2018

APPLICATION OF FINITE DIFFERENCE METHODS TO SOD SHOCK TUBE PROBLEM

MEEN 689-FINAL PROJECT

SATYA AKHIL REDDY BACHUGUDAM

826006779

1. INTRODUCTION

This report presents a finite difference analysis of the shock tube problem using SOD initial data. A shock tube consists of a long tube filled with the same gas in two different states and separated by a diaphragm at the center. The initial conditions are very simple: a contact discontinuity separating gasses with different pressure and density and zero velocity everywhere.

The shock tube problem is one of the standard problems in gas dynamics. It is used as a test problem for checking the accuracy of computational codes since the exact solution is known and can be compared with the simulation results. The problem is defined by a set of conservative equations, also known as the Euler equations of compressible gas dynamics. In our analysis, we only consider the problem along a single direction, thus giving rise to the following Euler gas dynamics equations.

$$\frac{\delta \vec{Q}}{\delta t} + \frac{\delta \vec{E}}{\delta x} = 0$$

where,

$$\vec{Q} = \begin{pmatrix} \rho \\ m \\ E \end{pmatrix}$$

$$\vec{E} = \begin{pmatrix} m \\ mu + p \\ Eu + pu \end{pmatrix}$$

Here ρ is the density, u is the velocity, p is the pressure, m is the mass and E is the total energy of the medium. The total energy can be expanded as $E = \frac{1\rho u^2}{2} + \frac{p}{\gamma - 1}$. In our case, we assume an ideal gas with $\gamma = 1.4$. We use SOD initial data as the initial conditions and hence the name SOD shock tube problem. The initial conditions are

$$\vec{Q} = \begin{pmatrix} \rho \\ m \\ E \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 2.5 \end{pmatrix} \; in \; (0,0.5)$$

$$\vec{Q} = \begin{pmatrix} \rho \\ m \\ E \end{pmatrix} = \begin{pmatrix} 0.125 \\ 0 \\ 0.25 \end{pmatrix} \; in \; (0.5,1)$$

The Euler equations of gas dynamics fall under the category of hyperbolic partial differential equations. The solution to a hyperbolic PDE (disturbance) propagates through the space at a finite speed and travels along the characteristics of the equation. To find the solution to the above defined system of equations, we have chosen one from each of lower order, monotonicity preserving and higher order schemes, whose method of solution has been mentioned in detail in the upcoming section.

2. METHOD OF SOLUTION

To start with, let us get an understanding of the Reimann problem. It is a problem posed to figure out the state value at a discontinuous interface, like the values at half nodes for finite volume scheme. Solving the Reimann problem at the interface helps to make the solution more and physical because it considers the flow of information through the characteristics.

A mesh was generated using a grid size of 0.01. The time step was determined specific to each scheme, since the stability conditions had to be satisfied. The following schemes were considered for each of the following categories:

- **LOWER ORDER** : Godunov scheme with Lax-Friedrich Flux
- **MONOTONOCITY PRESERVING** : Godunov with MUSCL scheme
- **HIGHER ORDER** : Lax-Wendroff scheme (Richtmeyer two step)

Let us look at the method of computation followed in each of the above schemes.

GODUNOV SCHEME WITH LAX-FRIEDRICH FLUX

Godunov is a finite volume scheme where the flux is evaluated at half points rather than at the grid points .This scheme is first order accurate in space and time. In this scheme, we use the cell averages at each time level as the main object of interest for our approximation. The cell average is defined as

$$Q_i^n = \frac{1}{\Delta x}\int_{x_{j-1/2}}^{x_{j+1/2}} Q(x,t^n)dx$$

In any finite volume method, the aim is to compute the cell average values at the end of each time step. Upon discretizing the set of Euler equations, we get

$$\oiiint \frac{\delta \vec{Q}}{\delta t} + \frac{\delta \vec{E}}{\delta x} dv = 0$$

$$\frac{\delta}{\delta t}\oiiint \vec{Q}dv + \oiint \vec{E}.ndS = 0$$

In one dimension, the above equation can be modified as

$$\frac{\delta}{\delta t}\bar{\vec{Q}}\Delta v + \left(-\vec{E}_{i-\frac{1}{2}} + \vec{E}_{i+\frac{1}{2}}\right) = 0$$

Finally, we get

$$\bar{\vec{Q}}_i^{n+1} = \bar{\vec{Q}}_i^n - (\frac{\Delta t}{\Delta x}) \left(-\vec{E}_{i-\frac{1}{2}} + \vec{E}_{i+\frac{1}{2}}\right)$$

The above equation represents the discretized equation obtained using Godunov scheme. Since the flux functions are dependent on the cell averages which are of first order, trying to solve the Reimann problem to calculate the fluxes would be computationally intensive. We avoid solving the Reimann problem by using an approximate flux function to get same order of accuracy in our solution. In this analysis, we used Lax-Freidrich flux function which is defined as

$$F(u,v) = 0.5\big(E(u) + E(v)\big) - 0.5\alpha(v-u)$$

Where $u = \vec{\bar{Q}}_i$, $v = \vec{\bar{Q}}_{i+1}$ and α is the maximum eigen value of the Jacobian function applied on flux vector. Thus, the final discretize equation used for computation was

$$\vec{\bar{Q}}_i^{n+1} = \vec{\bar{Q}}_i^n - (\frac{\Delta t}{\Delta x})\left(-\vec{F}_{i-\frac{1}{2}} + \vec{F}_{i+\frac{1}{2}}\right)$$

The stability criterion (CFL value) limits the value of maximum time step and is defined as

$$\max(u)\frac{\Delta t}{\Delta x} \leq \frac{1}{2}$$

For easier understanding of the scheme, the stencil is displayed in Fig. 2.

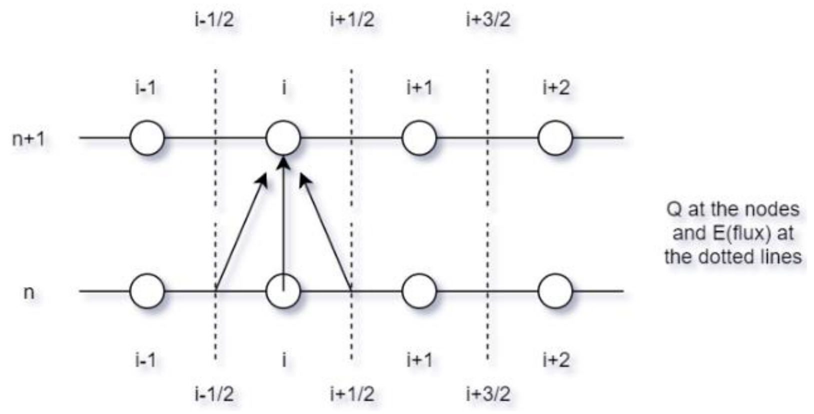

Figure 2 Stencil for Godunov scheme

GODUNOV WITH MUSCL SCHEME

In the previous scheme, the piecewise cell averages were considered in calculating the fluxes. This causes the evaluated fluxes to become first order accurate. In order to increase the accuracy, the piecewise constant approximation of Godunov's scheme is replaced by reconstructed piecewise polynomials. These polynomials are derived from cell averages according to the following formulation

$$\tilde{\vec{\bar{Q}}}_i = \vec{\bar{Q}}_i + s * (x - x_i)$$

Where

$$s = minmod\{\frac{\vec{\bar{Q}}_{i+1} - \vec{\bar{Q}}_i}{\Delta x}, \frac{\vec{\bar{Q}}_i - \vec{\bar{Q}}_{i-1}}{\Delta x}\}$$

From the above polynomials, the fluxes on the left and right are evaluated at any given half node. These fluxes can then be used as input to the Lax-Freidrich flux, which is then used in the Godunov scheme to obtain the values at the current time step.

$$F_{i+1/2} = 0.5\left(E\left(\tilde{\vec{\bar{Q}}}_i\right)_{i+\frac{1}{2}} + E\left(\widetilde{\vec{\bar{Q}}_{i+1}}\right)_{i+\frac{1}{2}}\right) - 0.5\alpha(\left(\widetilde{\vec{\bar{Q}}_{i+1}}\right)_{i+\frac{1}{2}} - \left(\tilde{\vec{\bar{Q}}}_i\right)_{i+\frac{1}{2}})$$

Thus, the polynomials must be evaluated at each time step to carry forward the computation until the final solution is obtained. The stability condition and stencil remain the same as shown in Fig. 2 for Godunov with MUSCL scheme.

