoet(i,j)    -
Q.rhoet(i,j-1)) + (1-k)*(Q.rhoet(i,j+1)- Q.rhoet(i,j)) ) ;
    end
end

% Fill j = 2 and ny+1 by averaging

for i = 1:nx+1
Q.LQj.Qrho(i,2) = 0.5*(Q.rho(i,1) + Q.rho(i,2)) ;
Q.LQj.Qrhou(i,2) = 0.5*(Q.rhou(i,1) + Q.rhou(i,2)) ;
Q.LQj.Qrhov(i,2) = 0.5*(Q.rhov(i,1) + Q.rhov(i,2)) ;
Q.LQj.Qrhoet(i,2) = 0.5*(Q.rhoet(i,1) + Q.rhoet(i,2)) ;
```

```
Q.LQj.Qrho(i,ny+1) = 0.5*(Q.rho(i,ny) + Q.rho(i,ny+1)) ;
Q.LQj.Qrhou(i,ny+1) = 0.5*(Q.rhou(i,ny) + Q.rhou(i,ny+1)) ;
Q.LQj.Qrhov(i,ny+1) = 0.5*(Q.rhov(i,ny) + Q.rhov(i,ny+1)) ;
Q.LQj.Qrhoet(i,ny+1) = 0.5*(Q.rhoet(i,ny) + Q.rhoet(i,ny+1)) ;

Q.RQj.Qrho(i,2) = 0.5*(Q.rho(i,1) + Q.rho(i,2)) ;
Q.RQj.Qrhou(i,2) = 0.5*(Q.rhou(i,1) + Q.rhou(i,2)) ;
Q.RQj.Qrhov(i,2) = 0.5*(Q.rhov(i,1) + Q.rhov(i,2)) ;
Q.RQj.Qrhoet(i,2) = 0.5*(Q.rhoet(i,1) + Q.rhoet(i,2)) ;

Q.RQj.Qrho(i,ny+1) = 0.5*(Q.rho(i,ny) + Q.rho(i,ny+1)) ;
Q.RQj.Qrhou(i,ny+1) = 0.5*(Q.rhou(i,ny) + Q.rhou(i,ny+1)) ;
Q.RQj.Qrhov(i,ny+1) = 0.5*(Q.rhov(i,ny) + Q.rhov(i,ny+1)) ;
Q.RQj.Qrhoet(i,ny+1) = 0.5*(Q.rhoet(i,ny) + Q.rhoet(i,ny+1)) ;
end

% u and v left and right states

% u and v on xi faces
for j=2:ny
    for i = 2:nx+1
        U.Lui(i,j) = Q.LQi.Qrhou(i,j)/Q.LQi.Qrho(i,j) ;
        U.Rui(i,j) = Q.RQi.Qrhou(i,j)/Q.RQi.Qrho(i,j) ;
        U.Lvi(i,j) = Q.LQi.Qrhov(i,j)/Q.LQi.Qrho(i,j) ;
        U.Rvi(i,j) = Q.RQi.Qrhov(i,j)/Q.RQi.Qrho(i,j) ;
    end
end


% u and v on eta faces

for i = 1:nx+1
    for j=2:ny+1
        U.Luj(i,j) = Q.LQj.Qrhou(i,j)/Q.LQj.Qrho(i,j) ;
        U.Ruj(i,j) = Q.RQj.Qrhou(i,j)/Q.RQj.Qrho(i,j) ;
        U.Lvj(i,j) = Q.LQj.Qrhov(i,j)/Q.LQj.Qrho(i,j) ;
        U.Rvj(i,j) = Q.RQj.Qrhov(i,j)/Q.RQj.Qrho(i,j) ;
    end
end


end


function [Q, U] = MUSCL3(Q,U,nx,ny)

% MUSCL 2nd order accurate epsilon = 1 and k =-1 with flux limiter.
epsilon = 1; k = -1;

% xi face

for j = 2:ny
    for i = 3:nx
        rL.rho   = (Q.rho(i,j)  -Q.rho(i-1,j)+0.00000001) /(Q.rho(i-1,j)    -
Q.rho(i-2,j)+0.00000001);
```

```
        rL.rhou  = (Q.rhou(i,j) -Q.rhou(i-1,j)+0.00000001)/(Q.rhou(i-1,j)   -
Q.rhou(i-2,j)+0.00000001);
        rL.rhov  = (Q.rhov(i,j) -Q.rhov(i-1,j)+0.00000001)/(Q.rhov(i-1,j)   -
Q.rhov(i-2,j)+0.00000001);
        rL.rhoet = (Q.rhoet(i,j)-Q.rhoet(i-1,j)+0.00000001)/(Q.rhoet(i-1,j) -
Q.rhoet(i-2,j)+0.00000001);

        Q.LQi.Qrho(i,j)   = Q.rho(i-1,j)  + (epsilon/4)*( (1-k)*(Q.rho(i-1,j)   -
Q.rho(i-2,j))*minmoid(rL.rho)     + (1+k)*(Q.rho(i,j)   - Q.rho(i-
1,j))*minmoid(1/rL.rho) );
        Q.LQi.Qrhou(i,j)  = Q.rhou(i-1,j) + (epsilon/4)*( (1-k)*(Q.rhou(i-1,j)  -
Q.rhou(i-2,j))*minmoid(rL.rhou)   + (1+k)*(Q.rhou(i,j)  - Q.rhou(i-
1,j))*minmoid(1/rL.rhou) );
        Q.LQi.Qrhov(i,j)  = Q.rhov(i-1,j) + (epsilon/4)*( (1-k)*(Q.rhov(i-1,j)  -
Q.rhov(i-2,j))*minmoid(rL.rhov)   + (1+k)*(Q.rhov(i,j)  - Q.rhov(i-
1,j))*minmoid(1/rL.rhov) );
        Q.LQi.Qrhoet(i,j) = Q.rhoet(i-1,j)+ (epsilon/4)*( (1-k)*(Q.rhoet(i-1,j) -
Q.rhoet(i-2,j))*minmoid(rL.rhoet) + (1+k)*(Q.rhoet(i,j) - Q.rhoet(i-
1,j))*minmoid(1/rL.rhoet) );

        rR.rho   = (Q.rho(i,j)  -Q.rho(i-1,j)+0.00000001) /(Q.rho(i+1,j)    -
Q.rho(i,j)+0.00000001);
        rR.rhou  = (Q.rhou(i,j) -Q.rhou(i-1,j)+0.00000001)/(Q.rhou(i+1,j)   -
Q.rhou(i,j)+0.00000001);
        rR.rhov  = (Q.rhov(i,j) -Q.rhov(i-1,j)+0.00000001)/(Q.rhov(i+1,j)   -
Q.rhov(i,j)+0.00000001);
        rR.rhoet = (Q.rhoet(i,j)-Q.rhoet(i-1,j)+0.00000001)/(Q.rhoet(i+1,j) -
Q.rhoet(i,j)+0.00000001);

