AS5460 Project-1

April 28, 2018

Sateesh Sivakoti, AE14B047
Department of Aerospace Engineering,
IIT Madras

Introduction:

Euler Equation:

The Euler equation in conservative form is given by

$$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} = 0 - -(1)$$

where ,

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho E_t \end{bmatrix}, F = \begin{bmatrix} \rho u \\ \rho u^2 + P \\ \rho u h_o \end{bmatrix}$$

The finite volume formulation of $\frac{\partial F}{\partial x}$ is given by

$$\frac{F^n_{i+1/2} - F^n_{i-1/2}}{\Delta x}$$

$$=> \frac{\partial Q}{\partial t} = -\frac{F^n_{i+1/2} - F^n_{i-1/2}}{\Delta x}$$

The flux can be represented as a vector in the following general form

$$F_{i+1/2} = \rho_i a_{1/2} C^+ \begin{bmatrix} 1 \\ u \\ h_o \end{bmatrix}_i + \rho_{i+1} a_{1/2} C^- \begin{bmatrix} 1 \\ u \\ h_o \end{bmatrix}_{i+1} + (\tilde{D}_i p_i + \tilde{D}_{i+1} p_{i+1}) \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$a_{1/2} = 0.5 * (a_i + a_{i+1})$$

$$\tilde{D}_i = \alpha^+_i (1 + \beta_i) - \beta_i D^+_i$$

$$\tilde{D}_{i+1} = \alpha^-_{i+1} (1 + \beta_{i+1}) - \beta_{i+1} D^-_{i+1}$$

$$D^+_i = 0.25 * (1 + M_i)^2 (2 - M_i)$$

$$D^-_{i+1} = 0.25 * (M_{i+1} - 1)^2 (2 + M_{i+1})$$

$$\alpha^+_i = 0.5 * (1 + sign(M_i))$$

$$\alpha^-_{i+1} = 0.5 * (1 - sign(M_{i+1}))$$

$$\beta_i = -max(0, 1 - floor(|M_i|))$$

$C^+$and C^-take up different values depending on whether Van Leer flux formulation or AUSM is being used.

Van Leer Flux Formulation:

$$C_{VL}^{+} = \alpha_i^{+}(1+\beta_i)M_i - 0.25 * \beta_i(1+M_i^2)$$

$$C_{VL}^{-} = \alpha_{i+1}^{-}(1+\beta_{i+1})M_{i+1} + 0.25 * \beta_{i+1}(1-M_{i+1}^2)$$

$$C^{+} = C_{VL}^{+}; C^{-} = C_{VL}^{-}$$

The Van Leer flux formulation does not preserve stationary contact discontinuities.

AUSM:

$$C^{+} = max[0, C_{VL}^{+} + C_{VL}^{-}]$$

$$C^{-} = min[0, C_{VL}^{+} + C_{VL}^{-}]$$

AUSM preserves stationary contact discontnuities but gives rise to non monotonicity.

First Order Time Integration:

$$\frac{Q_i^{n+1} - Q_i^n}{\Delta t} = -\frac{F_{i+1/2}^n - F_{i-1/2}^n}{\Delta x}$$

$$=> Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}(F_{i+1/2}^n - F_{i-1/2}^n) - -(2)$$

Fourth Order Time Integration (Runge-Kutta):

The fourth order Runge Kutta method estimates the derivative at a point by computing 4 derivatives of the function in the neighborhood of the point and averaging them with an associated weight to get a better estimate of the derivative.

$$Q_i^{n+1} = Q_i^n + \frac{1}{6}(R^0 + 2R^1 + 2R^2 + R^3)\Delta t - -(3)$$

$$R^0 = -(\frac{F_{i+1/2}^n - F_{i-1/2}^n}{\Delta x})$$

$$Q^{1stage} = Q^n + 0.5 * \Delta t * R^0$$

$$R^1 = -(\frac{F_{i+1/2}^{n+1/2} - F_{i-1/2}^{n+1/2}}{\Delta x})$$

$$Q^{2stage} = Q^n + 0.5 * \Delta t * R^1$$

$$R^2 = -(\frac{F_{i+1/2}^{n+1/2} - F_{i-1/2}^{n+1/2}}{\Delta x})$$

$$Q^{3stage} = Q^n + 0.5 * \Delta t * R^2$$

$$R^3 = -(\frac{F_{i+1/2}^{n+1} - F_{i-1/2}^{n+1}}{\Delta x})$$

R^0, R^1, R^2 and R^3 are computed by using the flux found out from the Q of the previous stage.

