



University of Illinois at Urbana-Champaign

Department of Aerospace Engineering

AE - 410 Computational Aerodynamics

## **AE 410 Final Project 2024**

**Abhyudaya Singh**  
**Department of Aerospace Engineering**

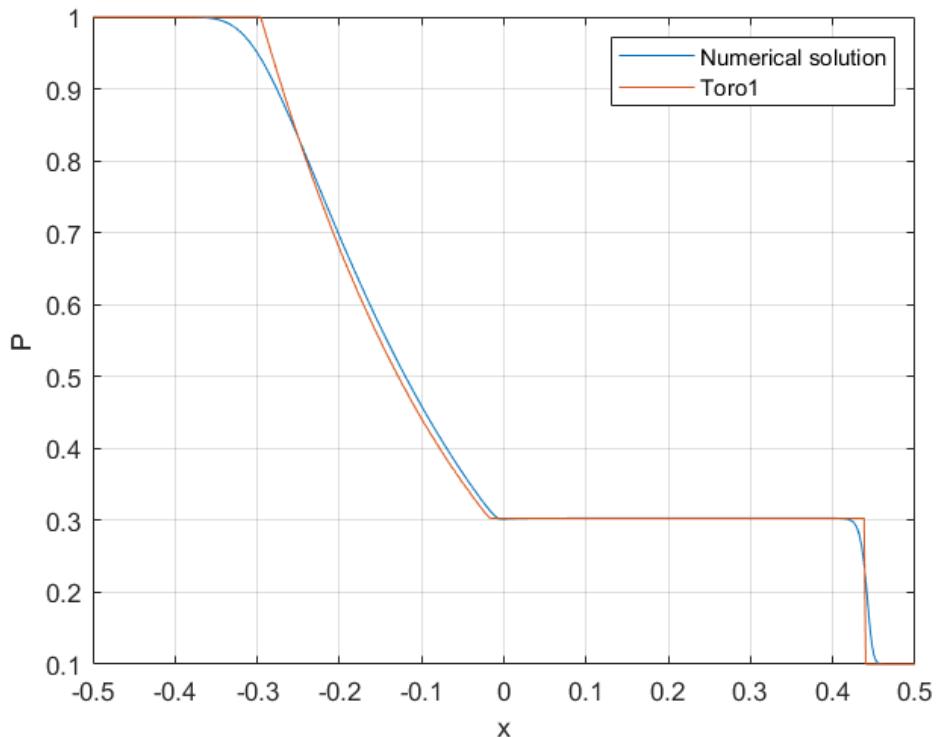
6<sup>th</sup> May 2024

**1).** A finite volume solver on MATLAB was developed to solve the one-dimensional Euler equations. The Godunov method was implemented to construct the fluxes at the faces. The code was used to solve the one-dimensional shock tube problem.

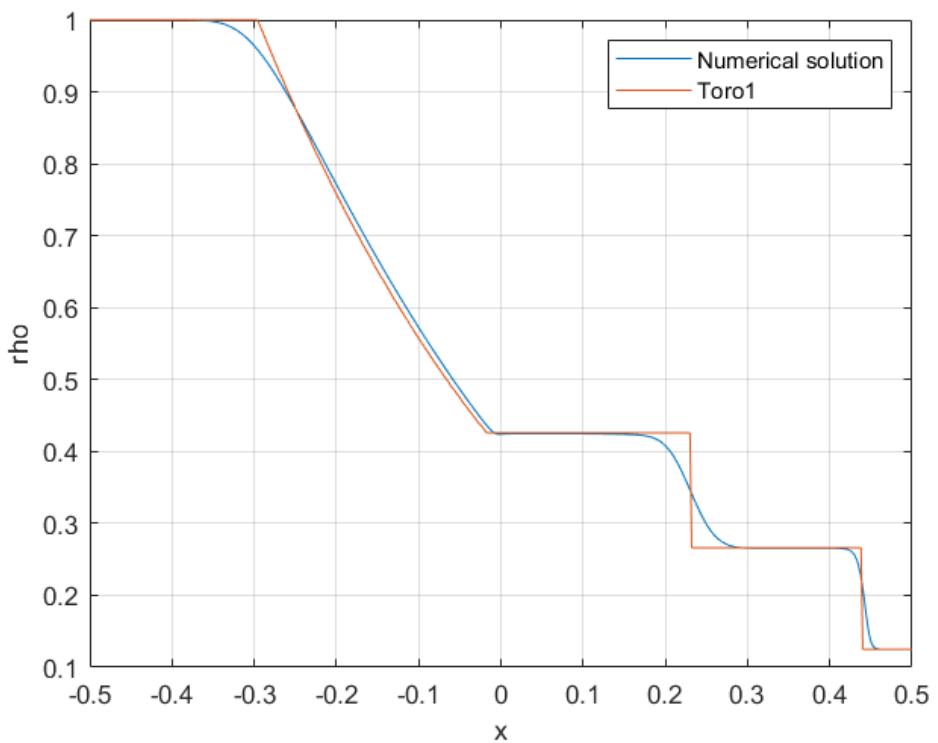
The code of the solver is provided in the appendix section.

**2).**

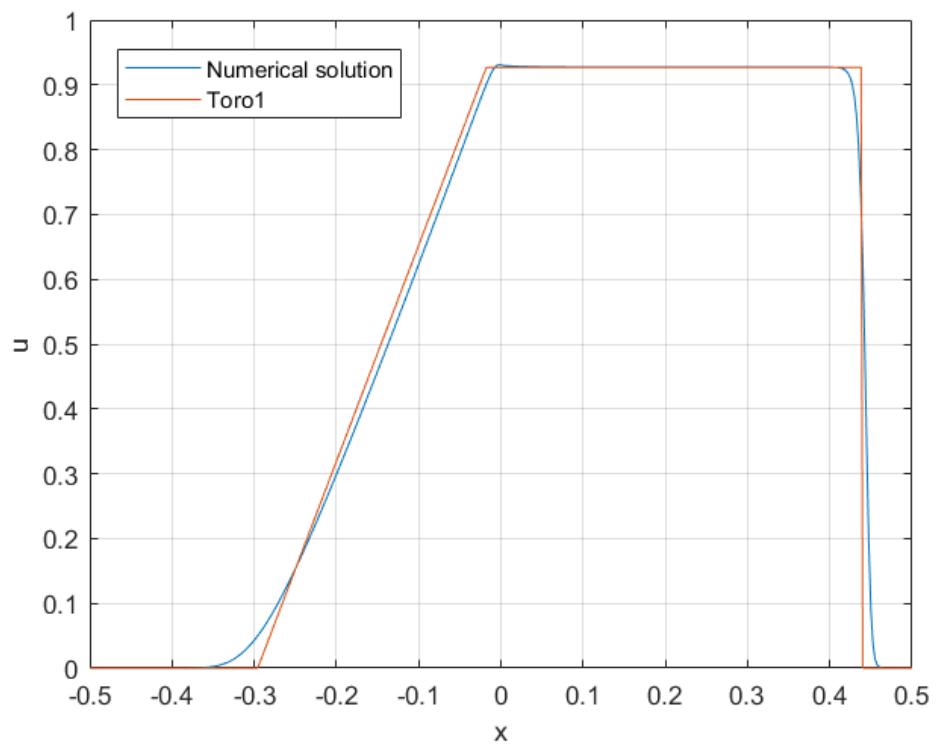
To verify the solver, the results were compared against tests 1 and 2 of Toro. Figures 1 to 6 show the comparison between the numerical results and the test results. For both cases, the number of cells and timestep size were taken to be 500 and 0.0001s, respectively. A good match between the numerical and Toro's result was observed.