LAX-WENDROFF SCHEME

Lax-Wendroff is a two-step method, where the first step calculates the values for E(Q) at half time steps and half grid points, followed by the second step which calculates the data at the grid points using the values from the first step. This scheme is second order accurate in both space and time. Since the Euler equations are non-linear, we can avoid the computation of Jacobian by following the Richtmeyer two-step Lax Wendroff method. The discretization followed in the first step (predictor step) is shown below

$$\vec{Q}_{i+1/2}^{*} = \frac{\vec{Q}_i^n + \vec{Q}_{i+1}^n}{2} - \frac{\Delta t}{2\Delta x}\left(\vec{E}_{i+1}^n - \vec{E}_i^n\right)$$

The values calculated in the above equation are then used in the second step (corrector step) as shown

$$\vec{Q}_i^{n+1} = \vec{Q}_i^n - \frac{\Delta t}{\Delta x}(\vec{E}_{i+1/2}^{*} - \vec{E}_{i-1/2}^{*})$$

The stability criterion determines the maximum value of time step that can be used in a problem and is defined as

$$\Delta t = \frac{CFL * \Delta x}{|\vec{Q}| + a}$$

Where a is the speed of sound and $CFL \leq 1$. The stencil followed by the scheme is shown in Fig. 3. The value of ith node depends on the values at i-1, i, i+1 and the values of half nodes between them.

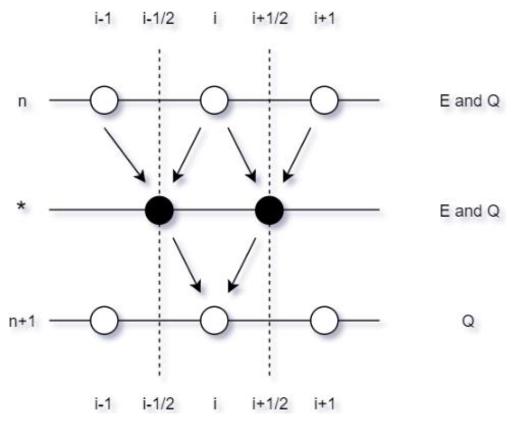

Figure 3 Stencil for Lax-Wendroff scheme

RESULTS AND DISCUSSION

The above-mentioned schemes were solved in MATLAB whose results have been discussed below. Only the components j=2:(N-1) of the solution were calculated from the schemes. The remaining components j=1 and j=N were prescribed by the Dirichlet boundary conditions. Practically, this is equivalent to leaving unchanged the first and last vectors of the solution vector. The computation was stopped at a time of t=0.1644s before one of the waves hits the boundary.

To understand the numerical solution developed by the schemes, let us attempt to understand the phenomenon from the point when diaphragm breaks at time t=0. To equalize the pressure in the tube, the gas at high pressure expands through an expansion wave and flows in to the low-pressure region, pushing the gas in this region apart. The compression of the low-pressure gas generates a shock wave that propagates to the right. The shock wave increases the temperature and pressure of the low-pressure gas and induces a flow in its direction. The expanded gas is separated from the compressed gas by a contact discontinuity, which is regarded as a fictitious membrane travelling to the right at constant speed.

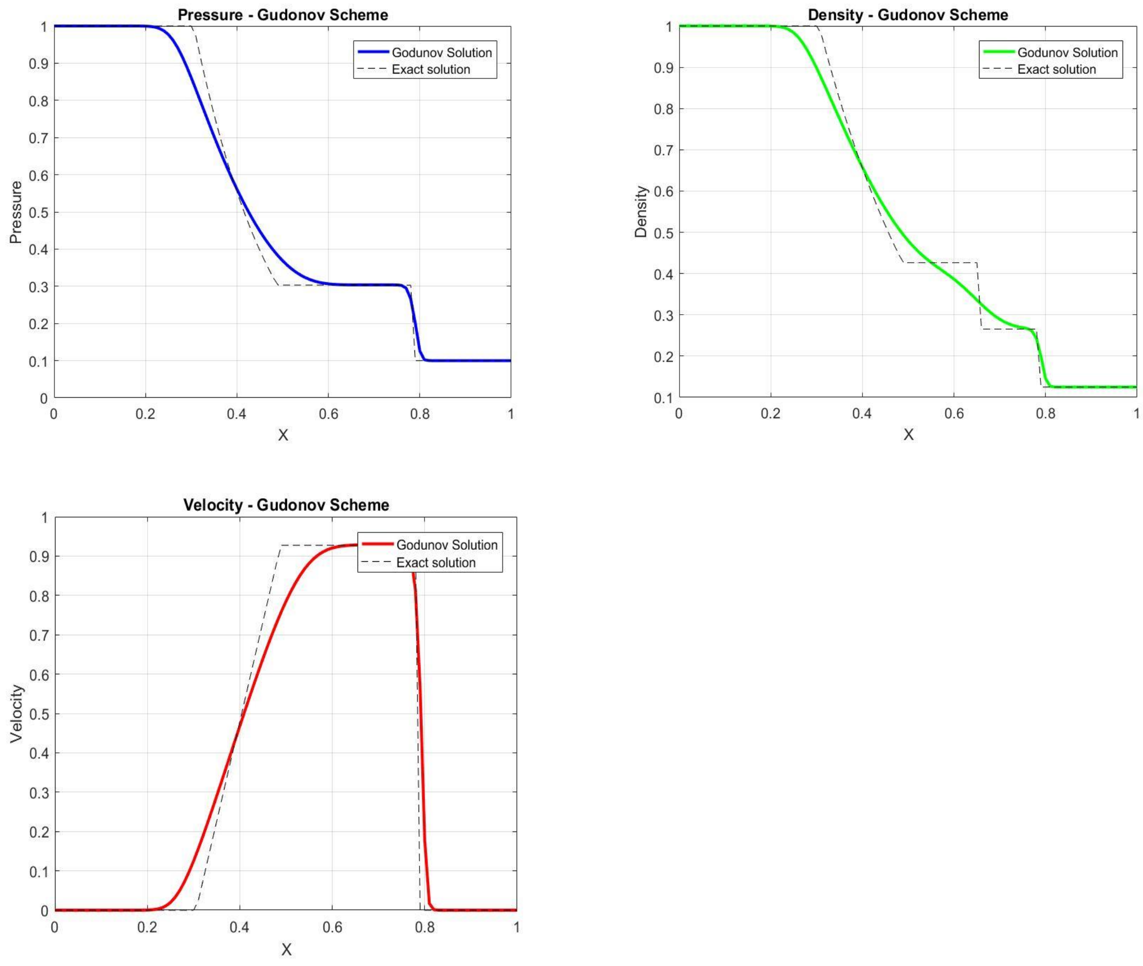

Figure 4 Pressure, density and velocity plots obtained using Godunov scheme

Figure 4 shows the plots of pressure, velocity and density obtained using the Godunov scheme with grid size and time step of Δx=0.01 and Δt=0.001644. The curve in dotted fashion represents the exact solution whereas the continuous curve represents the variable value obtained using Godunov scheme. By observing the numerical solution obtained, we find that the physics of the problem was rightly captured by the scheme. The results obtained were smooth and did not include any kind of oscillations.