        Q.RQi.Qrho(i,j)    = Q.rho(i,j)   - (epsilon/4)*( (1+k)*(Q.rho(i,j)    -
Q.rho(i-1,j))*minmoid(1/rR.rho)     + (1-k)*(Q.rho(i+1,j)  -
Q.rho(i,j))*minmoid(rR.rho) );
        Q.RQi.Qrhou(i,j)   = Q.rhou(i,j)  - (epsilon/4)*( (1+k)*(Q.rhou(i,j)   -
Q.rhou(i-1,j))*minmoid(1/rR.rhou)   + (1-k)*(Q.rhou(i+1,j) -
Q.rhou(i,j))*minmoid(rR.rho) );
        Q.RQi.Qrhov(i,j)   = Q.rhov(i,j)  - (epsilon/4)*( (1+k)*(Q.rhov(i,j)   -
Q.rhov(i-1,j))*minmoid(1/rR.rhov)   + (1-k)*(Q.rhov(i+1,j) -
Q.rhov(i,j))*minmoid(rR.rho) );
        Q.RQi.Qrhoet(i,j)  = Q.rhoet(i,j) - (epsilon/4)*( (1+k)*(Q.rhoet(i,j)  -
Q.rhoet(i-1,j))*minmoid(1/rR.rhoet) + (1-k)*(Q.rhoet(i+1,j)-
Q.rhoet(i,j))*minmoid(rR.rho) );
    end
end

% Fill i=2 and nx+1 xi faces by averaging Q

for j=2:ny

    Q.LQi.Qrho(2,j) = 0.5*(Q.rho(1,j)+Q.rho(2,j));
    Q.LQi.Qrhou(2,j) = 0.5*(Q.rhou(1,j)+Q.rhou(2,j));
    Q.LQi.Qrhov(2,j) = 0.5*(Q.rhov(1,j)+Q.rhov(2,j));
    Q.LQi.Qrhoet(2,j) = 0.5*(Q.rhoet(1,j)+Q.rhoet(2,j));

    Q.LQi.Qrho(nx+1,j) = 0.5*(Q.rho(nx,j)+Q.rho(nx+1,j));
    Q.LQi.Qrhou(nx+1,j) = 0.5*(Q.rhou(nx,j)+Q.rhou(nx+1,j));
```

```
    Q.LQi.Qrhov(nx+1,j) = 0.5*(Q.rhov(nx,j)+Q.rhov(nx+1,j));
    Q.LQi.Qrhoet(nx+1,j) = 0.5*(Q.rhoet(nx,j)+Q.rhoet(nx+1,j));


    Q.RQi.Qrho(2,j) = 0.5*(Q.rho(1,j)+Q.rho(2,j));
    Q.RQi.Qrhou(2,j) = 0.5*(Q.rhou(1,j)+Q.rhou(2,j));
    Q.RQi.Qrhov(2,j) = 0.5*(Q.rhov(1,j)+Q.rhov(2,j));
    Q.RQi.Qrhoet(2,j) = 0.5*(Q.rhoet(1,j)+Q.rhoet(2,j));

    Q.RQi.Qrho(nx+1,j) = 0.5*(Q.rho(nx,j)+Q.rho(nx+1,j));
    Q.RQi.Qrhou(nx+1,j) = 0.5*(Q.rhou(nx,j)+Q.rhou(nx+1,j));
    Q.RQi.Qrhov(nx+1,j) = 0.5*(Q.rhov(nx,j)+Q.rhov(nx+1,j));
    Q.RQi.Qrhoet(nx+1,j) = 0.5*(Q.rhoet(nx,j)+Q.rhoet(nx+1,j));

end


for i = 1:nx+1
    for j = 3:ny

        rL.rho   = (Q.rho(i,j)  -Q.rho(i,j-1)+0.00000001) /(Q.rho(i,j-1)    -
Q.rho(i,j-2)+0.00000001);
        rL.rhou  = (Q.rhou(i,j) -Q.rhou(i,j-1)+0.00000001)/(Q.rhou(i,j-1)   -
Q.rhou(i,j-2)+0.00000001);
        rL.rhov  = (Q.rhov(i,j) -Q.rhov(i,j-1)+0.00000001)/(Q.rhov(i,j-1)   -
Q.rhov(i,j-2)+0.00000001);
        rL.rhoet = (Q.rhoet(i,j)-Q.rhoet(i,j-1)+0.00000001)/(Q.rhoet(i,j-1) -
Q.rhoet(i,j-2)+0.00000001);

        Q.LQj.Qrho(i,j)    = Q.rho(i,j-1)  + (epsilon/4)*( (1-k)*(Q.rho(i,j-1)   -
Q.rho(i,j-2))*minmoid(rL.rho)     + (1+k)*(Q.rho(i,j)   - Q.rho(i,j-
1))*minmoid(1/rL.rho));
        Q.LQj.Qrhou(i,j)   = Q.rhou(i,j-1) + (epsilon/4)*( (1-k)*(Q.rhou(i,j-1)  -
Q.rhou(i,j-2))*minmoid(rL.rhou)   + (1+k)*(Q.rhou(i,j)  - Q.rhou(i,j-
1))*minmoid(1/rL.rhou));
        Q.LQj.Qrhov(i,j)   = Q.rhov(i,j-1) + (epsilon/4)*( (1-k)*(Q.rhov(i,j-1)  -
Q.rhov(i,j-2))*minmoid(rL.rhov)   + (1+k)*(Q.rhov(i,j)  - Q.rhov(i,j-
1))*minmoid(1/rL.rhov));
        Q.LQj.Qrhoet(i,j)  = Q.rhoet(i,j-1)+ (epsilon/4)*( (1-k)*(Q.rhoet(i,j-1) -
Q.rhoet(i,j-2))*minmoid(rL.rhoet) + (1+k)*(Q.rhoet(i,j) - Q.rhoet(i,j-
1))*minmoid(1/rL.rhoet));