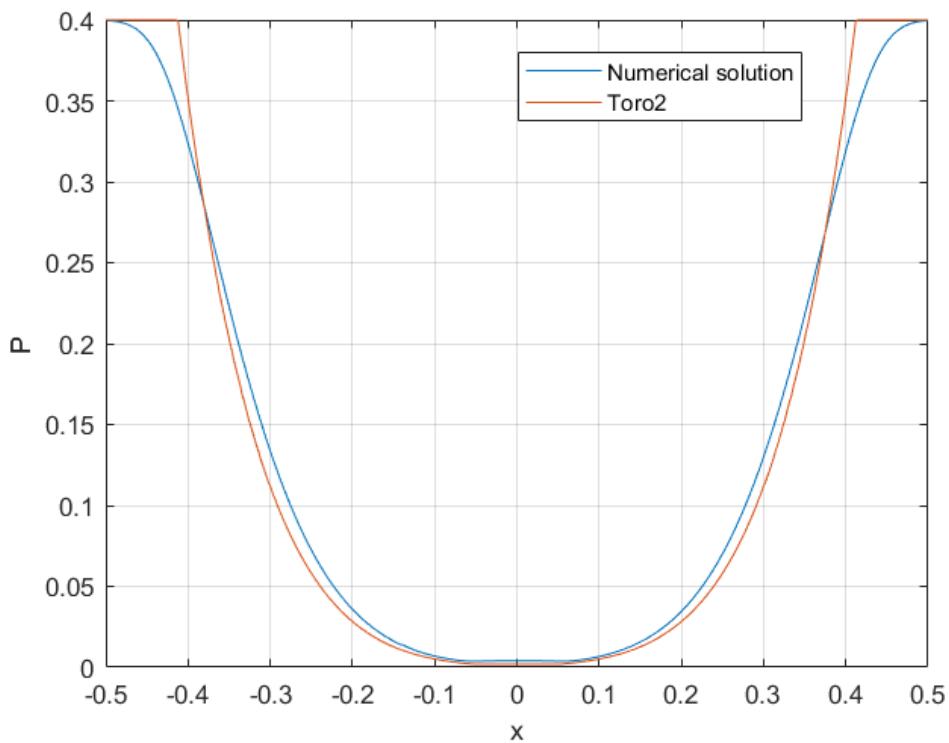
**Figure 1:** Comparison of pressure variation obtained by the numerical solutions against Toro's test1



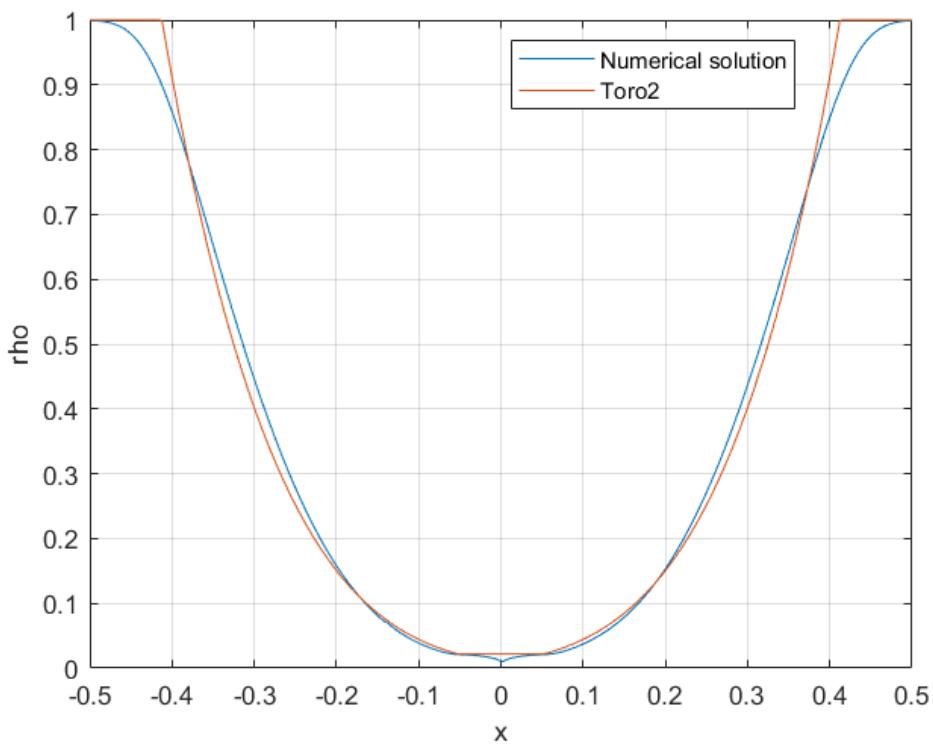
**Figure 2: Comparison of density variation obtained by the numerical solutions against Toro's test1**



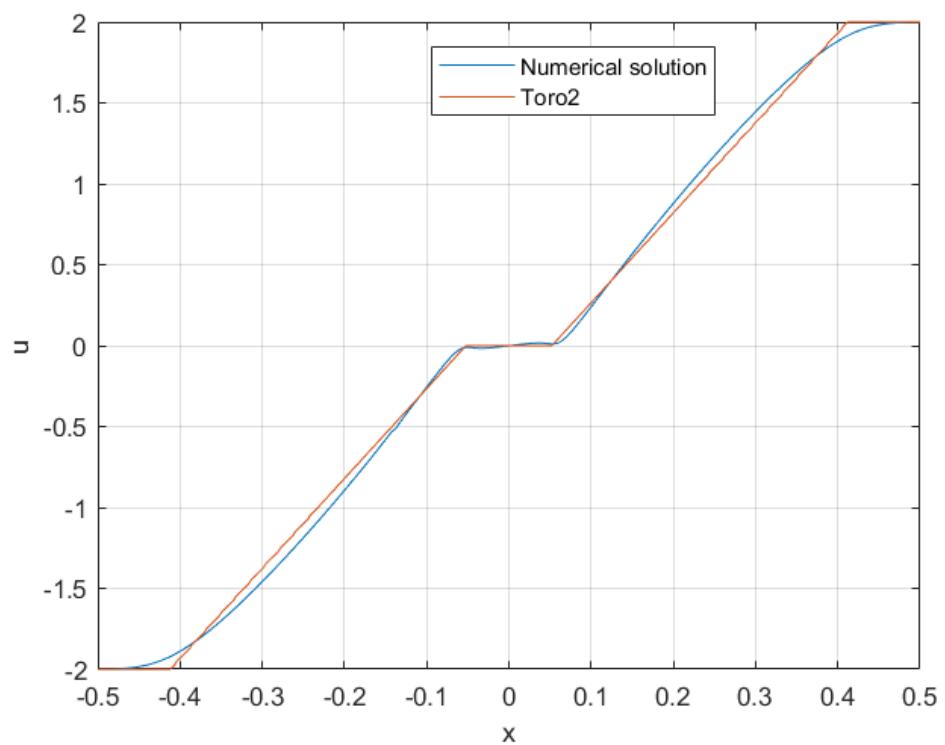
**Figure 3: Comparison of velocity variation obtained by the numerical solutions against Toro's test1**



**Figure 4: Comparison of pressure variation obtained by the numerical solutions against Toro's test2**



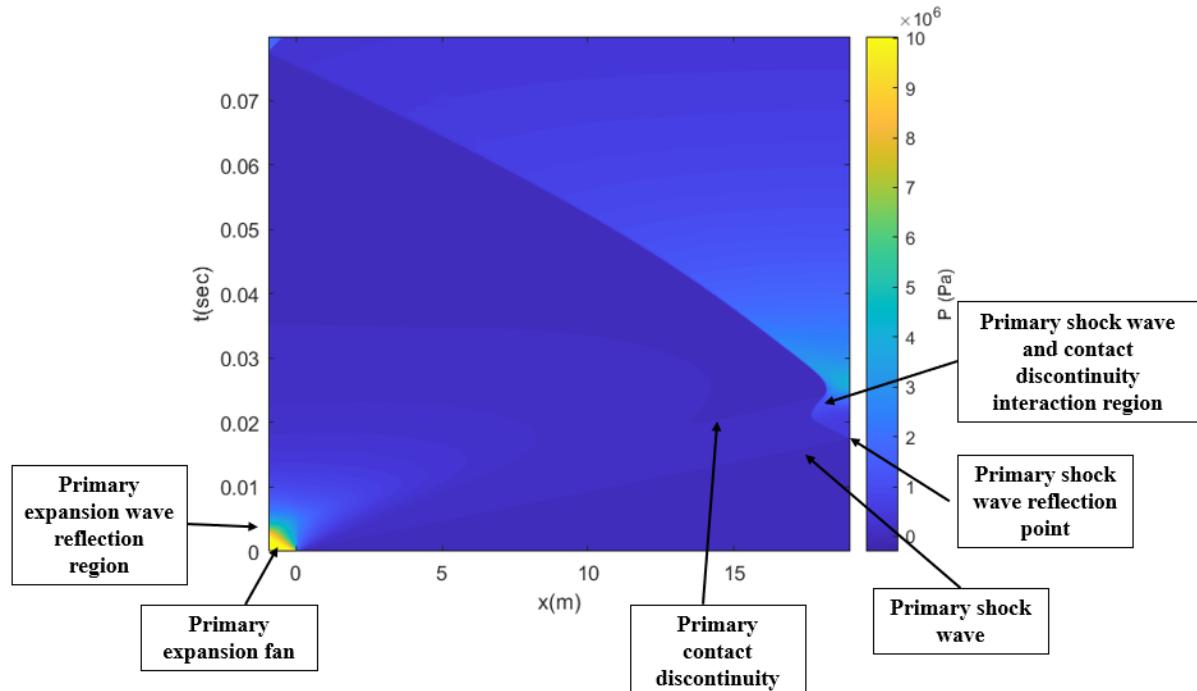
**Figure 5: Comparison of density variation obtained by the numerical solutions against Toro's test2**