Coming to the performance of Godunov scheme, we find that it performs well in approximating the solution in smooth regions where pressure, density and velocity do not have any discontinuities. The shock and expansion waves were accurately captured. Being first order in nature, the level of accuracy obtained is commendable. But the scheme fails badly in capturing the discontinuities before and after the expansion and shock waves. We know that Godunov scheme is a highly dissipative method, where in the efforts to decrease the ringing produced by lower order numerical schemes lead to smearing of the solution near the discontinuities.

To overcome this problem, one can attempt to increase the order of approximation of **Q** or choose methods with higher order of accuracy. Next, we look at the performance of Godunov with MUSCL scheme against the benchmark solutions.

Figure 5 shows the pressure, density and velocity plots obtained using Godunov with MUSCL scheme with grid size and time step of Δx=0.01 and Δt=0.00001644. Such small value of time step was taken to satisfy the stability criterion. Like the above case, dotted curve shows the exact solution whereas the continuous curve shows the solution to the variable obtained using Godunov with MUSCL scheme. By looking at the plots, we see that the physics of the problem was rightly captured by the scheme. The results obtained were smooth and did not include any oscillations.

Similar to the Godunov scheme, this scheme performs very well in approximating the solution in smooth regions where there are no discontinuities. The shock and expansion waves were captured sharply and accurately. The discontinuities look smeared due to the dissipative nature of the scheme, but the performance is better when compared to Godunov scheme. The high dissipative nature of Godunov is negated by the higher order approximation leading to small amount of anti-diffusion. This enables the solution near the discontinuities to be captured with better accuracy. Many other higher order approximations can be applied to Godunov to accurately capture the discontinuities, which can be left as the future scope of this work.

Next, we look at the performance of the higher order method, the two-step Lax-Wendroff scheme. We compare it to the benchmark solution and later compare the schemes amongst themselves.

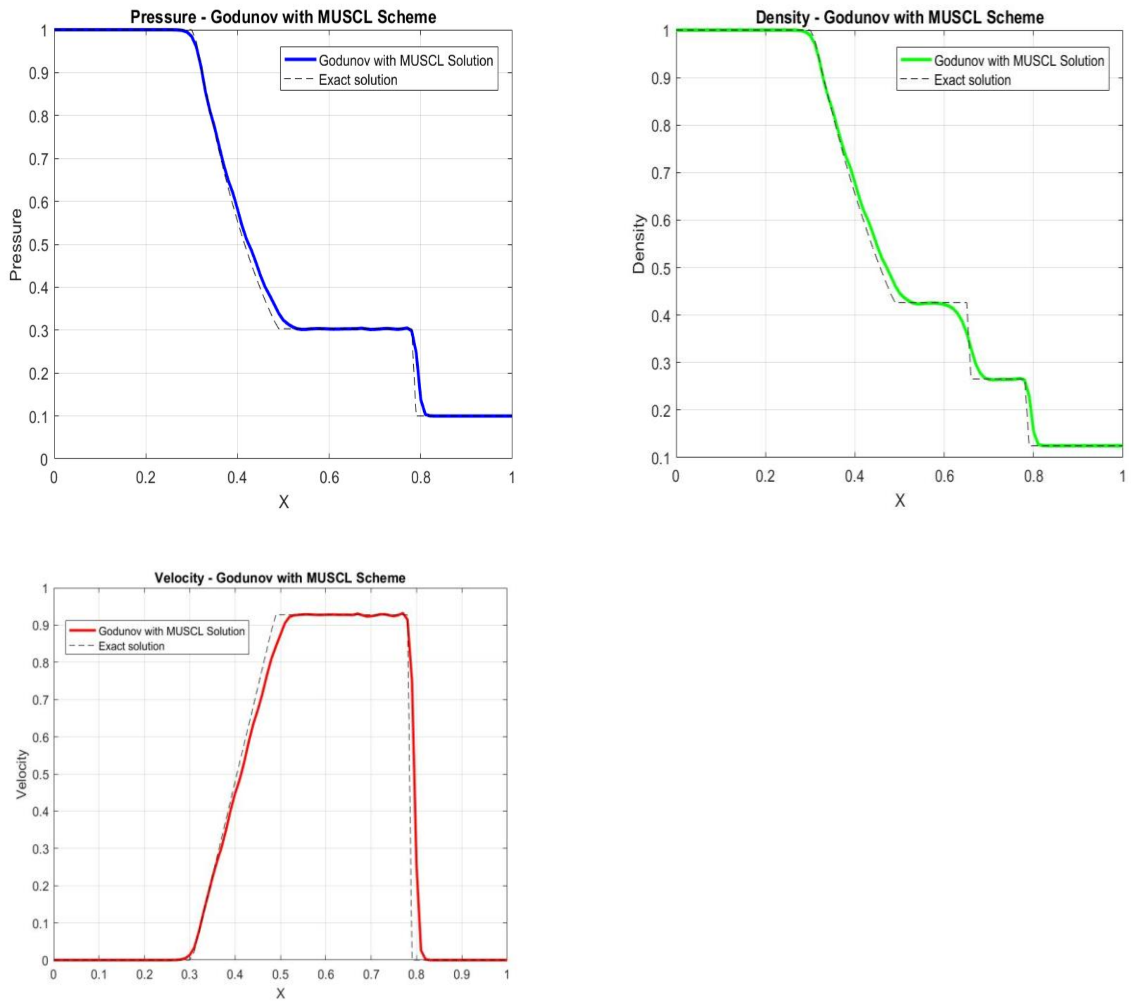

Figure 5 Pressure, density and velocity plots obtained using Godunov with MUSCL scheme

Figure 5 shows the pressure, density and velocity plots obtained using Godunov with MUSCL scheme with grid size and time step of Δx=0.01 and Δt=0.001644. The value of CFL used was close to 1. The dotted curve represents the exact solution to the Reimann problem, whereas the continuous curve plots the solution to a variable using Lax-Wendroff scheme. From the plots, we can infer that the physics of the problem was rightly captured by the scheme. The results obtained were smooth, but oscillations were present near the discontinuities

The scheme does an excellent job in approximating the solution in smooth regions. There is not much of a difference between the exact and computed solution. The shock and expansion waves were captured very accurately and sharply. The problem again arises in capturing the discontinuities near the end of expansion and beginning of shock waves. This is because two step Lax-Wendroff is centered scheme, where there is no component present to add dissipation to the scheme. Dispersion is useful for shock capturing as it keeps the shock jump discontinuity relatively vertical and the expansion fan relatively linear. However, this is at the expense of diminishing true local information with the oscillations.

To overcome such oscillations, we can add artificial dissipation around the discontinuities to the problem. This is done by adding a supplementary term to the initial equation.