        rR.rho   = (Q.rho(i,j)  -Q.rho(i,j-1)+0.00000001) /(Q.rho(i,j+1)    -
Q.rho(i,j)+0.00000001);
        rR.rhou  = (Q.rhou(i,j) -Q.rhou(i,j-1)+0.00000001)/(Q.rhou(i,j+1)   -
Q.rhou(i,j)+0.00000001);
        rR.rhov  = (Q.rhov(i,j) -Q.rhov(i,j-1)+0.00000001)/(Q.rhov(i,j+1)   -
Q.rhov(i,j)+0.00000001);
        rR.rhoet = (Q.rhoet(i,j)-Q.rhoet(i,j-1)+0.00000001)/(Q.rhoet(i,j+1) -
Q.rhoet(i,j)+0.00000001);

        Q.RQj.Qrho(i,j)    = Q.rho(i,j)   - (epsilon/4)*( (1+k)*(Q.rho(i,j)      -
Q.rho(i,j-1))*minmoid(1/rR.rho)     + (1-k)*(Q.rho(i,j+1)  -
Q.rho(i,j))*minmoid(rR.rho) ) ;
```

```
        Q.RQj.Qrhou(i,j)   = Q.rhou(i,j)  - (epsilon/4)*( (1+k)*(Q.rhou(i,j)     -
Q.rhou(i,j-1))*minmoid(1/rR.rhou)   + (1-k)*(Q.rhou(i,j+1) -
Q.rhou(i,j))*minmoid(rR.rhou) ) ;
        Q.RQj.Qrhov(i,j)   = Q.rhov(i,j)  - (epsilon/4)*( (1+k)*(Q.rhov(i,j)     -
Q.rhov(i,j-1))*minmoid(1/rR.rhov)   + (1-k)*(Q.rhov(i,j+1) -
Q.rhov(i,j))*minmoid(rR.rhov) ) ;
        Q.RQj.Qrhoet(i,j)  = Q.rhoet(i,j) - (epsilon/4)*( (1+k)*(Q.rhoet(i,j)    -
Q.rhoet(i,j-1))*minmoid(1/rR.rhoet) + (1-k)*(Q.rhoet(i,j+1)-
Q.rhoet(i,j))*minmoid(rR.rhoet) ) ;
    end
end

% Fill j = 2 and ny+1 by averaging

for i = 1:nx+1
Q.LQj.Qrho(i,2) = 0.5*(Q.rho(i,1) + Q.rho(i,2)) ;
Q.LQj.Qrhou(i,2) = 0.5*(Q.rhou(i,1) + Q.rhou(i,2)) ;
Q.LQj.Qrhov(i,2) = 0.5*(Q.rhov(i,1) + Q.rhov(i,2)) ;
Q.LQj.Qrhoet(i,2) = 0.5*(Q.rhoet(i,1) + Q.rhoet(i,2)) ;

Q.LQj.Qrho(i,ny+1) = 0.5*(Q.rho(i,ny) + Q.rho(i,ny+1)) ;
Q.LQj.Qrhou(i,ny+1) = 0.5*(Q.rhou(i,ny) + Q.rhou(i,ny+1)) ;
Q.LQj.Qrhov(i,ny+1) = 0.5*(Q.rhov(i,ny) + Q.rhov(i,ny+1)) ;
Q.LQj.Qrhoet(i,ny+1) = 0.5*(Q.rhoet(i,ny) + Q.rhoet(i,ny+1)) ;

Q.RQj.Qrho(i,2) = 0.5*(Q.rho(i,1) + Q.rho(i,2)) ;
Q.RQj.Qrhou(i,2) = 0.5*(Q.rhou(i,1) + Q.rhou(i,2)) ;
Q.RQj.Qrhov(i,2) = 0.5*(Q.rhov(i,1) + Q.rhov(i,2)) ;
Q.RQj.Qrhoet(i,2) = 0.5*(Q.rhoet(i,1) + Q.rhoet(i,2)) ;

Q.RQj.Qrho(i,ny+1) = 0.5*(Q.rho(i,ny) + Q.rho(i,ny+1)) ;
Q.RQj.Qrhou(i,ny+1) = 0.5*(Q.rhou(i,ny) + Q.rhou(i,ny+1)) ;
Q.RQj.Qrhov(i,ny+1) = 0.5*(Q.rhov(i,ny) + Q.rhov(i,ny+1)) ;
Q.RQj.Qrhoet(i,ny+1) = 0.5*(Q.rhoet(i,ny) + Q.rhoet(i,ny+1)) ;
end

% u and v left and right states

% u and v on xi faces
for j=2:ny
    for i = 2:nx+1
        U.Lui(i,j) = Q.LQi.Qrhou(i,j)/Q.LQi.Qrho(i,j) ;
        U.Rui(i,j) = Q.RQi.Qrhou(i,j)/Q.RQi.Qrho(i,j) ;
        U.Lvi(i,j) = Q.LQi.Qrhov(i,j)/Q.LQi.Qrho(i,j) ;
        U.Rvi(i,j) = Q.RQi.Qrhov(i,j)/Q.RQi.Qrho(i,j) ;
    end
end


% u and v on eta faces

for i = 1:nx+1
    for j=2:ny+1
        U.Luj(i,j) = Q.LQj.Qrhou(i,j)/Q.LQj.Qrho(i,j) ;
        U.Ruj(i,j) = Q.RQj.Qrhou(i,j)/Q.RQj.Qrho(i,j) ;
```

```
        U.Lvj(i,j) = Q.LQj.Qrhov(i,j)/Q.LQj.Qrho(i,j) ;
        U.Rvj(i,j) = Q.RQj.Qrhov(i,j)/Q.RQj.Qrho(i,j) ;
    end
end


end


function [Q, U] = MUSCL3(Q,U,nx,ny)

% MUSCL 2nd order accurate epsilon = 1 and k =-1 with flux limiter.
epsilon = 1; k = -1;