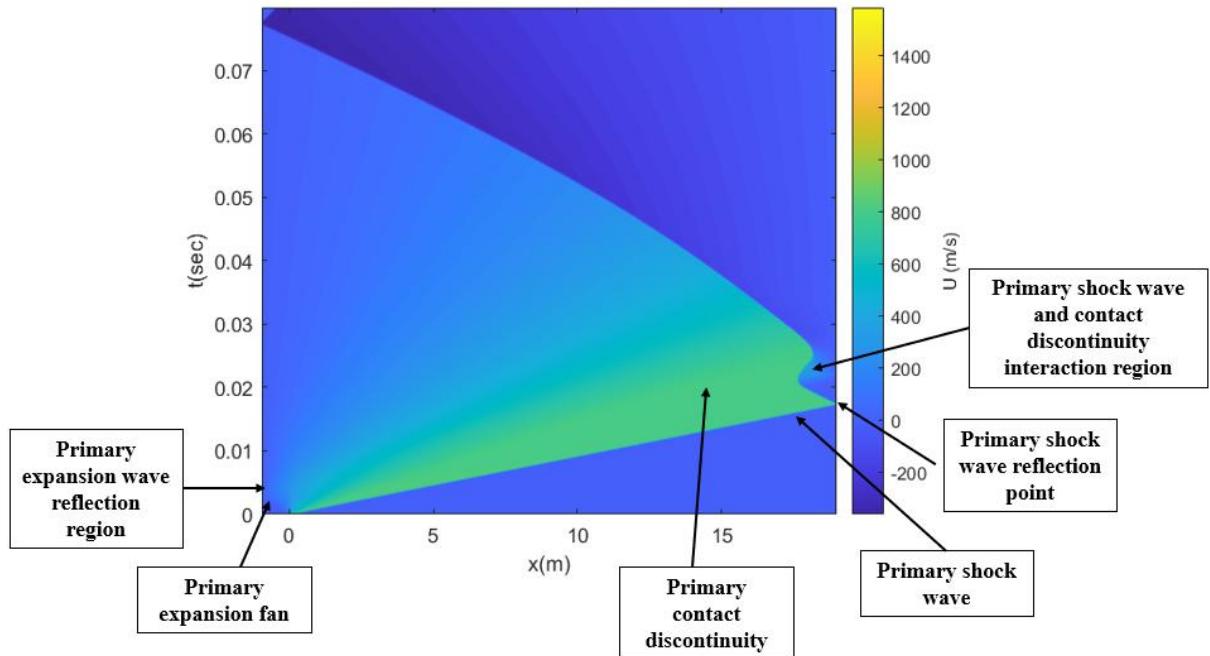
**Figure 6: Comparison of velocity variation obtained by the numerical solutions against Toro's test2**

3).

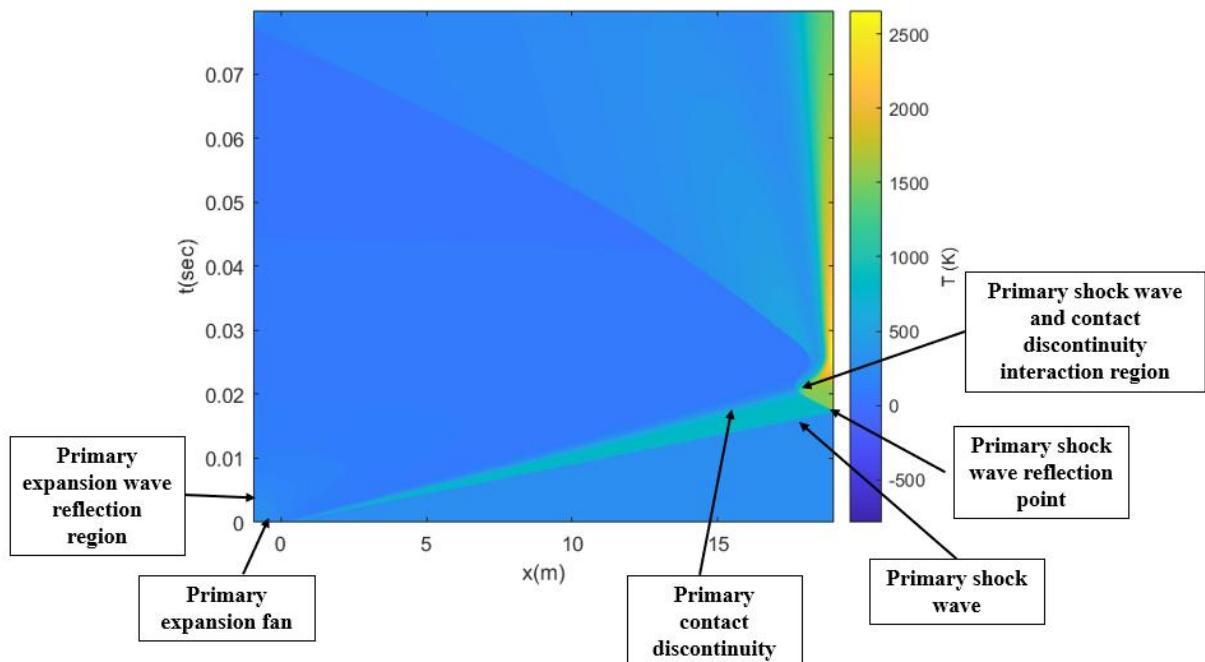
Figures 7 to 10 show the  $t$  vs  $x$  plot of Pressure, Velocity, density, and Temperature, respectively, along with the observed flow features. A grid spacing of 0.04m and time step size of  $10^{-5}$ s was chosen for this case.



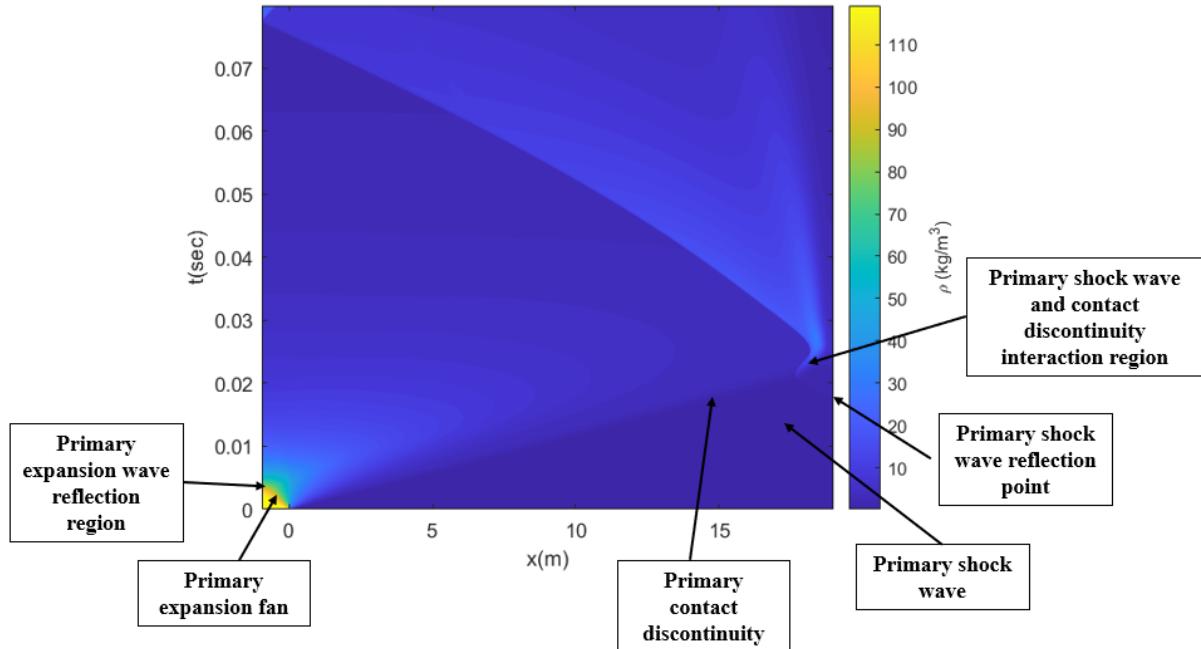
*Figure 7:  $t$  vs  $x$  diagram showing variation of pressure with time*