Algorithm:

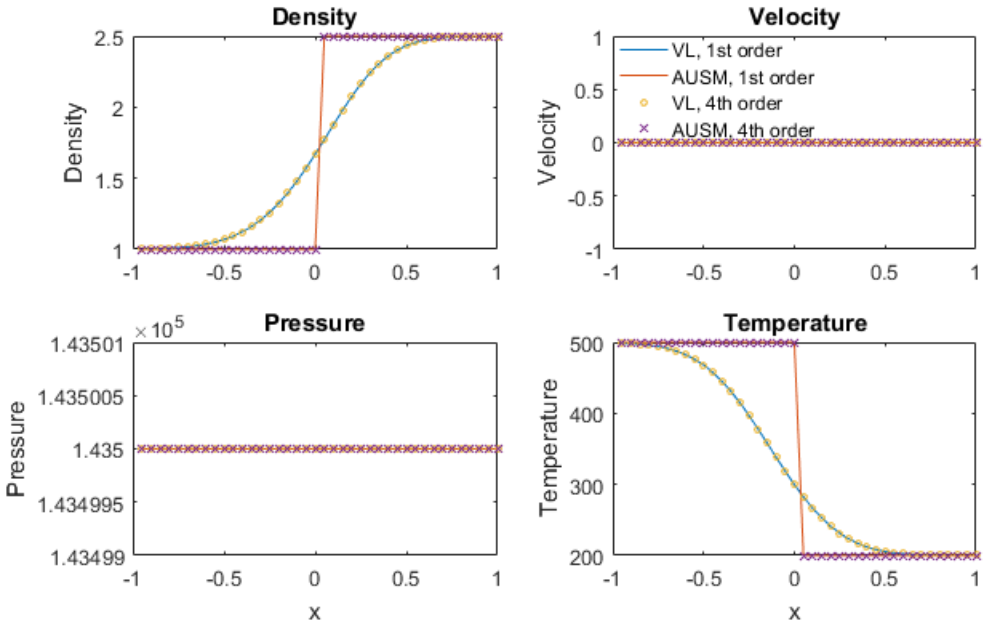
- Read input from the input file in which the flux reconstruction method, time integration method, grid information, time/ max number of iterations, CFL number and initial conditions are mentioned.
- Generate the grid and initialize the variable vectors from the input data.
- Based on whether the constraint is on number of iterations or maximum time, evolve the equation by
  - Reconstructing the flux at the interfaces.
  - Compute time step using the values of u and a through
  - $$t_g = \frac{\Delta x}{max(|u| + a)}$$
  - Time integrate using the calculated time step. If RK4 is being used, multiple calls are made to the flux reconstruction function for the intermediate time steps using the updated variables each time. Then the final estimate of Q is obtained at the next time using eqn (3). If first order time integration is used, eqn (2) is used directly.
  - Update values at the boundaries depending on whether wall bc or periodic bc is being used.
  - Extract the primitive variables from the solution (conservative variables).
  - Write the primitive variables into an output file after a specified iteration frequency.
- Plot the solution

Results:

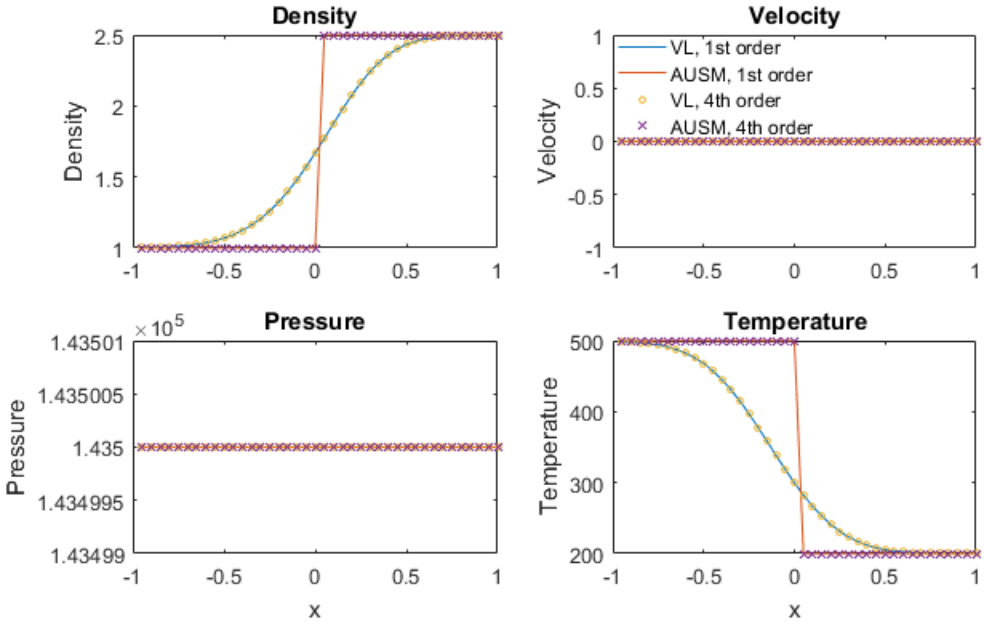
Case 1: [ρ u T] = [1 0 500] x $\leq$ 0; [ρ u T] = [2.5 0 200] x $>$ 0

This is the case of a stationary contact wave.

Wall boundaries:



Periodic boundaries:

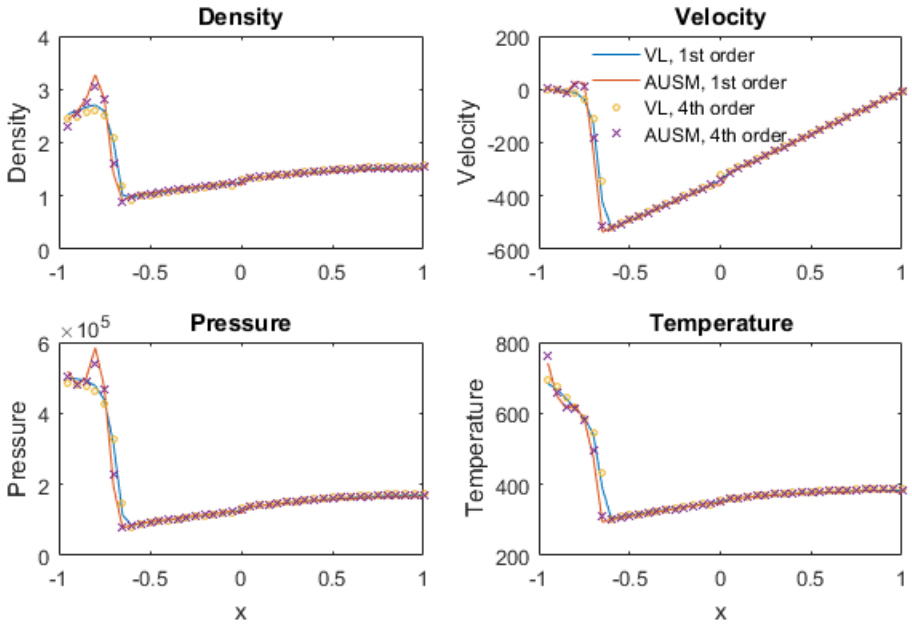


- Run time : 0.01s

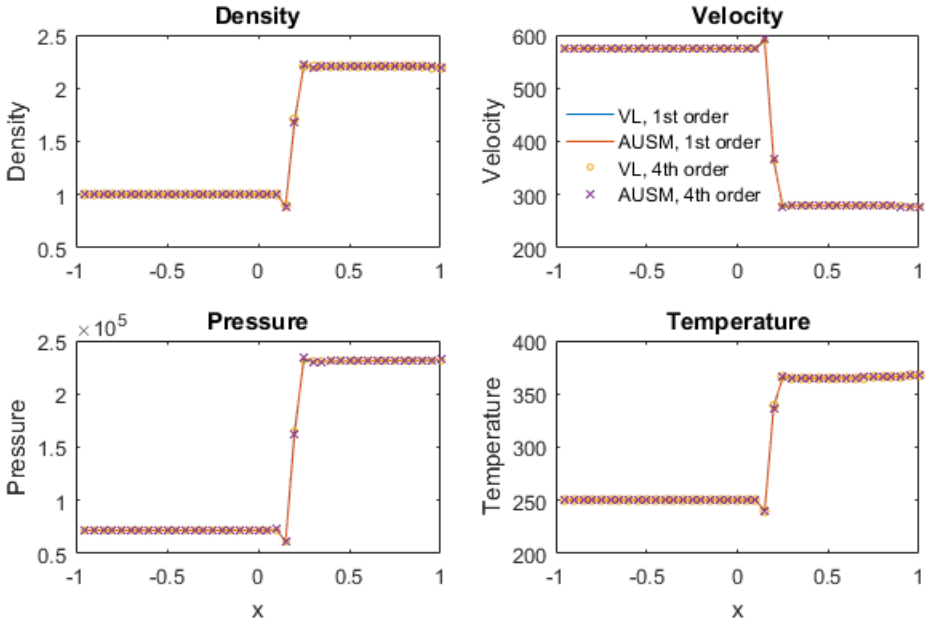
Case 2: $[\rho$ u T] = [1 574 250] x $\leq$ 0; $[\rho$ u T] = [2 287 412] x $>$ 0

This is the case of a moving shock wave.

Wall boundaries:



Periodic boundaries:

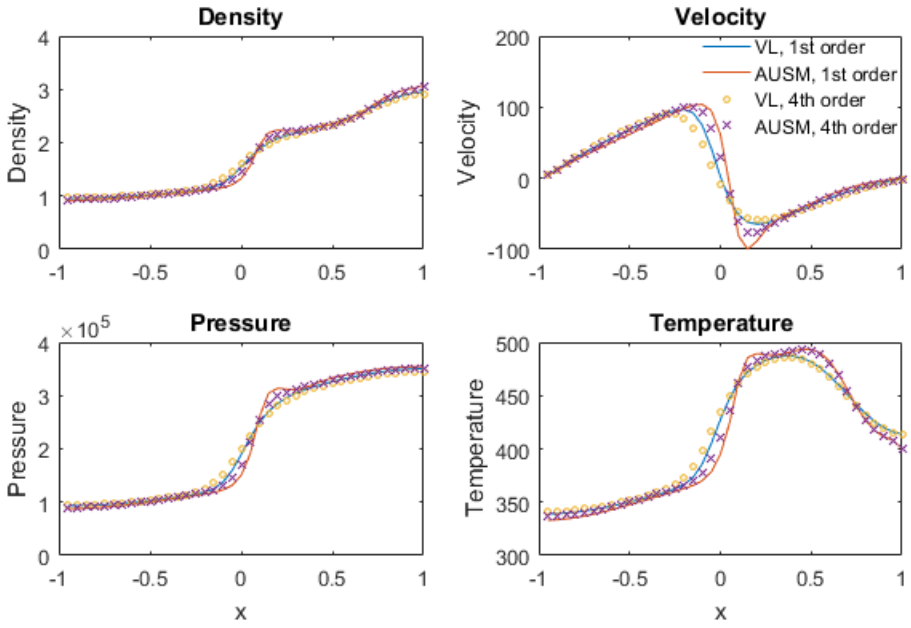


- Run time : 0.005s

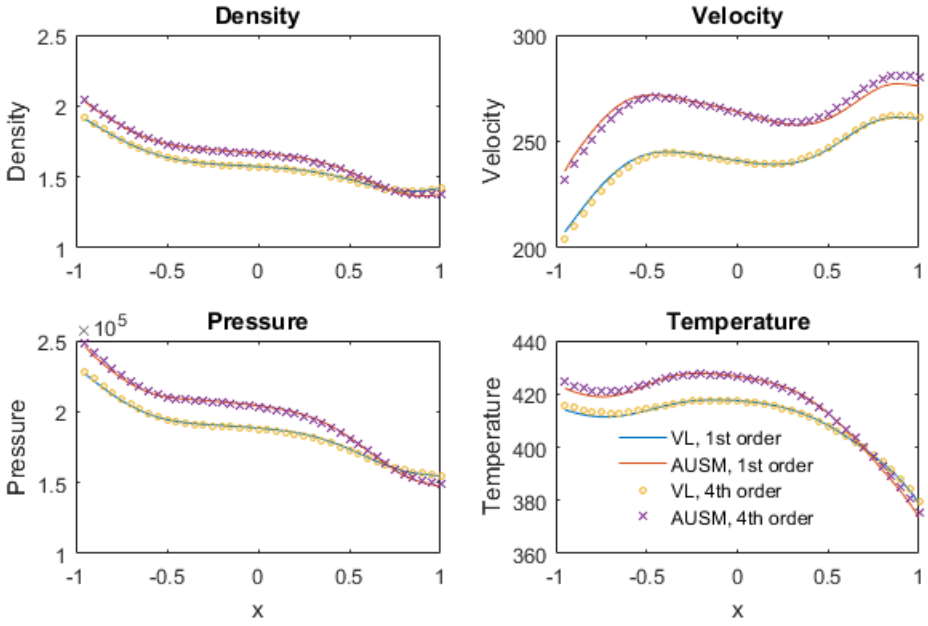
Case 3: $[\rho$ u T$]$ = [2.5 0 500] x $\leq$ 0; $[\rho$ u T$]$ = [1 0 250] x $>$ 0

This is the case of a shock tube.

Wall boundaries:



Periodic boundaries:



- Run time : 0.005s

Conclusions:

- For the case of the stationary contact wave, it is seen from the plots that fourth order and first order time integration give the same results. This is valid in the cases of wall and periodic boundary conditions.

- Van Leer scheme smears the stationary contact wave while AUSM preserves it.
- In the case of the moving shock, AUSM gives higher/lower extrema than compared to Van Leer scheme for wall boundary condition.
- The order of time integration does not seem to have too much of an impact.
- For the periodic boundary condition, neither choice of flux scheme nor the time integration make a difference.
- In the shock tube, for periodic boundary condition Van leer predicts considerably lower values for all the variables when compared to AUSM. The time integration has little effect on the solution.
- For wall condition, AUSM with first order time integration predicts the most extreme values. When used with fourth order time integration, the extrema are controlled but still more in comparison to the Van Leer plots.

Appendix:

Code:

Main:

```
clc; clear;
%% Euler Equation Solver
gamma = 1.4; R = 287;
%% Read Input File
filename = 'input.txt';
[CFL,fl_rec,time_integ,rhoL,rhoR,uL,uR,TL,TR,intervals,iterations,bc,...
    writeFreq] = get_input(filename);
%% Generate Grid and Initialize left and right states
[x, del_x, imn, imx] = generate_grid(intervals);
[rho, u, T, rhoWrite, uWrite, pWrite] = initialize(rhoL, rhoR, uL, uR, TL, TR,...
    bc,iterations,writeFreq, size(x,1));
%Vectors have been initilialized along with ghost cells
%% Initial Values
p = R*rho.*T;
a = (gamma*R.*T).^(0.5);
q1 = rho; q2 = rho.*u; q3 = p/(gamma - 1) + 0.5*rho.*u.^2;
M = u./a;
ho = q3./q1 + p./rho;
%% Evolution
tcheck = 0; iter = 0; count = 0; time = 0;
while(tcheck < 0.005 )
    fprintf('Passed %d\n',iter);
    iter = iter + 1;

    % Flux Reconstruction
    [R1, R2, R3] = flux_recon(M, ho, rho, p, u, imn, imx, a, fl_rec);
    % Time Step Calculation
    [tg, tcheck] = timeStep(CFL, u, a, tcheck, del_x);
    % Time Integration
    [q1_upd, q2_upd, q3_upd] = timeInteg(tg, del_x, q1, q2, q3, R1, R2,...
        R3, imn, imx, time_integ,fl_rec, bc);
    % Update BCs
    [q1_upd, q2_upd, q3_upd] = updateBC(q1_upd, q2_upd, q3_upd, imn, imx, bc);
    q1 = q1_upd; q2 = q2_upd; q3 = q3_upd;
    % Sanity check
    if ((min(q1) || min(q3)) < 0)
        fprintf('Obtaining unphysical negative values\n');
        break
    end
    % Extract primitive variables from the solution
```

```
    [rho, ho, T, a, M, p, u] = updateValues(q1_upd, q2_upd, q3_upd);
    % Write rho, u, p
    if mod(iter, writeFreq) == 0
        [rhoWrite,uWrite,pWrite,count] = writeSolution(count,rhoWrite,...
            uWrite,pWrite,rho,u,p,iter);
    end
end
%% % Plotting
plotSoln(rho(imn:imx), u(imn:imx), p(imn:imx), T(imn:imx), x(imn+1:imx+1))
```