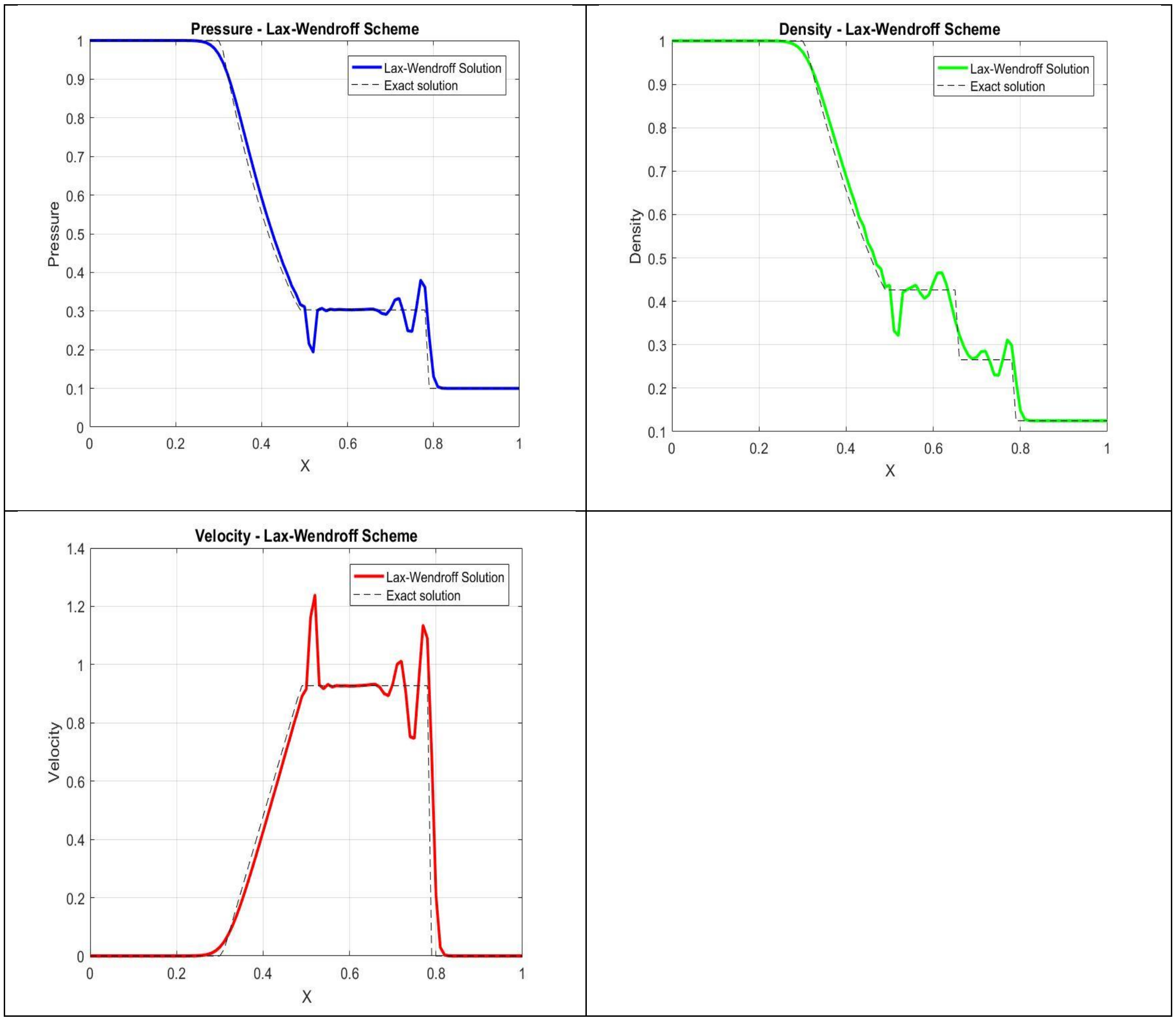

Figure 6 Pressure, density and velocity plots obtained using Lax-Wendroff scheme

From the above comparisons with the benchmark solution, we find that Godunov with MUSCL scheme performed the best with respect to approximating the discontinuities. The Lax-Wendroff scheme was very accurate in capturing the solution in smooth regions. The solution obtained through Godunov scheme happened to be smeared at the discontinuities and did not capture the solution accurately enough in the smooth regions around the discontinuities.

Looking at Godunov and Godunov with MUSCL, we find that the latter outperforms the former scheme by a large difference. This is because the order of approximation in MUSCL scheme is higher than regular Godunov scheme. In Godunov scheme, we use the cell average values directly in computing the LHS and RHS flux at any given half point. Thus, the order of such approximation is 1. In MUSCL scheme, we develop linear piecewise approximations and use them in computation of RHS and LHS flux, leading to second order approximation. This considers the information of the adjacent nodes, thereby leading to a stable and accurate solution.

Now, we compare Godunov with MUSCL and Lax-Wendroff schemes. These two schemes perform very well in smooth regions, but MUSCL outperforms Lax-Wendroff in capturing the discontinuities. As already mentioned, Lax-Wendroff is a centered scheme and thus there is no component which provides dissipation around the discontinuities. To improve Lax-Wendroff, one can add artificial dissipation to obtain a better solution around the discontinuities.

The computational times of the above schemes are mentioned in the Table 1 below

Scheme	*Time(s)*
Godunov	0.055669
Godunov with MUSCL	4.953
Lax-Wendroff	0.00504

Table 1 Computation time of diffeent schemes

We find that Lax-Wendroff has the lowest computational time while Godunov with MUSCL has the highest computational time. When we add the artificial dissipation to Lax-Wendroff, the computation time would not exceed that of Godunov scheme (needs to be proved). Godunov with MUSCL takes the highest time since the time step is very small. This was done to satisfy the stability criterion. Also, the time is high since piecewise linear approximations had to be developed for each time step.

<u>SUMMARY</u>

The SOD shock tube problem was considered to check the performance of certain hyperbolic schemes. The problem was defined using 1-D Euler equations of gas dynamics. The equations were discretized according to the pattern laid out by the specific schemes. MATLAB was used to develop the code and run the simulations. The results were plotted for pressure, density and velocity.

In the Godunov scheme, the solution obtained was smooth in regions without any discontinuities. The scheme failed to capture the discontinuities due to the highly dissipative nature of the scheme. In the Godunov with MUSCL scheme, the shortcoming of the former scheme is taken care of by the development of piecewise linear approximations. This scheme is successful in capturing the solution with relative ease near the discontinuities. In Lax-Wendroff, the centered nature of the scheme causes dispersion near the discontinuities, which can be dealt with the introduction of artificial dissipation. The solution captured in the smooth regions is accurate and sharp. All the above schemes capture the expansion and shock waves accurately.

<u>CONCLUSION</u>

There is no single scheme which would suit all the purposes of a problem. If the aim of the problem is to find an accurate solution, then one can use Godunov with MUSCL scheme and Lax-Wendroff coupled with artificial dissipation. If the aim of the problem is to find a solution in a timed situation, then Lax-Wendroff coupled with artificial dissipation can be applied to the equations.