*Figure 8:  $t$  vs  $x$  diagram showing variation of velocity with time*



*Figure 9:  $t$  vs  $x$  diagram showing variation of density with time*



**Figure 10:**  $t$  vs  $x$  diagram showing the variation of density with time

Figure 11 shows the shock location at two different time instances. Using this, the shock speed can be calculated:

$$S = \frac{x_2 - x_1}{t_2 - t_1} = \frac{x_2 - x_1}{y_2 - y_1} = \frac{16.04 - 10.36}{0.0145 - 0.0094}$$

$$S=1113.72\text{m/s}$$

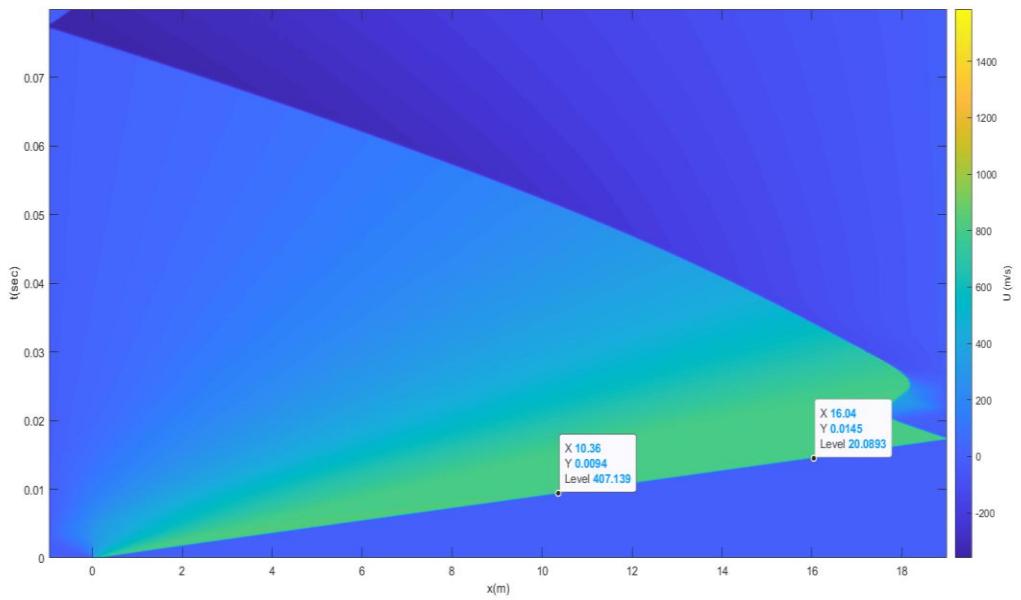
From initial conditions, we can calculate the speed of sound in the driven section as:

$$a_1=(\gamma RT_1)^{0.5}$$

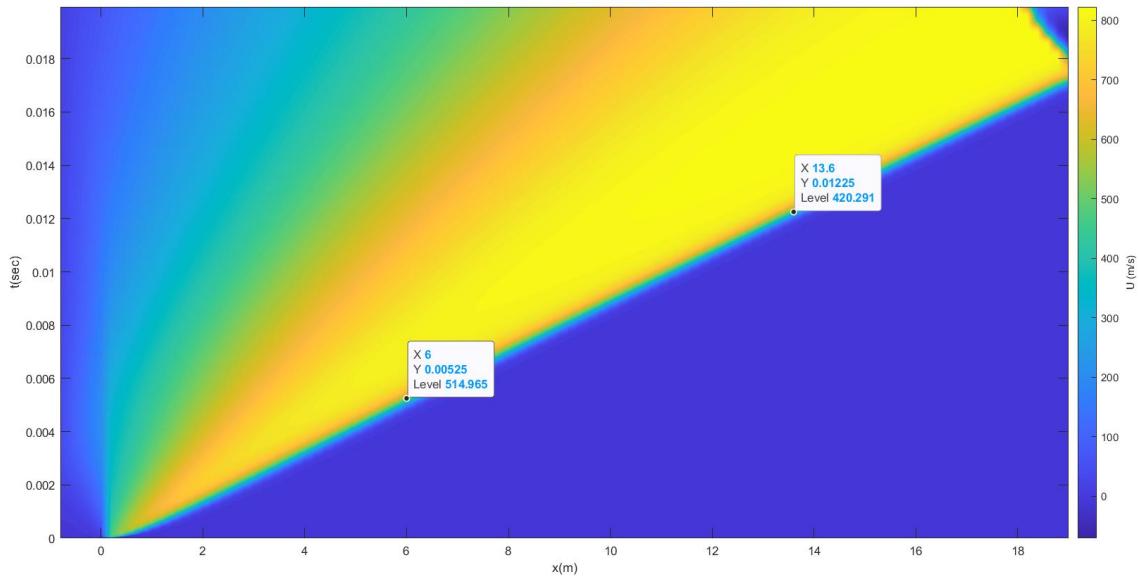
$$a_1=343.1142\text{m/s}$$

Hence, shock Mach number  $M_s$  will come out to be  $M_s=3.246$ .

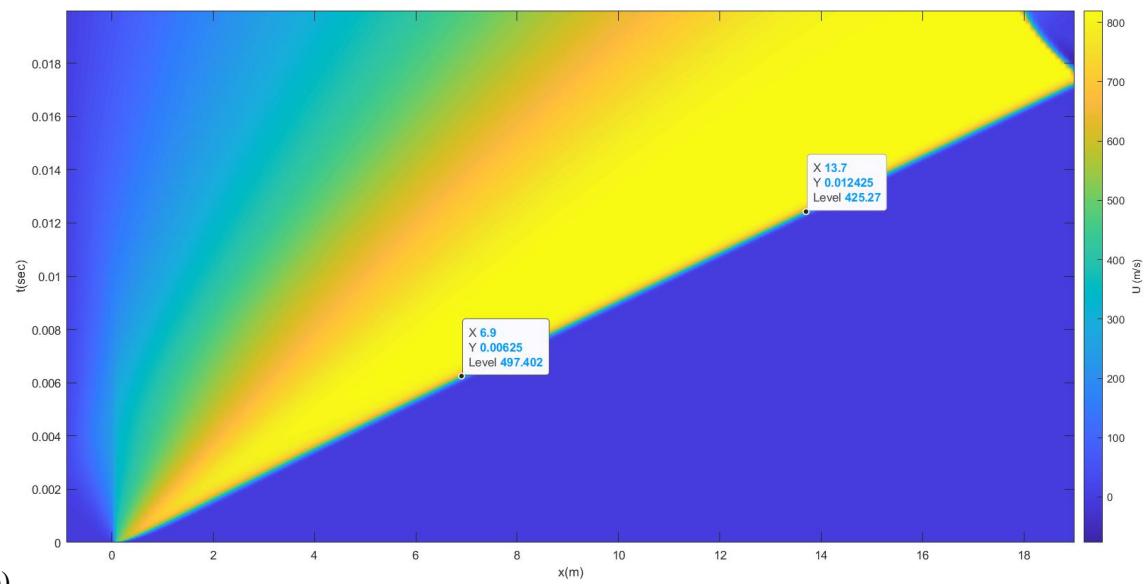
Now on solving the shock-tube equation (using fzero function on MATLAB), **the analytical value of the shock Mach number comes out to be 3.1505.**