% xi face

for j = 2:ny
    for i = 3:nx
        rL.rho   = (Q.rho(i,j)  -Q.rho(i-1,j)+0.00000001) /(Q.rho(i-1,j)    -
Q.rho(i-2,j)+0.00000001);
        rL.rhou  = (Q.rhou(i,j) -Q.rhou(i-1,j)+0.00000001)/(Q.rhou(i-1,j)   -
Q.rhou(i-2,j)+0.00000001);
        rL.rhov  = (Q.rhov(i,j) -Q.rhov(i-1,j)+0.00000001)/(Q.rhov(i-1,j)   -
Q.rhov(i-2,j)+0.00000001);
        rL.rhoet = (Q.rhoet(i,j)-Q.rhoet(i-1,j)+0.00000001)/(Q.rhoet(i-1,j) -
Q.rhoet(i-2,j)+0.00000001);

        Q.LQi.Qrho(i,j)   = Q.rho(i-1,j)  + (epsilon/4)*( (1-k)*(Q.rho(i-1,j)   -
Q.rho(i-2,j))*minmoid(rL.rho)     + (1+k)*(Q.rho(i,j)   - Q.rho(i-
1,j))*minmoid(1/rL.rho) );
        Q.LQi.Qrhou(i,j)  = Q.rhou(i-1,j) + (epsilon/4)*( (1-k)*(Q.rhou(i-1,j)  -
Q.rhou(i-2,j))*minmoid(rL.rhou)   + (1+k)*(Q.rhou(i,j)  - Q.rhou(i-
1,j))*minmoid(1/rL.rhou) );
        Q.LQi.Qrhov(i,j)  = Q.rhov(i-1,j) + (epsilon/4)*( (1-k)*(Q.rhov(i-1,j)  -
Q.rhov(i-2,j))*minmoid(rL.rhov)   + (1+k)*(Q.rhov(i,j)  - Q.rhov(i-
1,j))*minmoid(1/rL.rhov) );
        Q.LQi.Qrhoet(i,j) = Q.rhoet(i-1,j)+ (epsilon/4)*( (1-k)*(Q.rhoet(i-1,j) -
Q.rhoet(i-2,j))*minmoid(rL.rhoet) + (1+k)*(Q.rhoet(i,j) - Q.rhoet(i-
1,j))*minmoid(1/rL.rhoet) );

        rR.rho   = (Q.rho(i,j)  -Q.rho(i-1,j)+0.00000001) /(Q.rho(i+1,j)    -
Q.rho(i,j)+0.00000001);
        rR.rhou  = (Q.rhou(i,j) -Q.rhou(i-1,j)+0.00000001)/(Q.rhou(i+1,j)   -
Q.rhou(i,j)+0.00000001);
        rR.rhov  = (Q.rhov(i,j) -Q.rhov(i-1,j)+0.00000001)/(Q.rhov(i+1,j)   -
Q.rhov(i,j)+0.00000001);
        rR.rhoet = (Q.rhoet(i,j)-Q.rhoet(i-1,j)+0.00000001)/(Q.rhoet(i+1,j) -
Q.rhoet(i,j)+0.00000001);

        Q.RQi.Qrho(i,j)    = Q.rho(i,j)   - (epsilon/4)*( (1+k)*(Q.rho(i,j)    -
Q.rho(i-1,j))*minmoid(1/rR.rho)     + (1-k)*(Q.rho(i+1,j)  -
Q.rho(i,j))*minmoid(rR.rho) );
        Q.RQi.Qrhou(i,j)   = Q.rhou(i,j)  - (epsilon/4)*( (1+k)*(Q.rhou(i,j)   -
Q.rhou(i-1,j))*minmoid(1/rR.rhou)   + (1-k)*(Q.rhou(i+1,j) -
Q.rhou(i,j))*minmoid(rR.rho) );
```

```
        Q.RQi.Qrhov(i,j)   = Q.rhov(i,j)  - (epsilon/4)*( (1+k)*(Q.rhov(i,j)   -
Q.rhov(i-1,j))*minmoid(1/rR.rhov)   + (1-k)*(Q.rhov(i+1,j) -
Q.rhov(i,j))*minmoid(rR.rho) );
        Q.RQi.Qrhoet(i,j)  = Q.rhoet(i,j) - (epsilon/4)*( (1+k)*(Q.rhoet(i,j)  -
Q.rhoet(i-1,j))*minmoid(1/rR.rhoet) + (1-k)*(Q.rhoet(i+1,j)-
Q.rhoet(i,j))*minmoid(rR.rho) );
    end
end

% Fill i=2 and nx+1 xi faces by averaging Q

for j=2:ny

    Q.LQi.Qrho(2,j) = 0.5*(Q.rho(1,j)+Q.rho(2,j));
    Q.LQi.Qrhou(2,j) = 0.5*(Q.rhou(1,j)+Q.rhou(2,j));
    Q.LQi.Qrhov(2,j) = 0.5*(Q.rhov(1,j)+Q.rhov(2,j));
    Q.LQi.Qrhoet(2,j) = 0.5*(Q.rhoet(1,j)+Q.rhoet(2,j));

    Q.LQi.Qrho(nx+1,j) = 0.5*(Q.rho(nx,j)+Q.rho(nx+1,j));
    Q.LQi.Qrhou(nx+1,j) = 0.5*(Q.rhou(nx,j)+Q.rhou(nx+1,j));
    Q.LQi.Qrhov(nx+1,j) = 0.5*(Q.rhov(nx,j)+Q.rhov(nx+1,j));
    Q.LQi.Qrhoet(nx+1,j) = 0.5*(Q.rhoet(nx,j)+Q.rhoet(nx+1,j));


    Q.RQi.Qrho(2,j) = 0.5*(Q.rho(1,j)+Q.rho(2,j));
    Q.RQi.Qrhou(2,j) = 0.5*(Q.rhou(1,j)+Q.rhou(2,j));
    Q.RQi.Qrhov(2,j) = 0.5*(Q.rhov(1,j)+Q.rhov(2,j));
    Q.RQi.Qrhoet(2,j) = 0.5*(Q.rhoet(1,j)+Q.rhoet(2,j));