APPENDIX

Godunov Scheme

```
%% CFD - Final Project - Godunov scheme

%% Given Parameters

T_final=0.1644;
L=1;
dx=0.01;
g=1.4;
cfl=0.8;

%% Discretization Parameters

dt=0.001644;
T=0:dt:T_final;
X=0:dx:L;

%% Initialization

Q=zeros(3,size(X,2));
E=zeros(3,size(X,2));
F=zeros(3,size(X,2));
alpha=zeros(1,size(X,2)-1);

%% Initial SOD conditions

Q(1,1:(0.5/dx))=1;
Q(1,(0.5/dx)+1:size(X,2))=0.125;

Q(2,1:(0.5/dx))=0;
Q(2,(0.5/dx)+1:size(X,2))=0;

Q(3,1:(0.5/dx))=2.5;
Q(3,(0.5/dx)+1:size(X,2))=0.25;

E(1,:)=Q(2,:);
E(2,:)=((3-g)/2)*((Q(2,:).^2)./Q(1,:))+(g-1)*Q(3,:);
E(3,:)=(Q(2,:)./Q(1,:)).*(g*Q(3,:)-((g-1)/2)*((Q(2,:).^2)./Q(1,:)));

% E(1,:)=Q(2,:);
% E(2,:)=((g-3)/2)*Q(1,:).*Q(2,:)+(g-1)*Q(3,:);
% E(3,:)=(Q(2,:)./Q(1,:)).*(g*Q(3,:)-((g-1)/2)*Q(1,:).*Q(2,:));

%% Velocity, Pressure and density from Q and E vectors

u=Q(2,:)./Q(1,:);
rho=Q(1,:);
et=Q(3,:)./Q(1,:);
e=et-u.^2/2;
p=e.*rho*(g-1);

tic
%% Godunov Scheme

for t=1:size(T,2)
%t=0;
%while t<T_final
    Q_old=Q;
    for l=1:3
        % Finding Lax-Friedrich flux
        for i=1:(size(X,2)-1)
            % Velocity of sound in their respective domains
            cl=sqrt((g*p(i))/rho(i));
            cr=sqrt((g*p(i+1))/rho(i+1));

            % Finding the maximum eigen value
            A=[abs(u(i)),abs(u(i)+cl),abs(u(i)-
cl),abs(u(i+1)),abs(u(i+1)+cr),abs(u(i+1)-cr)];
            alpha(i)=max(A);

            % Lax-Friedrich flux
            F(l,i)=0.5*(E(l,i)+E(l,i+1))-0.5*alpha(i)*(Q_old(l,i+1)-Q_old(l,i));
        end

        % Time step determination
        %dt=cfl*(dx/max(alpha));

        % Time Marching
        for j=2:size(X,2)-1
            Q(l,j)=Q_old(l,j)-(dt/dx)*(F(l,j)-F(l,j-1));
        end
    end

    % Updating the flux
    E(1,:)=Q(2,:);
    E(2,:)=((3-g)/2)*((Q(2,:).^2)./Q(1,:))+(g-1)*Q(3,:);
    E(3,:)=(Q(2,:)./Q(1,:)).*(g*Q(3,:)-((g-1)/2)*((Q(2,:).^2)./Q(1,:)));

%     E(1,:)=Q(2,:);
%     E(2,:)=((g-3)/2)*Q(1,:).*Q(2,:)+(g-1)*Q(3,:);
%     E(3,:)=(Q(2,:)./Q(1,:)).*(g*Q(3,:)-((g-1)/2)*Q(1,:).*Q(2,:));

    % Updating velcity, pressure and density vectors
    u=Q(2,:)./Q(1,:);
    rho=Q(1,:);
    et=Q(3,:)./Q(1,:);
    e=et-u.^2/2;
    p=e.*rho*(g-1);

    %t=t+dt;
end
toc

%% Exact Solution

[p_e,u_e,rho_e]=RiemannExact(1,1,0,0.1,0.125,0,1e-4);

%% Visualization

figure
plot(X,u,'r-','LineWidth',2);
hold on
plot(X,u_e,'k--');
grid on
xlabel('X');
ylabel('Velocity');
title('Velocity - Gudonov Scheme');
legend('Godunov Solution','Exact solution')

figure
plot(X,rho,'g-','LineWidth',2);
hold on
plot(X,rho_e,'k--');
grid on
xlabel('X');
ylabel('Density');
title('Density - Gudonov Scheme');
legend('Godunov Solution','Exact solution')
```

```
figure
plot(X,p,'b-','LineWidth',2);
hold on
plot(X,p_e,'k--');
grid on
xlabel('X');
ylabel('Pressure');
title('Pressure - Gudonov Scheme');
legend('Godunov Solution','Exact solution')
```

Lax Wendroff

```
%% CFD - Final Project - Lax Wendroff

%% Given Parameters

T_final=0.1644;
L=1;
dx=0.01;
g=1.4;
cfl=0.8;

%% Discretization Parameters

dt=0.001644;
T=0:dt:T_final;
X=0:dx:L;

%% Initialization

Q=zeros(3,size(X,2));
E=zeros(3,size(X,2));
F=zeros(3,size(X,2));
alpha=zeros(1,size(X,2)-1);

Q_h=zeros(3,size(X,2)-1);

%% Initial SOD conditions

Q(1,1:(0.5/dx)+1)=1;
Q(1,(0.5/dx)+2:size(X,2))=0.125;

Q(2,1:(0.5/dx)+1)=0;
Q(2,(0.5/dx)+2:size(X,2))=0;

Q(3,1:(0.5/dx)+1)=2.5;
Q(3,(0.5/dx)+2:size(X,2))=0.25;

E(1,:)=Q(2,:);
E(2,:)=((3-g)/2)*((Q(2,:).^2)./Q(1,:))+(g-1)*Q(3,:);
E(3,:)=(Q(2,:)./Q(1,:)).*(g*Q(3,:)-((g-1)/2)*((Q(2,:).^2)./Q(1,:)));

%% Velocity, Pressure and density from Q and E vectors

u=Q(2,:)./Q(1,:);
rho=Q(1,:);
et=Q(3,:)./Q(1,:);
e=et-u.^2/2;
p=e.*rho*(g-1);

tic;
%% Lax Wendroff scheme
```

```
for t=1:size(T,2)
    Q_old=Q;

    % Predictor Step
    for l=1:3
        for i=1:size(X,2)-1
            Q_h(l,i)=0.5*(Q_old(l,i)+Q_old(l,i+1))-0.5*(dt/dx)*(E(l,i+1)-E(l,i));
        end
    end

    % Recalculating the flux
    E_h(1,:)=Q_h(2,:);
    E_h(2,:)=((3-g)/2)*((Q_h(2,:).^2)./Q_h(1,:))+(g-1)*Q_h(3,:);
    E_h(3,:)=(Q_h(2,:)./Q_h(1,:)).*(g*Q_h(3,:)-((g-1)/2)*((Q_h(2,:).^2)./Q_h(1,:)));

    % Corrector Step
    for l=1:3
        for i=2:size(X,2)-1
            Q(l,i)=Q_old(l,i)-(dt/dx)*(E_h(l,i)-E_h(l,i-1));
        end
    end

    % Updating the flux
    E(1,:)=Q(2,:);
    E(2,:)=((3-g)/2)*((Q(2,:).^2)./Q(1,:))+(g-1)*Q(3,:);
    E(3,:)=(Q(2,:)./Q(1,:)).*(g*Q(3,:)-((g-1)/2)*((Q(2,:).^2)./Q(1,:)));

    % Updating velcity, pressure and density vectors
    u=Q(2,:)./Q(1,:);
    rho=Q(1,:);
    et=Q(3,:)./Q(1,:);
    e=et-u.^2/2;
    p=e.*rho*(g-1);
end

toc;
%% Exact Solution

[p_e,u_e,rho_e]=RiemannExact(1,1,0,0.1,0.125,0,1e-4);

%% Visualization

figure
plot(X,u,'r-','LineWidth',2);
hold on
plot(X,u_e,'k--');
grid on
xlabel('X');
ylabel('Velocity');
title('Velocity - Lax-Wendroff Scheme');
legend('Lax-Wendroff Solution','Exact solution')

figure
plot(X,rho,'g-','LineWidth',2);
hold on
plot(X,rho_e,'k--');
grid on
xlabel('X');
ylabel('Density');
title('Density - Lax-Wendroff Scheme');
legend('Lax-Wendroff Solution','Exact solution')

figure
```

```
plot(X,p,'b-','LineWidth',2);
hold on
plot(X,p_e,'k--');
grid on
xlabel('X');
ylabel('Pressure');
title('Pressure - Lax-Wendroff Scheme');
legend('Lax-Wendroff Solution','Exact solution')
```