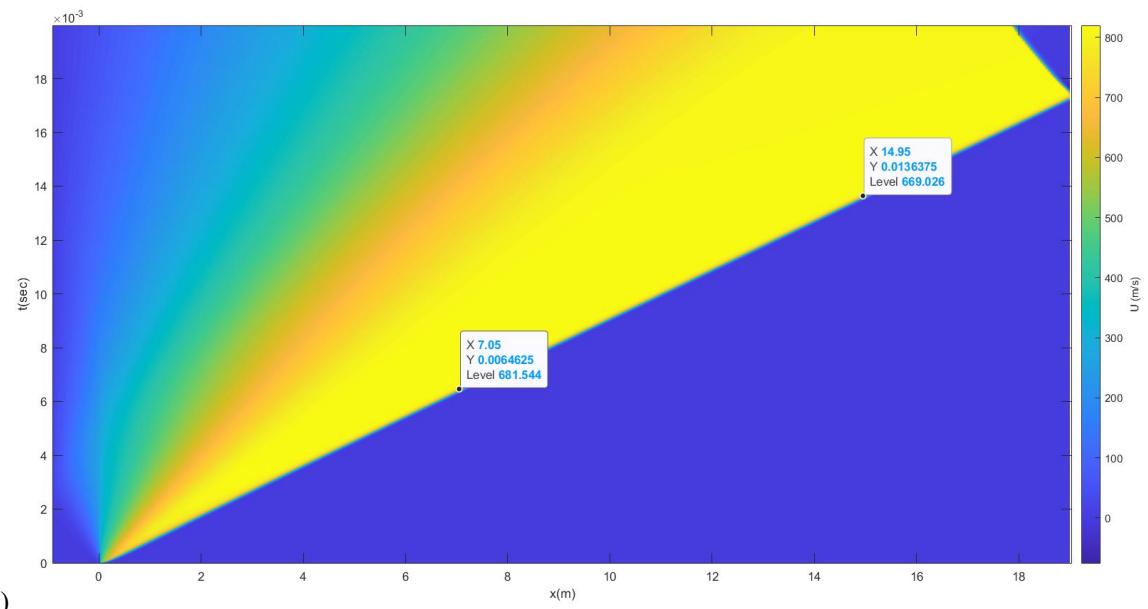
**Figure 11:**  $t$  vs  $x$  diagram showing shock location for two different time instances



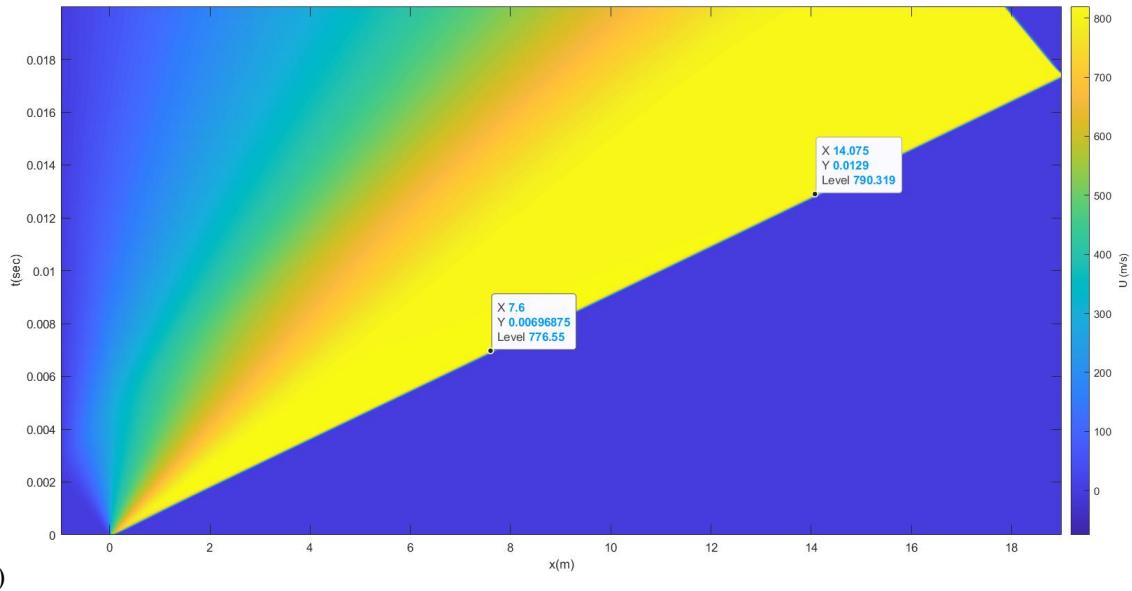
a)



b)



c)



d)

**Figure 12:  $t$  vs  $x$  diagram showing shock location for two different time instances for a)  $\Delta x=0.2\text{m}$   $\Delta t=0.00005\text{s}$ , b)  $\Delta x=0.1\text{m}$   $\Delta t=0.000025\text{s}$ , c)  $\Delta x=0.05\text{m}$   $\Delta t=0.0000125\text{s}$ , and d)  $\Delta x=0.01\text{m}$   $\Delta t=0.00000625\text{s}$**

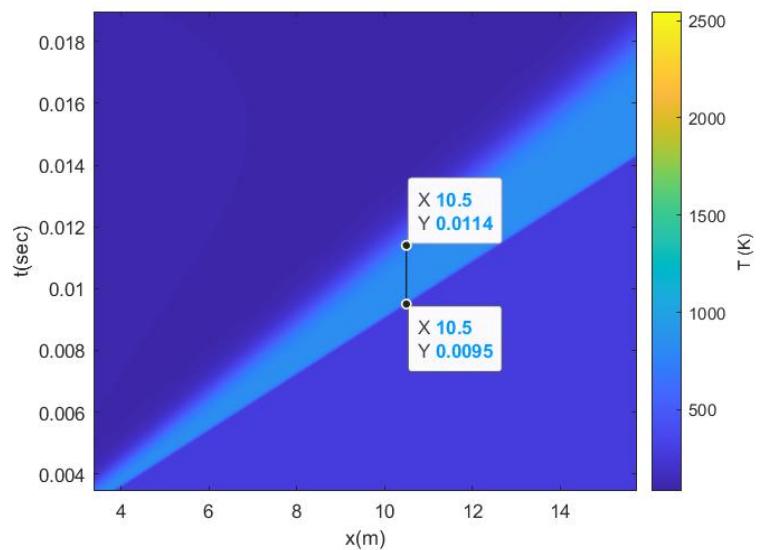
From Figures 12 a to 12 d, the shock Mach numbers of cases with different grid spacing and time step sizes are calculated and shown in the table below

Case	Analytical	$\Delta x=0.2\text{m}$ $\Delta t=0.00005\text{s}$	$\Delta x=0.1\text{m}$ $\Delta t=0.000025\text{s}$	$\Delta x=0.05\text{m}$ $\Delta t=0.0000125\text{s}$	$\Delta x=0.01\text{m}$ $\Delta t=0.00000625\text{s}$
$M_s$	3.1505	3.1643	3.2094	3.2089	3.1816

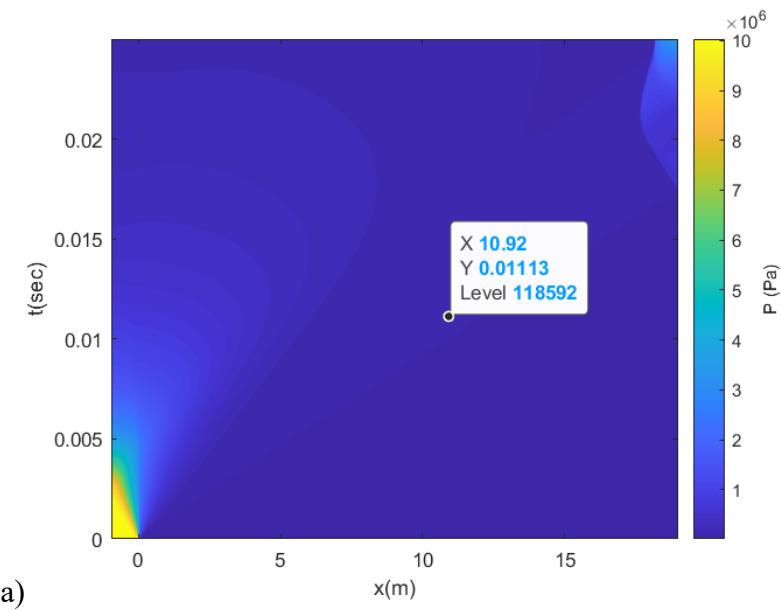
Figure 13 shows the  $t$  vs diagram for Temperature, clearly indicating the primary shock and the contact discontinuity. A vertical line is plotted at  $x=10.5\text{m}$ , that is, at the model location. The line is plotted between the primary shock and the primary contact discontinuity. Using the endpoints of the line, the test time can be found, which comes to be  $1.9 \times 10^{-3}\text{s}$  or 1.9ms.