    Q.RQi.Qrho(nx+1,j) = 0.5*(Q.rho(nx,j)+Q.rho(nx+1,j));
    Q.RQi.Qrhou(nx+1,j) = 0.5*(Q.rhou(nx,j)+Q.rhou(nx+1,j));
    Q.RQi.Qrhov(nx+1,j) = 0.5*(Q.rhov(nx,j)+Q.rhov(nx+1,j));
    Q.RQi.Qrhoet(nx+1,j) = 0.5*(Q.rhoet(nx,j)+Q.rhoet(nx+1,j));

end


for i = 1:nx+1
    for j = 3:ny

        rL.rho   = (Q.rho(i,j)  -Q.rho(i,j-1)+0.00000001) /(Q.rho(i,j-1)    -
Q.rho(i,j-2)+0.00000001);
        rL.rhou  = (Q.rhou(i,j) -Q.rhou(i,j-1)+0.00000001)/(Q.rhou(i,j-1)   -
Q.rhou(i,j-2)+0.00000001);
        rL.rhov  = (Q.rhov(i,j) -Q.rhov(i,j-1)+0.00000001)/(Q.rhov(i,j-1)   -
Q.rhov(i,j-2)+0.00000001);
        rL.rhoet = (Q.rhoet(i,j)-Q.rhoet(i,j-1)+0.00000001)/(Q.rhoet(i,j-1) -
Q.rhoet(i,j-2)+0.00000001);

        Q.LQj.Qrho(i,j)    = Q.rho(i,j-1)  + (epsilon/4)*( (1-k)*(Q.rho(i,j-1)   -
Q.rho(i,j-2))*minmoid(rL.rho)     + (1+k)*(Q.rho(i,j)   - Q.rho(i,j-
1))*minmoid(1/rL.rho));
        Q.LQj.Qrhou(i,j)   = Q.rhou(i,j-1) + (epsilon/4)*( (1-k)*(Q.rhou(i,j-1)  -
Q.rhou(i,j-2))*minmoid(rL.rhou)   + (1+k)*(Q.rhou(i,j)  - Q.rhou(i,j-
1))*minmoid(1/rL.rhou));
```

```
        Q.LQj.Qrhov(i,j)   = Q.rhov(i,j-1) + (epsilon/4)*( (1-k)*(Q.rhov(i,j-1)  -
Q.rhov(i,j-2))*minmoid(rL.rhov)   + (1+k)*(Q.rhov(i,j)  - Q.rhov(i,j-
1))*minmoid(1/rL.rhov));
        Q.LQj.Qrhoet(i,j)  = Q.rhoet(i,j-1)+ (epsilon/4)*( (1-k)*(Q.rhoet(i,j-1) -
Q.rhoet(i,j-2))*minmoid(rL.rhoet) + (1+k)*(Q.rhoet(i,j) - Q.rhoet(i,j-
1))*minmoid(1/rL.rhoet));

        rR.rho   = (Q.rho(i,j)  -Q.rho(i,j-1)+0.00000001) /(Q.rho(i,j+1)    -
Q.rho(i,j)+0.00000001);
        rR.rhou  = (Q.rhou(i,j) -Q.rhou(i,j-1)+0.00000001)/(Q.rhou(i,j+1)   -
Q.rhou(i,j)+0.00000001);
        rR.rhov  = (Q.rhov(i,j) -Q.rhov(i,j-1)+0.00000001)/(Q.rhov(i,j+1)   -
Q.rhov(i,j)+0.00000001);
        rR.rhoet = (Q.rhoet(i,j)-Q.rhoet(i,j-1)+0.00000001)/(Q.rhoet(i,j+1) -
Q.rhoet(i,j)+0.00000001);

        Q.RQj.Qrho(i,j)    = Q.rho(i,j)   - (epsilon/4)*( (1+k)*(Q.rho(i,j)      -
Q.rho(i,j-1))*minmoid(1/rR.rho)     + (1-k)*(Q.rho(i,j+1)  -
Q.rho(i,j))*minmoid(rR.rho) ) ;
        Q.RQj.Qrhou(i,j)   = Q.rhou(i,j)  - (epsilon/4)*( (1+k)*(Q.rhou(i,j)     -
Q.rhou(i,j-1))*minmoid(1/rR.rhou)   + (1-k)*(Q.rhou(i,j+1) -
Q.rhou(i,j))*minmoid(rR.rhou) ) ;
        Q.RQj.Qrhov(i,j)   = Q.rhov(i,j)  - (epsilon/4)*( (1+k)*(Q.rhov(i,j)     -
Q.rhov(i,j-1))*minmoid(1/rR.rhov)   + (1-k)*(Q.rhov(i,j+1) -
Q.rhov(i,j))*minmoid(rR.rhov) ) ;
        Q.RQj.Qrhoet(i,j)  = Q.rhoet(i,j) - (epsilon/4)*( (1+k)*(Q.rhoet(i,j)    -
Q.rhoet(i,j-1))*minmoid(1/rR.rhoet) + (1-k)*(Q.rhoet(i,j+1)-
Q.rhoet(i,j))*minmoid(rR.rhoet) ) ;
    end
end

% Fill j = 2 and ny+1 by averaging

for i = 1:nx+1
Q.LQj.Qrho(i,2) = 0.5*(Q.rho(i,1) + Q.rho(i,2)) ;
Q.LQj.Qrhou(i,2) = 0.5*(Q.rhou(i,1) + Q.rhou(i,2)) ;
Q.LQj.Qrhov(i,2) = 0.5*(Q.rhov(i,1) + Q.rhov(i,2)) ;
Q.LQj.Qrhoet(i,2) = 0.5*(Q.rhoet(i,1) + Q.rhoet(i,2)) ;

Q.LQj.Qrho(i,ny+1) = 0.5*(Q.rho(i,ny) + Q.rho(i,ny+1)) ;
Q.LQj.Qrhou(i,ny+1) = 0.5*(Q.rhou(i,ny) + Q.rhou(i,ny+1)) ;
Q.LQj.Qrhov(i,ny+1) = 0.5*(Q.rhov(i,ny) + Q.rhov(i,ny+1)) ;
Q.LQj.Qrhoet(i,ny+1) = 0.5*(Q.rhoet(i,ny) + Q.rhoet(i,ny+1)) ;