Godunov with MUSCL

```
%% CFD - Final Project - Godunov with MUSCL scheme

%% Given Parameters

T_final=0.1644;
L=1;
dx=0.01;
g=1.4;
cfl=0.8;

%% Discretization Parameters

dt=0.00001644;
T=0:dt:T_final;
X=0:dx:L;

%% Initialization

Q=zeros(3,size(X,2));
E=zeros(3,size(X,2));
F=zeros(3,size(X,2));
alpha=zeros(1,size(X,2)-1);

u_l=zeros(1,size(X,2)-1);
u_r=zeros(1,size(X,2)-1);
rho_l=zeros(1,size(X,2)-1);
rho_r=zeros(1,size(X,2)-1);
p_l=zeros(1,size(X,2)-1);
p_r=zeros(1,size(X,2)-1);

Q_l=zeros(1,size(X,2)-1);
Q_r=zeros(1,size(X,2)-1);

%% Initial SOD conditions

Q(1,1:(0.5/dx)+1)=1;
Q(1,(0.5/dx)+2:size(X,2))=0.125;

Q(2,1:(0.5/dx)+1)=0;
Q(2,(0.5/dx)+2:size(X,2))=0;

Q(3,1:(0.5/dx)+1)=2.5;
Q(3,(0.5/dx)+2:size(X,2))=0.25;

E(1,:)=Q(2,:);
E(2,:)=((3-g)/2)*((Q(2,:).^2)./Q(1,:))+(g-1)*Q(3,:);
E(3,:)=(Q(2,:)./Q(1,:)).*(g*Q(3,:)-((g-1)/2)*((Q(2,:).^2)./Q(1,:)));

%% Velocity, Pressure and density from Q and E vectors

u=Q(2,:)./Q(1,:);
```

```
rho=Q(1,:);
et=Q(3,:)./Q(1,:);
e=et-u.^2/2;
p=e.*rho*(g-1);

tic
%% Godunov - MUSCL Scheme

for t=1:size(T,2)
    Q_old=Q;

    % Approximations of Q at the i+1/2 boundaries
    for i=1:size(X,2)-1
        if i==1
            Q_l(1,i)=Q(1,i);%+((Q(1,i+1)-Q(1,i))/dx)*(dx/2);
            Q_l(2,i)=Q(2,i);%+((Q(2,i+1)-Q(2,i))/dx)*(dx/2);
            Q_l(3,i)=Q(3,i);%+((Q(3,i+1)-Q(3,i))/dx)*(dx/2);

            Q_r(1,i)=Q(1,i+1)-minmod(((Q(1,i+2)-Q(1,i+1))/dx),((Q(1,i+1)-
Q(1,i))/dx))*(dx/2);
            Q_r(2,i)=Q(2,i+1)-minmod(((Q(2,i+2)-Q(2,i+1))/dx),((Q(2,i+1)-
Q(2,i))/dx))*(dx/2);
            Q_r(3,i)=Q(3,i+1)-minmod(((Q(3,i+2)-Q(3,i+1))/dx),((Q(3,i+1)-
Q(3,i))/dx))*(dx/2);
        else
            if i==size(X,2)-1
                Q_l(1,i)=Q(1,i)+minmod(((Q(1,i+1)-Q(1,i))/dx),((Q(1,i)-Q(1,i-
1))/dx))*(dx/2);
                Q_l(2,i)=Q(2,i)+minmod(((Q(2,i+1)-Q(2,i))/dx),((Q(2,i)-Q(2,i-
1))/dx))*(dx/2);
                Q_l(3,i)=Q(3,i)+minmod(((Q(3,i+1)-Q(3,i))/dx),((Q(3,i)-Q(3,i-
1))/dx))*(dx/2);

                Q_r(1,i)=Q(1,i+1);%-((Q(1,i+1)-Q(1,i))/dx)*(dx/2);
                Q_r(2,i)=Q(2,i+1);%-((Q(2,i+1)-Q(2,i))/dx)*(dx/2);
                Q_r(3,i)=Q(3,i+1);%-((Q(3,i+1)-Q(3,i))/dx)*(dx/2);
            else
                Q_l(1,i)=Q(1,i)+minmod(((Q(1,i+1)-Q(1,i))/dx),((Q(1,i)-Q(1,i-
1))/dx))*(dx/2);
                Q_l(2,i)=Q(2,i)+minmod(((Q(2,i+1)-Q(2,i))/dx),((Q(2,i)-Q(2,i-
1))/dx))*(dx/2);
                Q_l(3,i)=Q(3,i)+minmod(((Q(3,i+1)-Q(3,i))/dx),((Q(3,i)-Q(3,i-
1))/dx))*(dx/2);

                Q_r(1,i)=Q(1,i+1)-minmod(((Q(1,i+2)-Q(1,i+1))/dx),((Q(1,i+1)-
Q(1,i))/dx))*(dx/2);
                Q_r(2,i)=Q(2,i+1)-minmod(((Q(2,i+2)-Q(2,i+1))/dx),((Q(2,i+1)-
Q(2,i))/dx))*(dx/2);
                Q_r(3,i)=Q(3,i+1)-minmod(((Q(3,i+2)-Q(3,i+1))/dx),((Q(3,i+1)-
Q(3,i))/dx))*(dx/2);
            end
        end
    end

    % Formulation of E
    E_l(1,:)=Q_l(2,:);
    E_l(2,:)=((3-g)/2)*((Q_l(2,:).^2)./Q_l(1,:))+(g-1)*Q_l(3,:);
    E_l(3,:)=(Q_l(2,:)./Q_l(1,:)).*(g*Q_l(3,:)-((g-1)/2)*((Q_l(2,:).^2)./Q_l(1,:)));

    E_r(1,:)=Q_r(2,:);
    E_r(2,:)=((3-g)/2)*((Q_r(2,:).^2)./Q_r(1,:))+(g-1)*Q_r(3,:);
    E_r(3,:)=(Q_r(2,:)./Q_r(1,:)).*(g*Q_r(3,:)-((g-1)/2)*((Q_r(2,:).^2)./Q_r(1,:)));
    for l=1:3
        % Finding Lax-Friedrich flux
        for i=1:(size(X,2)-1)
            % Velocity of sound in their respective domains
            cl=sqrt((g*p_l(i))/rho_l(i));
            cr=sqrt((g*p_r(i))/rho_r(i));

            % Finding the maximum eigen value
            A=[abs(u(i)),abs(u(i)+cl),abs(u(i)-
cl),abs(u(i+1)),abs(u(i+1)+cr),abs(u(i+1)-cr)];
            alpha(i)=max(A);

            % Lax-Friedrich flux
            F(l,i)=0.5*(E_l(l,i)+E_r(l,i))-0.5*alpha(i)*(Q_r(l,i)-Q_l(l,i));
        end

        % Time Marching
        for j=2:size(X,2)-1
            Q(l,j)=Q_old(l,j)-(dt/dx)*(F(l,j)-F(l,j-1));
        end
    end

    % Updating the flux
    E(1,:)=Q(2,:);
    E(2,:)=((3-g)/2)*((Q(2,:).^2)./Q(1,:))+(g-1)*Q(3,:);
    E(3,:)=(Q(2,:)./Q(1,:)).*(g*Q(3,:)-((g-1)/2)*((Q(2,:).^2)./Q(1,:)));

    % Updating velcity, pressure and density vectors
    u=Q(2,:)./Q(1,:);
    rho=Q(1,:);
    et=Q(3,:)./Q(1,:);
    e=et-u.^2/2;
    p=e.*rho*(g-1);
end
toc

%% Exact Solution

[p_e,u_e,rho_e]=RiemannExact(1,1,0,0.1,0.125,0,1e-4);

%% Visualization

figure
plot(X,u,'r-','LineWidth',2);
hold on
plot(X,u_e,'k--');
grid on
xlabel('X');
ylabel('Velocity');
title('Velocity - Godunov with MUSCL Scheme');
legend('Godunov with MUSCL Solution','Exact solution')

figure
plot(X,rho,'g-','LineWidth',2);
hold on
plot(X,rho_e,'k--');
grid on
xlabel('X');
ylabel('Density');
title('Density - Godunov with MUSCL Scheme');
legend('Godunov with MUSCL Solution','Exact solution')
```

```
figure
plot(X,p,'b-','LineWidth',2);
hold on
plot(X,p_e,'k--');
grid on
xlabel('X');
ylabel('Pressure');
title('Pressure - Godunov with MUSCL Scheme');
legend('Godunov with MUSCL Solution','Exact solution')
```