From figures 14 a) to 14 d), the post-shock conditions can be extracted, which come out to be:

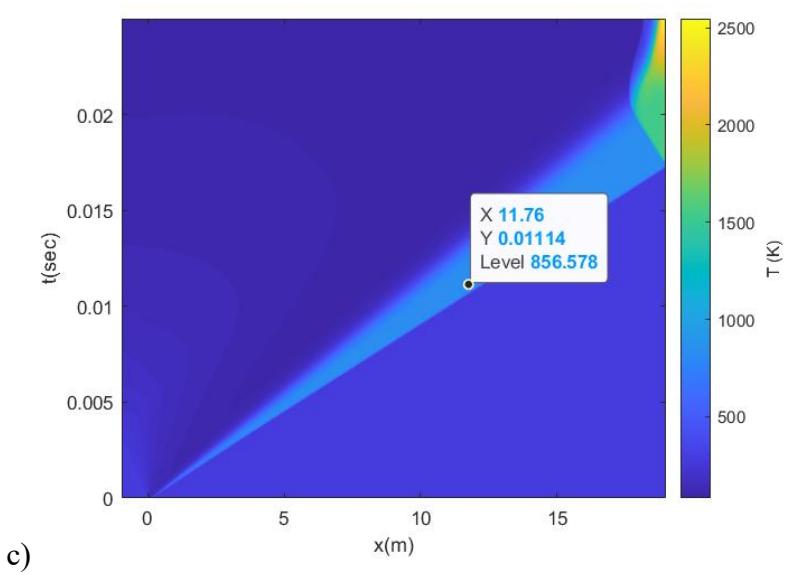
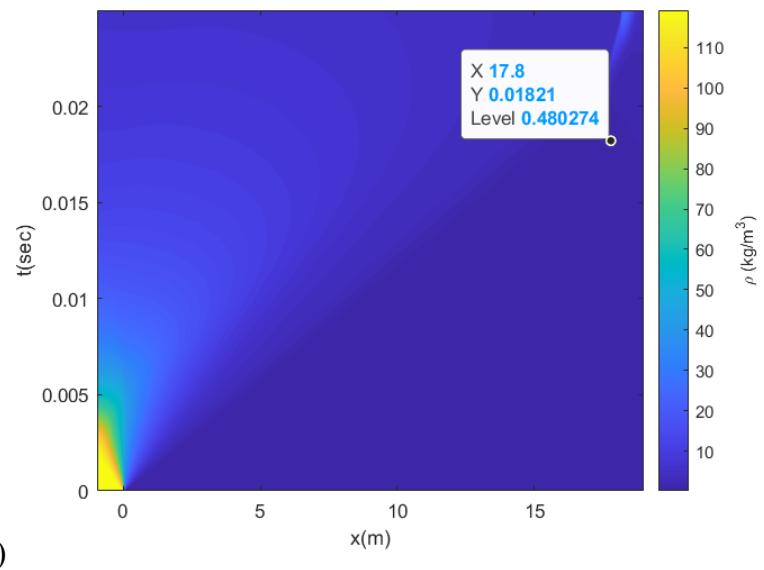
Property	Value
Pressure	118592 Pa
Temperature	856.578 K
Density	0.4802 kg/m <sup>3</sup>
Velocity	824.028 m/s
Mach number	1.404

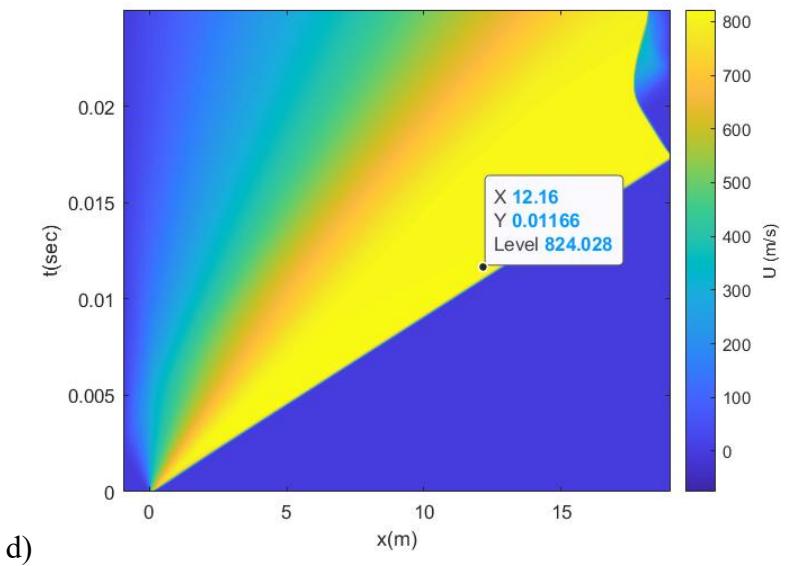


**Figure 13:  $t$  vs  $x$  diagram for Temperature showing the test time region at the model location**



a)





**Figure 14:**  $t$  vs  $x$  diagram for a) Pressure, b) Density, c) Temperature, and d) Velocity indicating the post-shock or the test conditions.

## Appendix 1: Main

```
clc
clear
close all

% Author      : Abhyudaya Singh
% UIN         : 655691216
% NetID       : singh124@illinois.edu

%% Main

fprintf('\nEnter 1 for solving Toro case1 in Problem 2
\nEnter 2 for solving Toro case2 in Problem 2 \nEnter 3
for solving Problem 3 \nEnter 4 to enter custom
conditions')
ch=input ('\nEnter choice');

switch ch
    case 1
        L1=0.5;
        L4=0.5;
        BC1="SW";
        BC4="SW";
        qlin=[1,1/287,0];
        q4in=[0.1,0.1/(287*0.125),0];
        Nx=500;
        T=0.25;
        delt=0.0001;

        FUN_Finite_Volume_Solver(L1,BC1,qlin,L4,BC4,q4in,Nx,T,del
t,ch);
    case 2
        L1=0.5;
        L4=0.5;
        BC1="SW";
        BC4="SW";
        qlin=[0.4,0.4/287,-2];
        q4in=[0.4,0.4/287,2];
        Nx=500;
        T=0.15;
        delt=0.0001;

        FUN_Finite_Volume_Solver(L1,BC1,qlin,L4,BC4,q4in,Nx,T,del
t,ch);
    case 3
```

```

L1=19;
L4=1;
BC1="SW";
BC4="SW";
q1in=[100*101325,293,0];
q4in=[0.1*101325,293,0];
Nx=500;
T=0.08;
delt=0.00001;

FUN_Finite_Volume_Solver(L1,BC1,q1in,L4,BC4,q4in,Nx,T,del
t,ch);
case 4
    L1=input('\nEnter length of the driven section');
%driven section length length
    L4=input('\nEnter length of the driver section');
%driver section length length

    q4in=input('\nEnter the driver section initial
conditions[p1,T1,u1]');
    q1in=input('\nEnter the driven section initial
conditions[p4,T4,u4]');

    fprintf('\nEnter boundary conditoin type: \nSW->
solid wall condition \nFF-> far-field condition')
    BC1=input('\ndriven section boundary type','s');
%driven section boundary type
    BC4=input('\ndrive4 section boundary type','s');
%driven section boundary type

    Nx=input('\nEnter number of grid points');
%number of cells
    T=input('\nEnter final simulation time');
%total time
    delt=input('\nEnter the time step size');
%time step size

FUN_Finite_Volume_Solver(L1,BC1,q1in,L4,BC4,q4in,Nx,T,del
t,ch);
otherwise
    fprintf('\n wrong choice')
end