Q.RQj.Qrho(i,2) = 0.5*(Q.rho(i,1) + Q.rho(i,2)) ;
Q.RQj.Qrhou(i,2) = 0.5*(Q.rhou(i,1) + Q.rhou(i,2)) ;
Q.RQj.Qrhov(i,2) = 0.5*(Q.rhov(i,1) + Q.rhov(i,2)) ;
Q.RQj.Qrhoet(i,2) = 0.5*(Q.rhoet(i,1) + Q.rhoet(i,2)) ;

Q.RQj.Qrho(i,ny+1) = 0.5*(Q.rho(i,ny) + Q.rho(i,ny+1)) ;
Q.RQj.Qrhou(i,ny+1) = 0.5*(Q.rhou(i,ny) + Q.rhou(i,ny+1)) ;
Q.RQj.Qrhov(i,ny+1) = 0.5*(Q.rhov(i,ny) + Q.rhov(i,ny+1)) ;
Q.RQj.Qrhoet(i,ny+1) = 0.5*(Q.rhoet(i,ny) + Q.rhoet(i,ny+1)) ;
end
```

```
% u and v left and right states

% u and v on xi faces
for j=2:ny
    for i = 2:nx+1
        U.Lui(i,j) = Q.LQi.Qrhou(i,j)/Q.LQi.Qrho(i,j) ;
        U.Rui(i,j) = Q.RQi.Qrhou(i,j)/Q.RQi.Qrho(i,j) ;
        U.Lvi(i,j) = Q.LQi.Qrhov(i,j)/Q.LQi.Qrho(i,j) ;
        U.Rvi(i,j) = Q.RQi.Qrhov(i,j)/Q.RQi.Qrho(i,j) ;
    end
end


% u and v on eta faces

for i = 1:nx+1
    for j=2:ny+1
        U.Luj(i,j) = Q.LQj.Qrhou(i,j)/Q.LQj.Qrho(i,j) ;
        U.Ruj(i,j) = Q.RQj.Qrhou(i,j)/Q.RQj.Qrho(i,j) ;
        U.Lvj(i,j) = Q.LQj.Qrhov(i,j)/Q.LQj.Qrho(i,j) ;
        U.Rvj(i,j) = Q.RQj.Qrhov(i,j)/Q.RQj.Qrho(i,j) ;
    end
end


end


function [M_plus] = M4_plus(M)

if abs(M) >= 1
M_plus = 0.5* (M + abs(M));
else
    beta = 1/8;
    M_plus = ( ((1/4)*(M+1)^2) * (1 - 16*beta*(-(1/4)*(M-1)^2) ));
end


end


function [M_minus] = M4_minus(M)

if abs(M) >= 1
M_minus = 0.5* (M - abs(M));
else
    beta = 1/8;
    M_minus = ((-(1/4)*(M-1)^2) * (1 + 16*beta*((1/4)*(M+1)^2) ));
end


end


function phi = minmoid(r)
```

```
if r <= 0
    phi = 0;
else
    if r >=1
        phi = 1;
    else
        phi = r;
    end
end
end


function [E,F,U] = AUSM_UP(Q,U, nx,ny,gamma, R, S_xi_x,S_xi_y,S_xi,S_eta_x,S_eta_y,
S_eta)

%% xi faces

% 1. Calculate mass flux

% Mass Flux is rhoL/R times c1/2 times M1/2.
    % Calculate M1/2 using U_xi and c1/2
%Calculate c1/2 using Temperature.
% Mp needs P left and right too.

% Left and Right states of U_xi
for j = 2:ny
    for i = 2:nx+1

        U.U_xi_L(i,j) = S_xi_x(i,j)*(U.Lui(i,j))/S_xi(i,j) +
S_xi_y(i,j)*(U.Lvi(i,j))/S_xi(i,j);
        U.U_xi_R(i,j) = S_xi_x(i,j)*(U.Rui(i,j))/S_xi(i,j) +
S_xi_y(i,j)*(U.Rvi(i,j))/S_xi(i,j);

        T_Li = ((gamma-1)/(2*((Q.LQi.Qrho(i,j))^2)*R)) *
((2*Q.LQi.Qrho(i,j)*Q.LQi.Qrhoet(i,j)) - ((Q.LQi.Qrhou(i,j))^2 +
(Q.LQi.Qrhov(i,j))^2));
        T_Ri = ((gamma-1)/(2*((Q.RQi.Qrho(i,j))^2)*R)) *
((2*Q.RQi.Qrho(i,j)*Q.RQi.Qrhoet(i,j)) - ((Q.RQi.Qrhou(i,j))^2 +
(Q.RQi.Qrhov(i,j))^2));

        P_Li = ((gamma-1)/(2*Q.LQi.Qrho(i,j))) *
((2*Q.LQi.Qrho(i,j)*Q.LQi.Qrhoet(i,j)) - ((Q.LQi.Qrhou(i,j)^2) +
(Q.LQi.Qrhov(i,j)^2)));
        P_Ri = ((gamma-1)/(2*Q.RQi.Qrho(i,j))) *
((2*Q.RQi.Qrho(i,j)*Q.RQi.Qrhoet(i,j)) - ((Q.RQi.Qrhou(i,j)^2) +
(Q.RQi.Qrhov(i,j)^2)));

        c_Li = sqrt(gamma*R*T_Li);
        c_Ri = sqrt(gamma*R*T_Ri);
        U.cface_i(i,j) = 0.5*(c_Li + c_Ri);

        MLi = U.U_xi_L(i,j)/U.cface_i(i,j);
        MRi = U.U_xi_R(i,j)/U.cface_i(i,j);
```

```
        M.facei(i,j) = M4_plus(MLi) + M4_minus(MRi) +
M_p(MLi,MRi,P_Li,P_Ri,Q.LQi.Qrho(i,j),Q.RQi.Qrho(i,j),U.cface_i(i,j));


        if M.facei(i,j) > 0
            m_dot = Q.LQi.Qrho(i,j) * U.cface_i(i,j) * M.facei(i,j);
        else
            m_dot = Q.RQi.Qrho(i,j) * U.cface_i(i,j) * M.facei(i,j);
        end


        if m_dot > 0
            psi_L = [1 ; U.Lui(i,j) ; U.Lvi(i,j) ; Q.LQi.Qrhoet(i,j)/Q.LQi.Qrho(i,j)
+ P_Li/Q.LQi.Qrho(i,j)];
            psi = psi_L ;
        else
            psi_R = [1 ; U.Rui(i,j) ; U.Rvi(i,j) ; Q.RQi.Qrhoet(i,j)/Q.RQi.Qrho(i,j)
+ P_Ri/Q.RQi.Qrho(i,j)];
            psi = psi_R ;
        end