Function : minmod

```
%% Function - minmod

function s=minmod(a,b)
    if a*b>0
        s=(a/abs(a))*min(abs(a),abs(b));
    else
        s=0;
    end
end
```

Function : ReimannExact (Taken from the internet)

```
function [P,U,rho] = RiemannExact(p1,rho1,u1,p4,rho4,u4,tol)

%This code gives the exact solution of Riemann's 1-D problem in the context
%of the unsteady 1-D shock tube,by means of the Newton-Raphson's method.
%After having set the inputs to the left and right
%gases' variables, one can easily check the waves generated from the
%discontinuity; the program returns in output the type of the solution,
%which can be for example an RCS (meaning that an expansion has generated
%on the first family and a shock on the second family). The user then has
%the chance to get the most important variables (sound speed, pressure,
%velocity) plotted versus the shock tube's length coordinate. NB: all
%interactions are neglected.
%
%INPUTS:
%
%p1: pressure of the gas in the left part of the shock tube
%rho1: density of the gas in the left part of the shock tube
%u1: velocity of the particles in the left part of the shock tube
%p4: pressure of the gas in the right part of the shock tube
%rho4: density of the gas in the right part of the shock tube
%u4: velocity of the particles in the right part of the shock tube
%tol: tolerance of solution
%
%Virginia Notaro


iter=0;
gamma=1.4;
delta=(gamma-1)/2;
alfa=2*gamma/(gamma-1);

a1=sqrt(gamma*p1/rho1);
a4=sqrt(gamma*p4/rho4);

zeta=(p1/p4)^(1/alfa)*(a1/a4); %Entropy difference between gas 1 and gas 4

if u4-u1>a4/delta+a1/delta
```

```
    disp('Warning: Solution does not exist!')
else

    %First approximation velocity
    ustar=(zeta*(a1+delta*u1)-a4+delta*u4)/((1+zeta)*delta);
    u=ustar;
    p2=1;
    p3=0.5;

    while abs(p2-p3)>tol

    %Checking wave on first family
    if u>u1  %R
    a2=a1+delta*(u1-u);
    p2=p1*(a2/a1)^alfa;
    dp2=-gamma*p2/a2;

    elseif u<u1 %S
    V1=(gamma+1)/4*(u1-u)+sqrt(((gamma+1)/4)^2*(u1-u)^2+a1^2);
    M1=V1/a1;
    p2=p1*(1+(2*gamma)/(gamma+1)*(M1^2-1));
    a2=a1*sqrt((gamma+1+(gamma-1)*(p2/p1))/(gamma+1+(gamma-1)*(p1/p2)));
    ws=u1-V1;
    M2=(u-ws)/a2;
    dp2=-2*gamma*(p1/a1)*((abs(M1))^3/(1+M1^2));



    end

    %Checking wave on the second family
    if u>u4 %S
    V4=(gamma+1)/4*(u-u4)+sqrt(((gamma+1)/4)^2*(u-u4)^2+a4^2);
    M4=V4/a4;
    p3=p4*(1+(2*gamma)/(gamma+1)*(M4^2-1));
    V3=V4*(1+delta*M4^2)/((gamma+1)/2*M4^2);
    a3=a4*sqrt((gamma+1+(gamma-1)*(p3/p4))/(gamma+1+(gamma-1)*(p4/p3)));
    wd=u4-V4;
    M3=(u-wd)/a3;
    dp3=2*gamma*(p4/a4)*((abs(M4))^3/(1+M4^2));

    elseif u<u4 %R
    a3=a4+delta*(u-u4);
    p3=p4*(a3/a4)^alfa;
    dp3=gamma*p3/a3;

    end

    %New velocity and update of the iteration
    u=u-(p2-p3)/(dp2-dp3);
    iter=iter+1;

    %Checking number of iterations
    if iter>45000
        disp('The problem can not be solved using this method!')
        break
    end
    end

    %End of cycle and density on the right (3) and left (2) side of the
    %contact discontinuity

    rho2=gamma*p2/a2^2;
```

```
rho3=gamma*p3/a3^2;


%Type of the solution


if ((u>u1)&&(abs(p2-p1)>tol)&&(abs(p2-p4)>tol))
    if (u<u4)
        tipo=('RCR');
        disp('The solution is RCR')

    else
        tipo=('RCS');
        disp('The solution is RCS')
        disp('wd= '),disp(wd)
    end
elseif ((u<u1)&&(abs(p2-p1)>tol)&&(abs(p2-p4)>tol))
    if (u<u4)
        tipo=('SCR');
        disp('The solution is SCR')
        disp('ws= '),disp(ws)
    else
        tipo=('SCS');
        disp('The solution is SCS')
        disp('ws= '),disp(ws)
        disp('wd= '),disp(wd)
    end
elseif ((abs(u-u1)<tol)&&(abs(p2-p1)<tol)&&(abs(u-u4)>tol))
    if (u<u4)
        tipo=('NCR');
        disp('The solution is NCR')
        u=u1;
        p2=p1;
        p3=p1;
        a2=a1;

        rho2=rho1;


    else
        tipo=('NCS');
        disp('The solution is NCS')
        u = u1;
        p2 = p1;
        p3 = p1;
        a2 = a1;
        disp('wd= '),disp(wd)

        rho2 = rho1;


    end
elseif ((abs(u-u1)>tol)&&(abs(p2-p4)<tol)&&(abs(u-u4)<tol))
    if (u>u1)
        tipo=('RCN');
        disp('The solution is RCN')
        u = u4;
        p2 = p4;
        p3 = p4;
        a3 = a4;
        rho3 = rho4;

    else
```

```
            tipo=('SCN');
            disp('The solution is SCN')
            u = u4;
            p2 = p4;
            p3 = p4;
            disp('ws= '),disp(ws)

            a3 = a4;
            rho3 = rho4;

        end
    else
        tipo=('NCN');
        disp('There isn''t any wave! NCN')
        u = u1;
        p2 = p1; p3 = p1;
        a2 = a1; a3 = a1;
        rho2 = rho1; rho3 = rho1;

    end

    disp('u= '),disp(u)
    disp('p2= '),disp(p2)
    disp('p3= '),disp(p3)
    disp('rho2= '),disp(rho2)
    disp('rho3= '),disp(rho3)


end

    disp('The number of iterations is: '),disp(iter)


%Graphical representation of pressure, velocity and sound speed versus the
%position x in the shock tube.

t= 0.1644 %input('Insert the time which you want to calculate the solution at (check
the velocities'' order of magnitude first!), t= ');
x0=-0.5 %input('Insert the lower bound of the visualization domain, x0= ');
xf= 0.5 %input('Insert the upper bound of the visualization domain, xf= ');