```

## Appendix 2: Function FUN\_Finite\_Volume\_Solver

```
function
[] = FUN_Finite_Volume_Solver(L1, BC1, q1in, L4, BC4, q4in, Nx, T,
delt, ch)

% Author      : Abhyudaya Singh
% UIN         : 655691216
% NetID       : singh124@illinois.edu

% This code solves the 1D Euler's equation using Finite
% Volume method using Godunov's method for flux
% reconstruction

%% initialization

gama=1.4;
R=287;

L=L1+L4;
delx=L/(Nx);

N_dia=round(L4/delx);           % node at diaphram

q0=zeros(10000,Nx);
q1=zeros(10000,Nx);
q2=zeros(10000,Nx);

FL=zeros(3,Nx);
FR=zeros(3,Nx);

%% initializing flow field
rho1in=q1in(1)/(R*q1in(2));
rho4in=q4in(1)/(R*q4in(2));
q0(1,1:N_dia) = ones(1,N_dia)*rho1in;
q1(1,1:N_dia) = q0(1,1:N_dia)*q1in(3);
q2(1,1:N_dia) = ones(1,N_dia)*(q1in(1)/(gama-1) +
0.5*rho1in*q1in(3)^2);

q0(1,N_dia+1:end) = ones(1,Nx-N_dia)*rho4in;
q1(1,N_dia+1:end) = q0(1,N_dia+1:end)*q4in(3);
q2(1,N_dia+1:end) = ones(1,Nx-N_dia)*(q4in(1)/(gama-1) +
0.5*rho4in*q4in(3)^2);

%% solution
fprintf('\nCalculating the solution!...')
```

```

fprintf('\n\titer \t Flow time')

t=zeros(10000,1);
ct=0;
while t(ct+1)<=T
    ct=ct+1;
    for n=1:Nx

        if n==1
            if BC1 == "SW"
                if q1(n)>0
                    qL=FUN_Godunov_flux_recon([q0(ct,n),
0, q2(ct,n)], [q0(ct,n),q1(ct,n),q2(ct,n)],gama,0);
                else

qL=FUN_Godunov_flux_recon([2*q0(ct,n)-q0(ct,n+1), 0,
2*q2(ct,n)-
q2(ct,n+1)], [q0(ct,n),q1(ct,n),q2(ct,n)],gama,0);
                end
            elseif BC1 == "FF"
                if q1(n)>0
                    qL=FUN_Godunov_flux_recon([q0(ct,n),
q1(ct,n), q2(ct,n)], [q0(ct,n),q1(ct,n),q2(ct,n)],gama,0);
                else

qL=FUN_Godunov_flux_recon([2*q0(ct,n)-q0(ct,n+1),
2*q1(ct,n)-q1(ct,n+1), 2*q2(ct,n)-
q2(ct,n+1)], [q0(ct,n),q1(ct,n),q2(ct,n)],gama,0);
                end
            end
        end

qR=FUN_Godunov_flux_recon([q0(ct,n),q1(ct,n),q2(ct,n)], [q
0(ct,n+1),q1(ct,n+1),q2(ct,n+1)],gama,0);
        PL=(gama-1)*(qL(3)-(0.5*qL(2)^2 /qL(1)));
        FL(1:3,n)=[qL(2), qL(2)^2 /qL(1) + PL,
(qL(3)+PL)*qL(2)/qL(1)]';
        PR=(gama-1)*(qR(3)-(0.5*qR(2)^2 /qR(1)));
        FR(1:3,n)=[qR(2), qR(2)^2 /qR(1) + PR,
(qR(3)+PR)*qR(2)/qR(1)]';

        elseif n==Nx

            if BC4 == "SW"
                if q1(n)>0

qR=FUN_Godunov_flux_recon([q0(ct,n),q1(ct,n),q2(ct,n)], [2
*q0(ct,n)-q0(ct,n-1), 0, 2*q2(ct,n)-q2(ct,n-1)],gama,0);

```

```

    else

qR=FUN_Godunov_flux_recon([q0(ct,n),q1(ct,n),q2(ct,n)], [q
0(ct,n),0,q2(ct,n)],gama,0);
    end
elseif BC4 == "FF"
    if q1(n)>0

qR=FUN_Godunov_flux_recon([q0(ct,n),q1(ct,n),q2(ct,n)], [2
*q0(ct,n)-q0(ct,n-1), 2*q1(ct,n)-q1(ct,n-1), 2*q2(ct,n)-
q2(ct,n-1)],gama,0);
    else

qR=FUN_Godunov_flux_recon([q0(ct,n),q1(ct,n),q2(ct,n)], [q
0(ct,n),q1(ct,n),q2(ct,n)],gama,0);
    end
end
qL=FUN_Godunov_flux_recon([q0(ct,n-
1),q1(ct,n-1),q2(ct,n-
1)], [q0(ct,n),q1(ct,n),q2(ct,n)],gama,0);
    PL=(gama-1)*(qL(3)-(0.5*qL(2)^2 /qL(1)));
    FL(1:3,n)=[qL(2), qL(2)^2 /qL(1) + PL,
(qL(3)+PL)*qL(2)/qL(1)]';
    PR=(gama-1)*(qR(3)-(0.5*qR(2)^2 /qR(1)));
    FR(1:3,n)=[qR(2), qR(2)^2 /qR(1) + PR,
(qR(3)+PR)*qR(2)/qR(1)]';

else

qL=FUN_Godunov_flux_recon([q0(ct,n-
1),q1(ct,n-1),q2(ct,n-
1)], [q0(ct,n),q1(ct,n),q2(ct,n)],gama,0);

qR=FUN_Godunov_flux_recon([q0(ct,n),q1(ct,n),q2(ct,n)], [q
0(ct,n+1),q1(ct,n+1),q2(ct,n+1)],gama,0);
    PL=(gama-1)*(qL(3)-(0.5*qL(2)^2 /qL(1)));
    FL(1:3,n)=[qL(2), qL(2)^2 /qL(1) + PL,
(qL(3)+PL)*qL(2)/qL(1)]';
    PR=(gama-1)*(qR(3)-(0.5*qR(2)^2 /qR(1)));
    FR(1:3,n)=[qR(2), qR(2)^2 /qR(1) + PR,
(qR(3)+PR)*qR(2)/qR(1)]';
    end
end

q0(ct+1,:)= q0(ct,:)-(delt/delx)*(FR(1,:)-
FL(1,:));

```

```

        q1(ct+1,:) = q1(ct,:)- (delt/delx)*(FR(2,:)-
FL(2,:));
        q2(ct+1,:) = q2(ct,:)- (delt/delx)*(FR(3,:)-
FL(3,:));

    if ct>1
        fprintf(repmat('\b',1,lineLength))
    end
    lineLength = fprintf('\n\t%d \t %f',ct,t(ct));

    t(ct+1)=t(ct)+delt;

end

fprintf('\nCalculation complete!')

%% plotting
if ch == 1 || ch == 2
    if ch==1
        toro= table2array(readtable('Test1_Toro1.txt'));
    else
        toro= table2array(readtable('Test2_Toro1.txt'));
    end
    xana=toro(:,1);
    rhoana=toro(:,4);
    uana=toro(:,5);
    Pana=toro(:,6);

    figure(1)
    plot(-L4+delx:delx:L1,q0(ct,:))
    grid on
    hold on
    plot(xana,rhoana)
    xlabel('x')
    ylabel('rho')
    legend('Numerical solution','Toro')

    figure(2)
    plot(-L4+delx:delx:L1,q1(ct,:)./q0(ct,:))
    grid on
    hold on
    plot(xana,uana)
    xlabel('x')
    ylabel('u')
    legend('Numerical solution','Toro')

    figure(3)