Functions: (Each function has it's own MATLAB file)

```
function [CFL,fl_rec,time_integ,rhoL,rhoR,uL,uR,TL,TR,int,iters,bc,writeFreq] = get_input(fil

    fileID = fopen(file);
    C = textscan(fileID,'%f %f %f','CommentStyle','#');
    fclose(fileID);

    col1 = C{1}; col2 = C{2}; col3 = C{3};
    CFL = col1(1); fl_rec = col1(2); time_integ = col1(3); int = col1(7);
    iters = col1(8); writeFreq = col1(9); bc = col1(4);
    rhoL = col1(5); rhoR = col1(6);
    uL = col2(5); uR = col2(6);
    TL = col3(5); TR = col3(6);

function [x, del_x, imn, imx] = generate_grid(intervals)
    points =  intervals+1;
    x = linspace(-1,1,points)';
    del_x = x(2)-x(1);
    x = linspace(-1-del_x,1+del_x,points+2)';  % Including ghost cells
    imn = 2; imx = size(x,1)-2;          % Limits of computational domain

function [rho, u, T, rhoWrite, uWrite, pWrite] = initialize(rhoL, rhoR, uL, uR, TL, TR, bc, i
    R = 287;
    rho = zeros(len-1,1);
    rho(1: 0.5*(size(rho,1))) = rhoL;
    rho(0.5*(size(rho,1)+2) : end) = rhoR;

    u = zeros(len-1,1);
    u(1: 0.5*(size(rho,1))) = uL;
    u(0.5*(size(rho,1)+2) : end) = uR;
    u(1) = bc*uL;
```

```
    u(end) = bc*uR;

    T = zeros(len-1,1);
    T(1: 0.5*(size(T,1))) = TL;
    T(0.5*(size(T,1)+2) : end) = TR;

    % Write variables
    rhoWrite = zeros(len-2, iters/writeFreq + 1);
    rhoWrite(1,1) = 0 ; rhoWrite(2:end, 1) = rho(2:end-1);
    uWrite = zeros(len-2, iters/writeFreq + 1);
    uWrite(1,1) = 0 ; uWrite(2:end, 1) = u(2:end-1);
    pWrite = zeros(len-2, iters/writeFreq + 1);
    pWrite(1,1) = 0 ; pWrite(2:end, 1) = rho(2:end-1).*T(2:end-1)*R;

function [R1, R2, R3] = flux_recon(M, ho, rho, p, u, imn, imx, a, fl_rec)
    % Flux Formulation Parameters
    alpha_p = 0.5*(1 + sign(M));
    alpha_n = 0.5*(1 - sign(M));
    beta = -max(0, 1 - floor(abs(M)));
    D_p = 0.25*(1 + M).^(2).*(2 - M);
    D_n = 0.25*(M - 1).^(2).*(2 + M);
    Di = alpha_p.*(1 + beta) - beta.*D_p;
    Di1 = alpha_n.*(1 + beta) - beta.*D_n;
    Cvl_p = alpha_p.*(1 + beta).*M - 0.25*beta.*((1 + M).^2);
    Cvl_n = alpha_n.*(1 + beta).*M + 0.25*beta.*((1 - M).^2);

    Cvl_p1 = max(0,(Cvl_p(1:imx) + Cvl_n(2:imx+1)));    % AUSM
    Cvl_n1(2:imx+1) = min(0,(Cvl_p(1:imx) + Cvl_n(2:imx+1)));
    Cvl_p1(imx+1)=0;Cvl_n1(1) = 0;

    if fl_rec == 1
        Cvl_p = Cvl_p1;
        Cvl_n = Cvl_n1;
    end

    fl_q1 = zeros(size(M,1)-1,1);fl_q2 = zeros(size(M,1)-1,1);
    fl_q3 = zeros(size(M,1)-1,1);
    R1 = zeros(size(M,1)-2,1);R2 = zeros(size(M,1)-2,1);
    R3 = zeros(size(M,1)-2,1);
    % Flux
    for i = imn-1: imx
        a_av = 0.5*(a(i) + a(i+1));
        fl_q1(i) = rho(i)*a_av*Cvl_p(i)*1  + rho(i+1)*a_av*Cvl_n(i+1)*1;
        fl_q2(i) = rho(i)*a_av*Cvl_p(i)*u(i)  + rho(i+1)*a_av*Cvl_n(i+1)*u(i+1)...
            + (Di(i)*p(i)+Di1(i+1)*p(i+1))*1;
```

```
        fl_q3(i) = rho(i)*a_av*Cvl_p(i)*ho(i)  + rho(i+1)*a_av*Cvl_n(i+1)*ho(i+1);
    end
    % Residue
    for i = imn: imx
        R1(i-1) = fl_q1(i) - fl_q1(i-1);
        R2(i-1) = fl_q2(i) - fl_q2(i-1);
        R3(i-1) = fl_q3(i) - fl_q3(i-1);
    end

function [tg, tcheck] = timeStep(CFL, u, a, tcheck, del_x)
    tinv = (max(abs(u)+ a))/del_x;       % t inverse
    tg = CFL*1/max(tinv);                % Global time step
    tcheck = tcheck+tg;

function [q1_upd, q2_upd, q3_upd] = timeInteg(tg, del_x, q1, q2, q3, R1,...
    R2, R3, imn, imx, time_integ, fl_rec, bc)