        % 2. Calculate Pressure Flux
        % need to calculate Pressure + and -

        M_bar = 0.5*(MLi + MRi);
        f_a = sqrt((min(1,max((M_bar^2),(2^2)))))*(2-
sqrt((min(1,max((M_bar^2),(2^2))))));

        if abs(MLi)>=1
            P_plus = (1/MLi)*(0.5*(MLi+abs(MLi)));
        else
            P_plus = (0.25*((MLi + 1)^2)) * ((2 - MLi) - (16*((3/16)*((5*(f_a^2))-
4))*MLi*(-0.25*((MLi - 1)^2))));
        end

        if abs(MRi)>=1
            P_minus = (1/MRi)*(0.5*(MRi-abs(MRi)));
        else
            P_minus = (-0.25*((MRi - 1)^2)) * ((-2 - MRi) + (16*((3/16)*((5*(f_a^2))-
4))*MRi*(0.25*((MRi + 1)^2))));
        end

        P_u = - ((3/4) * P_plus * P_minus * (Q.LQi.Qrho(i,j)+ Q.RQi.Qrho(i,j)) *
(f_a*U.cface_i(i,j)) * (U.U_xi_R(i,j) - U.U_xi_L(i,j)));


        P.facei(i,j) = P_plus*P_Li + P_minus*P_Ri + P_u ;


        E.E1(i,j) = m_dot* psi(1) ;
        E.E2(i,j) = m_dot* psi(2) + S_xi_x(i,j)*P.facei(i,j)/S_xi(i,j);
        E.E3(i,j)  = m_dot* psi(3) + S_xi_y(i,j)*P.facei(i,j)/S_xi(i,j);
        E.E4(i,j)  = m_dot* psi(4);
    end
end
```

```
%% eta faces

% 1. Calculate mass flux

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

        U.V_eta_L(i,j) = (S_eta_x(i,j)*(U.Luj(i,j))/S_eta(i,j) +
S_eta_y(i,j)*(U.Lvj(i,j))/S_eta(i,j));
        U.V_eta_R(i,j) = (S_eta_x(i,j)*(U.Ruj(i,j))/S_eta(i,j) +
S_eta_y(i,j)*(U.Rvj(i,j))/S_eta(i,j));

        T_Lj = ((gamma-1)/(2*((Q.LQj.Qrho(i,j))^2)*R)) *
((2*Q.LQj.Qrho(i,j)*Q.LQj.Qrhoet(i,j)) - ((Q.LQj.Qrhou(i,j))^2 +
(Q.LQj.Qrhov(i,j))^2));
        T_Rj = ((gamma-1)/(2*((Q.RQj.Qrho(i,j))^2)*R)) *
((2*Q.RQj.Qrho(i,j)*Q.RQj.Qrhoet(i,j)) - ((Q.RQj.Qrhou(i,j))^2 +
(Q.RQj.Qrhov(i,j))^2));

        P_Lj = ((gamma-1)/(2*Q.LQj.Qrho(i,j))) *
((2*Q.LQj.Qrho(i,j)*Q.LQj.Qrhoet(i,j)) - ((Q.LQj.Qrhou(i,j)^2) +
(Q.LQj.Qrhov(i,j)^2)));
        P_Rj = ((gamma-1)/(2*Q.RQj.Qrho(i,j))) *
((2*Q.RQj.Qrho(i,j)*Q.RQj.Qrhoet(i,j)) - ((Q.RQj.Qrhou(i,j)^2) +
(Q.RQj.Qrhov(i,j)^2)));

        c_Lj = sqrt(gamma*R*T_Lj);
        c_Rj = sqrt(gamma*R*T_Rj);
        U.cface_j(i,j) = 0.5*(c_Lj + c_Rj);

        MLj = U.V_eta_L(i,j)/U.cface_j(i,j);
        MRj = U.V_eta_R(i,j)/U.cface_j(i,j);

        M.facej(i,j) = M4_plus(MLj) + M4_minus(MRj) +
M_p(MLj,MRj,P_Lj,P_Rj,Q.LQj.Qrho(i,j),Q.RQj.Qrho(i,j),U.cface_j(i,j));

        if M.facej(i,j) > 0
            m_dot = Q.LQj.Qrho(i,j) * U.cface_j(i,j) * M.facej(i,j);
        else
            m_dot = Q.RQj.Qrho(i,j) * U.cface_j(i,j) * M.facej(i,j);
        end


        if m_dot > 0
            psi_L = [1 ; U.Luj(i,j) ; U.Lvj(i,j) ; Q.LQj.Qrhoet(i,j)/Q.LQj.Qrho(i,j)
+ P_Lj/Q.LQj.Qrho(i,j)];
            psi = psi_L ;
        else
            psi_R = [1 ; U.Ruj(i,j) ; U.Rvj(i,j) ; Q.RQj.Qrhoet(i,j)/Q.RQj.Qrho(i,j)
+ P_Rj/Q.RQj.Qrho(i,j)];
            psi = psi_R ;
        end

        % 2. Calculate Pressure Flux
```

```
        % Need to calculate Pressure + and -

        M_bar = 0.5*(MLj + MRj);
        f_a = sqrt((min(1,max((M_bar^2),(2^2)))))*(2-
sqrt((min(1,max((M_bar^2),(2^2))))));

        if abs(MLj)>=1
            P_plus= (1/MLj)*(0.5*(MLj+abs(MLj)));
        else
            P_plus = (0.25*((MLj + 1)^2)) * ((2 - MLj) - (16*((3/16)*((5*(f_a^2))-
4))*MLj*(-0.25*((MLj - 1)^2))));
        end

        if abs(MRj)>=1
            P_minus = (1/MRj)*(0.5*(MRj-abs(MRj)));
        else
            P_minus = (-0.25*((MRj - 1)^2)) * ((-2 - MRj) + (16*((3/16)*((5*(f_a^2))-
4))*MRj*(0.25*((MRj + 1)^2))));
        end