npoints= 101 %input('insert the number of points which you want to calculate the
solution in, npoints= ');

x=linspace(x0,xf,npoints);

A = zeros(npoints,1);
P = zeros(npoints,1);
U = zeros(npoints,1);


x1=(u1-a1)*t;
x2=(u-a2)*t;
x3=u*t;
x4=(u+a3)*t;
x5=(u4+a4)*t;

%Analytical solution by means of the characteristic equations

if tipo==('RCR')

    for k=1:npoints
        if x(k)<=x1
```

```
            U(k)=u1;
            P(k)=p1;
            A(k)=a1;

        elseif x(k)<=x2 && x(k)>=x1

            U(k)=1/(1+delta)*(a1+delta*u1+(x(k))/t);
            A(k)=1/(1+delta)*(a1)+delta/(delta+1)*(u1)-delta/(1+delta)*(x(k))/t;
            P(k)=p1.*(A(k)/a1).^(alfa);

        elseif x(k)>=x2 && x(k)<=x3

            U(k)=u;
            A(k)=a2;
            P(k)=p2;

        elseif x(k)>=x3 && x(k)<=x4

            U(k)=u;
            A(k)=a3;
            P(k)=p3;

        elseif x(k)>=x4 && x(k)<=x5

            U(k)=1/(1+delta)*((x(k))/t-a4+delta*u4);
            A(k)=1/(1+delta)*(a4)-delta/(1+delta)*(u4)+delta/(1+delta)*(x(k))/t;
            P(k)=p4.*(A(k)./a4).^(alfa);

        elseif x(k)>=x5

            U(k)=u4;
            P(k)=p4;
            A(k)=a4;
        end
    end
elseif tipo==('RCS')

    xUD=abs(wd)*t;

    for k=1:npoints
        if x(k)<=x1

            U(k)=u1;
            P(k)=p1;
            A(k)=a1;

        elseif x(k)<=x2 && x(k)>=x1

            U(k)=1/(1+delta)*(a1+delta*u1+(x(k))/t);
            A(k)=1/(1+delta)*(a1)+delta/(delta+1)*(u1)-delta/(1+delta)*(x(k))/t;
            P(k)=p1.*(A(k)./a1).^(alfa);

        elseif x(k)>=x2 && x(k)<=x3

            U(k)=u;
            A(k)=a2;
            P(k)=p2;

        elseif x(k)>=x3 && x(k)<=xUD

            U(k)=u;
            P(k)=p3;
            A(k)=a3;
        elseif x(k)>=xUD

            U(k)=u4;
            P(k)=p4;
            A(k)=a4;
        end
    end

elseif tipo==('SCR')

    xUS=-abs(ws)*t;

    for k=1:npoints;

        if x(k)<=xUS
            U(k)=u1;
            P(k)=p1;
            A(k)=a1;

        elseif x(k)>=xUS && x(k)<=x3

            U(k)=u;
            P(k)=p2;
            A(k)=a2;

        elseif x(k)>=x3 && x(k)<=x4

            U(k)=u;
            A(k)=a3;
            P(k)=p3;

        elseif x(k)>=x4 && x(k)<=x5

            U(k)=1/(1+delta)*((x(k))/t-a4+delta*u4);
            A(k)=1/(1+delta)*(a4)-delta/(delta+1)*(u4)+delta/(1+delta)*(x(k))/t;
            P(k)=p4.*(A(k)./a4).^(alfa);

        elseif x(k)>=x5

            U(k)=u4;
            P(k)=p4;
            A(k)=a4;
        end
    end

elseif tipo==('SCS')

    xUS=-abs(ws)*t;

    xUD=abs(wd)*t;

    for k=1:npoints
        if x(k)<=xUS
            U(k)=u1;
            P(k)=p1;
            A(k)=a1;

        elseif x(k)>=xUS && x(k)<=x3

            U(k)=u;
            P(k)=p2;
            A(k)=a2;
        elseif x(k)>=x3 && x(k)<=xUD

            U(k)=u;
            P(k)=p3;
            A(k)=a3;

        elseif x(k)>=xUD

            U(k)=u4;
            P(k)=p4;
            A(k)=a4;
        end
    end

elseif tipo==('NCR')

    for k=1:npoints
        if x(k)<=x3

            U(k)=u1;
            A(k)=a1;
            P(k)=p1;

        elseif x(k)>=x3 && x(k)<=x4

            U(k)=u;
            A(k)=a3;
            P(k)=p3;

        elseif x(k)>=x4 && x(k)<=x5

            U(k)=1/(1+delta)*((x(k))/t-a4+delta*u4);
            A(k)=1/(1+delta)*(a4)-delta/(delta+1)*(u4)+delta/(1+delta)*(x(k))/t;
            P(k)=p4.*(A(k)./a4).^(alfa);

        elseif x(k)>=x5

            U(k)=u4;
            P(k)=p4;
            A(k)=a4;

        end
    end

elseif tipo==('NCS')

    xUD=abs(wd)*t;

    for k=1:npoints
        if x(k)<=x3

            U(k)=u1;
            A(k)=a1;
            P(k)=p1;

        elseif x(k)>=x3 && x(k)<=xUD

            U(k)=u;
            P(k)=p3;
            A(k)=a3;

        elseif x(k)>=xUD
```

```
            U(k)=u4;
            P(k)=p4;
            A(k)=a4;

        end
    end

elseif tipo==('RCN')

    for k=1:npoints
        if x(k)<=x1

            U(k)=u1;
            P(k)=p1;
            A(k)=a1;

        elseif x(k)>=x1 && x(k)<=x2

            U(k)=1/(1+delta)*(a1+delta*u1+(x(k))/t);
            A(k)=1/(1+delta)*(a1)+delta/(delta+1)*(u1)-delta/(1+delta)*(x(k))/t;
            P(k)=p1.*(A(k)./a1).^(alfa);

        elseif x(k)>=x2 && x(k)<=x3

            U(k)=u;
            A(k)=a2;
            P(k)=p2;

        elseif x(k)>=x3

            U(k)=u4;
            A(k)=a4;
            P(k)=p4;

        end
    end

elseif tipo==('SCN')

    xUS=-abs(ws)*t;

    for k=1:npoints

        if x(k)<=xUS

            U(k)=u1;
            P(k)=p1;
            A(k)=a1;

        elseif x(k)>=xUS && x(k)<=x3

            U(k)=u;
            P(k)=p2;
            A(k)=a2;

        elseif x(k)>=x3

            U(k)=u4;
            A(k)=a4;
            P(k)=p4;

        end
    end

elseif tipo==('NCN')

    for k=1:npoints

        U(k)=u1;
        P(k)=p1;
        A(k)=a1;

    end
end

rho = gamma*P./(A.^2);
%Data plot

% grid on
%
% figure(1)
% plot(x,U,'-b','LineWidth',2)
% xlabel('Position')
% ylabel('Velocity')
%
% figure(2)
% plot(x,P,'-b','LineWidth',2)
% xlabel('Position')
% ylabel('Pressure')
%
% figure(3)
% plot(x,A,'-b','LineWidth',2)
% xlabel('Position')
% ylabel('Sound speed')
%
% figure(4)
% plot(x,rho, '-b', 'Linewidth', 2)
% xlabel('Position')
% ylabel('Density')
end
```