    q1_upd = zeros(size(q1));q2_upd = zeros(size(q1));
    q3_upd = zeros(size(q1));

    if (time_integ == 0)
        q1_upd(imn:imx) = q1(imn:imx) - (tg/del_x)*R1;
        q2_upd(imn:imx) = q2(imn:imx) - (tg/del_x)*R2;
        q3_upd(imn:imx) = q3(imn:imx) - (tg/del_x)*R3;
    end

    if (time_integ == 1)     % RK4
       R11 = R1; R12 = R2; R13 = R3;
       k11 = -R11/del_x; k12 = -R12/del_x; k13 = -R13/del_x;
       q1_upd(imn:imx) = q1(imn:imx) + 0.5*tg*k11;
       q2_upd(imn:imx) = q2(imn:imx) + 0.5*tg*k12;
       q3_upd(imn:imx) = q3(imn:imx) + 0.5*tg*k13;
       [q1_upd, q2_upd, q3_upd] = updateBC(q1_upd, q2_upd, q3_upd, imn, imx, bc);
       [rho, ho, ~, a, M, p, u] = updateValues(q1_upd, q2_upd, q3_upd);

       [R21, R22, R23] = flux_recon(M, ho, rho, p, u, imn, imx, a, fl_rec);
       k21 = -R21/del_x; k22 = -R22/del_x; k23 = -R23/del_x;
       q1_upd(imn:imx) = q1(imn:imx) + 0.5*tg*k21;
       q2_upd(imn:imx) = q2(imn:imx) + 0.5*tg*k22;
       q3_upd(imn:imx) = q3(imn:imx) + 0.5*tg*k23;
       [q1_upd, q2_upd, q3_upd] = updateBC(q1_upd, q2_upd, q3_upd, imn, imx, bc);
       [rho, ho, ~, a, M, p, u] = updateValues(q1_upd, q2_upd, q3_upd);

       [R31, R32, R33] = flux_recon(M, ho, rho, p, u, imn, imx, a, fl_rec);
       k31 = -R31/del_x; k32 = -R32/del_x; k33 = -R33/del_x;
       q1_upd(imn:imx) = q1(imn:imx) + tg*k31;
```

```
        q2_upd(imn:imx) = q2(imn:imx) + tg*k32;
        q3_upd(imn:imx) = q3(imn:imx) + tg*k33;
        [q1_upd, q2_upd, q3_upd] = updateBC(q1_upd, q2_upd, q3_upd, imn, imx, bc);
        [rho, ho, ~, a, M, p, u] = updateValues(q1_upd, q2_upd, q3_upd);

        [R41, R42, R43] = flux_recon(M, ho, rho, p, u, imn, imx, a, fl_rec);
        k41 = -R41/del_x; k42 = -R42/del_x; k43 = -R43/del_x;



        q1_upd(imn:imx) = q1(imn:imx) + tg*(k11 + 2*k21 + 2*k31 + k41)/6;
        q2_upd(imn:imx) = q2(imn:imx) + tg*(k12 + 2*k22 + 2*k32 + k42)/6;
        q3_upd(imn:imx) = q3(imn:imx) + tg*(k13 + 2*k23 + 2*k33 + k43)/6;
    end

function [q1_upd, q2_upd, q3_upd] = updateBC(q1_upd, q2_upd, q3_upd, imn, imx, bc)
    q1_upd(1) = q1_upd(imn);
    q1_upd(end) = q1_upd(imx);

    q2_upd(1) = bc*q2_upd(imn);
    q2_upd(end) = bc*q2_upd(imx);

    q3_upd(1) = q3_upd(imn);
    q3_upd(end) = q3_upd(imx);

function [rho, ho, T, a, M, p, u] = updateValues(q1_upd, q2_upd, q3_upd)
    R = 287; gamma = 1.4;
    rho = q1_upd; u = q2_upd./q1_upd; p = (q3_upd - 0.5*(q2_upd.^2)./q1_upd)*(gamma-1);
    ho = q3_upd./q1_upd + p./rho;
    T = p./(rho*R);
    a = (gamma*R*T).^(0.5);
    M = u./a;

function [rhoWrite,uWrite,pWrite,count] = writeSolution(count,rhoWrite,uWrite,pWrite,rho,u,p,
    count = count + 1;
    rhoWrite(2:end,count+1) = rho(2:end-1); rhoWrite(1,count+1) = iter;
    uWrite(2:end,count+1) = u(2:end-1); uWrite(1,count+1) = iter;
    pWrite(2:end,count+1) = p(2:end-1); pWrite(1,count+1) = iter;

    save('Density.txt', 'rhoWrite', '-ascii');
    save('Velocity.txt', 'uWrite', '-ascii');
    save('Pressure.txt', 'pWrite', '-ascii');

function plotSoln(rho, u, p, T, x)
    subplot(2,2,1)
    plot(x, rho);
```

```
ylabel('Density');
title('Density');
hold on;

subplot(2,2,2)
plot(x, u);
ylabel('Velocity');
title('Velocity');
hold on;

subplot(2,2,3)
plot(x, p);
ylabel('Pressure');
xlabel('x');
title('Pressure');
hold on;

subplot(2,2,4)
plot(x, T);
ylabel('Temperature');
xlabel('x');
title('Temperature');
hold on;
```