        P_u = - ((3/4) * P_plus * P_minus * (Q.LQj.Qrho(i,j)+ Q.RQj.Qrho(i,j)) *
(f_a*U.cface_j(i,j)) * (U.V_eta_R(i,j) - U.V_eta_L(i,j)));

        P.facej(i,j) = P_plus*P_Lj + P_minus*P_Rj + P_u ;

        F.F1(i,j) = m_dot* psi(1) ;
        F.F2(i,j) = m_dot* psi(2) + S_eta_x(i,j)*P.facej(i,j)/S_eta(i,j);
        F.F3(i,j)  = m_dot* psi(3) + S_eta_y(i,j)*P.facej(i,j)/S_eta(i,j);
        F.F4(i,j)  = m_dot* psi(4);

    end
end


end


function [Q] = bc(Q,P_ref,V_ref,R,T_ref,gamma,nx,ny,S_eta,S_eta_x,S_eta_y,yxi,xxi)
% Boundary Conditions

% compute U,V,P,T first

U.u = zeros(nx+1,ny+1);
U.v = zeros(nx+1,ny+1);
P   = zeros(nx+1,ny+1);
T   = zeros(nx+1,ny+1);

for i =1:nx+1
    for j = 1:ny+1
        U.u(i,j) = Q.rhou(i,j)/Q.rho(i,j);
        U.v(i,j) = Q.rhov(i,j)/Q.rho(i,j);
        P(i,j)  = (gamma-1)*(Q.rhoet(i,j) - (0.5*(Q.rhou(i,j)^2 +
Q.rhov(i,j)^2)/Q.rho(i,j)));
        T(i,j)  = P(i,j)/(Q.rho(i,j)*R);
    end
```

```
end


% Apply constant Inflow
U.u(1,:) = V_ref;
U.v(1,:) = 0.0;
P(1,:) = P_ref;
T(1,:) = T_ref;
Q.rho(1,:) = P_ref/(R*T_ref);
Q.rhou(1,:) = U.u(1,:).*P_ref/(R*T_ref);
Q.rhov(1,:) = U.v(1,:).*P_ref/(R*T_ref);
Q.rhoet(1,:) = (P_ref/(gamma-1)) + 0.5.*Q.rho(1,:).*(U.u(1,:).^2 + U.v(1,:).^2);

% Apply BC at Bottom wall
% need U.u , V.v and P T at boundaries to calculate this.

for i=2:nx

uv0 = ((1/(S_eta(i,1)))*[S_eta_y(i,1), -S_eta_x(i,1); S_eta_x(i,1),
S_eta_y(i,1)])\((1/(S_eta(i,2)))*[S_eta_y(i,2)*U.u(i,2)-S_eta_x(i,2)*U.v(i,2);-
S_eta_x(i,2)*U.u(i,2)-S_eta_y(i,2)*U.v(i,2)]);
U.u(i,1) = uv0(1);
U.v(i,1) = uv0(2);
U.V_eta(i,1) = ((-yxi(i,1))/(S_eta(i,1))) * U.u(i,1) + (xxi(i,1)/S_eta(i,1)) *
U.v(i,1);
U.V_eta(i,2) = ((-yxi(i,2))/S_eta(i,2)) * U.u(i,2) + (xxi(i,2)/S_eta(i,2)) *
U.v(i,2);
P(i,1)       =  P(i,2);
T(i,1)       =  T(i,2) ;

Q.rho(i,1) = P(i,1)/(R*T(i,1));
Q.rhou(i,1) = U.u(i,1)*P(i,1)/(R*T(i,1));
Q.rhov(i,1) = U.v(i,1)*P(i,1)/(R*T(i,1));
Q.rhoet(i,1) = (P(i,1)/(gamma-1)) + 0.5*Q.rho(i,1)*(U.u(i,1)^2 + U.v(i,1)^2);
end

% 4. For Top Wall
for i=2:nx

uvny2 = ((1/(S_eta(i,end)))*[S_eta_y(i,end), -S_eta_x(i,end); S_eta_x(i,end),
S_eta_y(i,end)])\((1/(S_eta(i,end-1)))*[S_eta_y(i,end-1)*U.u(i,end-1)-S_eta_x(i,end-
1)*U.v(i,end-1);-S_eta_x(i,end-1)*U.u(i,end-1)-S_eta_y(i,end-1)*U.v(i,end-1)]);
U.u(i,end) = uvny2(1);
U.v(i,end) = uvny2(2);
U.V_eta(i,end-1) = ((-yxi(i,end-1))/(S_eta(i,end-1))) * U.u(i,end-1) + (xxi(i,end-
1)/S_eta(i,end-1)) * U.v(i,end-1);
U.V_eta(i,end) = ((-yxi(i,end))/S_eta(i,end)) * U.u(i,end) +
(xxi(i,end)/S_eta(i,end)) * U.v(i,end);
P(i,end)       =  P(i,end-1);
T(i,end)       =  T(i,end-1) ;

Q.rho(i,end) = P(i,end)/(R*T(i,end));
Q.rhou(i,end) = U.u(i,end)*P(i,end)/(R*T(i,end));
Q.rhov(i,end) = U.v(i,end)*P(i,end)/(R*T(i,end));
```

```
Q.rhoet(i,end) = (P(i,end)/(gamma-1)) + 0.5*Q.rho(i,end)*(U.u(i,end)^2 +
U.v(i,end)^2);

end

% Update Outflow:
Q.rho(nx+1,:) = Q.rho(nx,:);
Q.rhou(nx+1,:)= Q.rhou(nx,:);
Q.rhov(nx+1,:) = Q.rhov(nx,:);
Q.rhoet(nx+1,:)= Q.rhoet(nx,:);

end


function plotVvector(Q,nx,ny,xc,yc,x2d,y2d, figNum)
for i = 1:nx+1
for j = 1:ny+1
U.u(i,j) = Q.rhou(i,j)/Q.rho(i,j);
U.v(i,j) = Q.rhov(i,j)/Q.rho(i,j);
end
end

figure(figNum); clf
quiver(xc,yc,U.u,U.v)
hold on;
for i = 1:nx+2
    plot(x2d(i,1:ny+2), y2d(i,1:ny+2), 'k-','LineWidth',1); % vertical

end

for j =1:ny+2
    plot(x2d(1:nx+2,j), y2d(1:nx+2,j), 'k-','LineWidth',1); % horizontal

end
end
```