```

```

    plot(-L4+delx:delx:L1, (gama-1) * (q2(ct,:))-  

(0.5*(q1(ct,:).^2)./q0(ct,:))))  

    grid on  

    hold on  

    plot(xana,Pana)  

    xlabel('x')  

    ylabel('P')  

    legend('Numerical solution','Toro')
end

if ch == 3 || ch == 4
    figure(4)
    contourf(-L4+delx:delx:L1,t(1:10:ct), (gama-
1)*(q2(1:10:ct,:))-  

(0.5*(q1(1:10:ct,:).^2)./q0(1:10:ct,:))),100,'edgecolor',
'none')
    a=colorbar;
    ylabel(a,'P (Pa)', 'Rotation',90);
    xlabel('x(m)')
    ylabel('t(sec)')

    figure(5)
    contourf(-
L4+delx:delx:L1,t(1:10:ct),q0(1:10:ct,:),100,'edgecolor',
'none')
    a=colorbar;
    ylabel(a,'rho (kg/m^{3})', 'Rotation',90);
    xlabel('x(m)')
    ylabel('t(sec)')

    figure(6)
    contourf(-
L4+delx:delx:L1,t(1:10:ct),q1(1:10:ct,:)./q0(1:10:ct,:),1
00,'edgecolor','none')
    a=colorbar;
    ylabel(a,'U (m/s)', 'Rotation',90);
    xlabel('x(m)')
    ylabel('t(sec)')

    figure(7)
    contourf(-L4+delx:delx:L1,t(1:10:ct),((gama-
1)*(q2(1:10:ct,:))-  

(0.5*(q1(1:10:ct,:).^2)./q0(1:10:ct,:)))./(R*q0(1:10:ct,
:))),100,'edgecolor','none')
    a=colorbar;
    ylabel(a,'T (K)', 'Rotation',90);
    xlabel('x(m)')

```

```
    ylabel('t(sec)')  
end  
end
```

### Appendix 3: Function FUN\_Godunov\_flux\_recon

```

function [Q] = FUN_Godunov_flux_recon(q1,q2,gama,xt)

% Author      : Abhyudaya Singh
% UIN        : 655691216
% NetID      : singh124@illinois.edu

% This function constructs the fluxes on the cell faces
% by using the Riemann solver

%converting vectors from [rho, rhou, rhoE] to [rho, u, P]
qL=[q1(1), q1(2)/q1(1), (gama-1)*(q1(3)-(0.5*q1(2)^2
/q1(1)))];
qR=[q2(1), q2(2)/q2(1), (gama-1)*(q2(3)-(0.5*q2(2)^2
/q2(1)))];

cL=sqrt(gama*qL(3)/qL(1));    % sound speed on left
cR=sqrt(gama*qR(3)/qR(1));    % sound speed on right

%% solving for P*
[pstar,ustar,rhoLstar,cLstar,rhoRstar,cRstar] =
FUN_Root_Bisection(qL,qR,gama);

%% Calculating the fluxes

sL = qL(2)-cL*sqrt(((gama+1)*pstar)/(2*gama*qL(3)) +
((gama-1)/(2*gama)));
sR = qR(2)+cR*sqrt(((gama+1)*pstar)/(2*gama*qR(3)) +
((gama-1)/(2*gama)));

if xt<ustar
    if qL(3)>=pstar
        if xt<=qL(2)-cL
            q=qL;
        elseif xt<=ustar-cLstar
            rhofL = qL(1)*((2/(gama+1)) + ((gama-
1)/(gama+1))*((qL(2)-xt)/cL))^(2/(gama-1));
            ufl = ((gama-1)/(gama+1)*(qL(2) +
2*((xt+cL)/(gama-1))));
            PfL = qL(3)*((2/(gama+1)) + ((gama-
1)/(gama+1))*((qL(2)-xt)/cL))^(2*gama/(gama-1));
            q=[rhofL,ufl,PfL];
        elseif xt>ustar-cLstar && xt<=ustar
            q=[rhoLstar,ustar,pstar];
        end
    end
end

```

```

        end
    else
        if xt<=sL
            q=qL;
        else%if xt>sL && xt<=ustar
            q=[rhoLstar,ustar,pstar];
        end
    end

else
    if qR(3)>=pstar
        if xt<=ustar+cRstar
            q=[rhoRstar,ustar,pstar];
        elseif xt<=qR(2)+cR
            rhoFR = qR(1)*((2/(gama+1)) + ((gama-
1)/(gama+1))*((qR(2)-xt)/cR))^(2/(gama-1));
            uFR = ((gama-1)/(gama+1)*(qR(2) + 2*((xt-
cR)/(gama-1))) ;
            PfR = qR(3)*((2/(gama+1)) + ((gama-
1)/(gama+1))*((qR(2)-xt)/cR))^(2*gama/(gama-1));
            q=[rhoFR,uFR,PfR];
        else
            q=qR;
        end
    else
        if xt<=sR
            q=[rhoRstar,ustar,pstar];
        else%if xt>sR
            q=qR;
        end
    end
end
Q=[q(1), q(1)*q(2), q(3)/(gama-1) + 0.5*q(1)*q(2)^2];
%[rho,rhou,rhoE]
end

```

## Appendix 4: Function FUN\_Root\_Bisection

```
function [pstar,ustar,rhoLstar,cLstar,rhoRstar,cRstar] =
FUN_Root_Bisection(qL,qR,gama)

% Author      : Abhyudaya Singh
% UIN         : 655691216
% NetID       : singh124@illinois.edu

% this function finds the root using the Bisection method
% the qk passed in this function should have qk=[rhok,
uk, Pk];

itmax=100;
a = 0.9 * (qL(3) + qR(3))/2;
b = 1.1 * (qL(3) + qR(3))/2;
it = 0;

% Make sure f(a) is negative
while qR(2) - qL(2) + FUN_fk(a,qL,gama) +
FUN_fk(a,qR,gama) > 0 && it < itmax
    a = a/2;
    it = it + 1;
end

it = 0;

% Make sure f(b) is positive
while qR(2) - qL(2) + FUN_fk(b,qL,gama) +
FUN_fk(b,qR,gama) < 0 && it < itmax
    b = b*2;
    it = it + 1;
end

% Bisection algorithm

it = 0;
e = 1e-6;
pstar = (a + b)/2;
fpstar = qR(2) - qL(2) + FUN_fk(pstar,qL,gama) +
FUN_fk(pstar,qR,gama);

while abs(fpstar) > e && (b - a)/2 > e && it < itmax
    pstar = (a + b)/2;
    fpstar = qR(2) - qL(2) + FUN_fk(pstar,qL,gama) +
FUN_fk(pstar,qR,gama);
```

```

if fpstar > 0
    b = pstar;
else
    a = pstar;
end
it = it + 1;
end

ustar = (qR(2) + qL(2) + FUN_fk(pstar,qR,gama) -
FUN_fk(pstar,qL,gama))/2;
rhoLstar = qL(1)*(pstar/qL(3))^(1/gama);
cLstar = sqrt(gama*pstar/rhoLstar);
rhoRstar = qR(1)*(pstar/qR(3))^(1/gama);
cRstar = sqrt(gama*pstar/rhoRstar);
end

```

## Appendix 5: Function FUN\_fk

```
function fK = FUN_fk(pstar,qK,gamma)

% Author      : Abhyudaya Singh
% UIN         : 655691216
% NetID       : singh124@illinois.edu

    % First extract density, velocity, and pressure from
q
    rhoK = qK(1);
    pK = qK(3);

    if pstar <= pK
        % Rarefaction wave
        % Calculate speed of sound
        cK = sqrt(gamma * pK / rhoK);
        % Return fK
        fK = 2 * cK / (gamma - 1) * ((pstar / pK)^( (gamma
- 1)/(2 * gamma)) - 1);
    else
        % Shock wave
        % Calculate A and B
        AK = 2 / (rhoK * (gamma + 1));
        BK = (gamma - 1) / (gamma + 1) * pK;
        % Return fK
        fK = (pstar - pK) * sqrt(AK / (BK + pstar));
    end
end
```