ME 673 - PROGRAMMING ASSIGNMENT # 1

Gowtham Kuntumalla, 140100091 Sept 2017

Gas Dynamics Assignment

This code is written in python to numerically solve the unsteady flow situation which occurs in a shock tube. Expansion waves and compression shock waves are generated after the diaphragm in the tube is punctured.

Python Code

Initial conditions are mentioned in the beginning lines. The first part of the code is for the compression shock reflection from a rigid wall. The second part of code is for reflection of expansion wave from a rigid wall with 'N' number of wavefronts.

To the right of x=0 is compression wave and to the left is expansion wave. In the end all the plots are combined in a single x-t diagram for a given N.

Listing 1: Python code – Shock tube reflections – solving analytically.

```
#!/usr/bin/python
# Python code for Numerical assignment 1, ME 678
# Written by Gowtham Kuntumalla, 140100091
# Finite amplitude reflection in a Shock tube.
# Both compression shock wave and expansion wave are considered.

import matplotlib.pyplot as plt
import numpy as np
import math
from scipy.optimize import fsolve

print("hey Code is running !")
y=1.4
R=287
Ti=300
P1=101000
rho1=P1/(R*Ti)
# Ratio 1 (Shock Strength): r1=P2/P1=P3/P1;
# Ratio 2 (Diapraghm Pressure Ratio): r2= P4/P1
r2=5
Lr=9 # driven section length
Ll=3 # driver section length
```

```python
N=5 # number of expansion wave fronts


# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% #
## Compression shock wave reflection, r-running

print("Compression shock wave reflection at right end")

    #  %% BEFORE REFLECTION %% #
T1=Ti
func1 = lambda r1: r1*(1-(y-1)*(r1-1)/math.sqrt(2*y*(2*y+(y+1)*(r1-1))))**(-2*y/(y-1))-r2
r1_sol = fsolve(func1,0.9*r2) # it is some initial guess
r1 = r1_sol
a1 = math.sqrt(y*R*Ti) #sound speed
u1c = 0
u2c = math.sqrt(R*Ti/y)*(r1-1)*(2*y/(y+1)/(r1+(y-1)/(y+1)))
up = u2c # piston speed i.e contact surface
#Cs=u2c/(1-(r1+(y+1)/(y-1))/(1+r1*(y+1)/(y-1)))#shock speed
Cs = a1*math.sqrt((y+1)/(2*y)*(r1-1)+1)

rho1_by_rho2 = (Cs-u2c)/Cs
T2_by_T1 = r1*(rho1_by_rho2)
P3_by_P4 = r1/r2
T2 = T2_by_T1*Ti
rho2 = rho1/rho1_by_rho2
a2 = math.sqrt(y*R*T2)

    #  %% AFTER REFLECTION %%#

u5c = 0 # rigid wall
Ms = Cs/a1#incident mach
func2 = lambda Mr: Mr/(Mr**2-1)-Ms/(Ms**2-1)*math.sqrt(1+(2*(y-1)/(y+1)**2)*(Ms**2-1)*(y+1/Ms**2))
Mr_sol = fsolve(func2,1.1*Ms) #Note it is actually less than Ms. Math trick ! quadratic graph upward facing curve
Cr = Mr_sol*a2-up #print("%f %f" %(Cr,up))

    # %% X-T DIAGRAM %%#

xcom = np.arange(0,Lr+1)
tcom_befr = 1/Cs*xcom
tcom_aftr = 1/Cs*Lr+1/Cr*Lr-1/Cr*xcom
plt.plot(xcom,tcom_befr,linewidth=2.0)
plt.plot(xcom,tcom_aftr,linewidth=2.0)
plt.xlabel("X(m)")
plt.ylabel("Time (sec)")
```

```python
#plt.show()

# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% #

## Centred Expansion wave reflection l-running

print("Expansion wave reflection l-running with %d wavefronts" %(N))
print("w1,w2,w3 ... are expansion wavefronts")

    #  %% BEFORE REFLECTION %% #
P4 = P1*r2
a4 = a1
T4 = Ti
u3e = u2c

a3_by_inf = 1-(y-1)/2*u3e/math.sqrt(y*R*Ti)  # Using EOS & Isentropic relations
P3_by_inf = (1-(y-1)/2*u3e/math.sqrt(y*R*Ti))**(2*y/(y-1))
T3_by_inf = (1-(y-1)/2*u3e/math.sqrt(y*R*Ti))**(2)
rho3_by_inf = (1-(y-1)/2*u3e/math.sqrt(y*R*Ti))**(2/(y-1))
a3 = a4*a3_by_inf
T3 = Ti*T3_by_inf
P3 = P4*P3_by_inf
rho3 = P4/(R*Ti)*rho3_by_inf

P_ewave = [None] * (N+1)
rho_ewave = [None] * (N+1)
T_ewave = [None] * (N+1)
speed_ewave = [None] * (N+1)

#Slope interpolation
speed_ewave[1] = -a4
speed_ewave[N] = up-a3
rand_var1 = (1/speed_ewave[N] -1/speed_ewave[1])/(N-1)
rand_var2 = 1/speed_ewave[1]-rand_var1

for i in range(2,N):
    speed_ewave[i] = 1/(rand_var1 * i + rand_var2)

    #  %% AFTER REFLECTION %% # (NON SIMPLE REGION)

# N(N+1)/2 intersection points are present
# Concept of Reimann invariants is used
u_ew0 = [None] * (N+1)
u_ew = [None] * (N+1)
a_ew = [None] * (N+1)
u_ew0[1] = 0
```

```python
u_ew0[N] = up
u_ew[1] = 0
a_ew[1] = a4

## FIRST N POINTS ##

# Flow velocity interpolation at origin
for i in range(2,N):
    u_ew0[i] = i*up/(N-1)-up/(N-1)
# Reimann invariants
for i in range(2,N+1):
    coeff = np.array([[1,2/(y-1)], [1,-2/(y-1)]]) # ax=b
    ordinate = np.array([u_ew[i-1]+2*a_ew[i-1]/(y-1),u_ew0[i]-2*(u_ew0[i]-speed_ewave[i])/(y-1)])
    x = np.linalg.solve(coeff, ordinate)
    u_ew[i] = x[0]
    a_ew[i] = x[1]

texp_point = [None] * (N+1) # (x,t) is the coordinate pair of an intersection point
xexp_point = [None] * (N+1)
xexp_point[1] = -Ll
texp_point[1] = xexp_point[1]/speed_ewave[1]

for i in range(2,N+1):
    coeff = np.array([[1,-speed_ewave[i]],[0.5*(1/(u_ew[i-1]+a_ew[i-1])+1/(u_ew[i]+a_ew[i])),-1]])
    ordinate = np.array([0,xexp_point[i-1]*0.5*(1/(u_ew[i-1]+a_ew[i-1])+1/(u_ew[i]+a_ew[i]))-texp_point[i-1]])
    x=np.linalg.solve(coeff,ordinate)
    xexp_point[i] = x[0]
    texp_point[i] = x[1]


## REMAINING POINTS ##
# NOTE FROM HERE ON USE 0 to N-1 notation of python for easy computation
# (0 TO N-1) x (0 TO N-1) matrix for points
print("non simple region")
Points_Matrix_u = [[0 for x in range(N)] for x in range(N)] # free stream velocity
Points_Matrix_a = [[0 for x in range(N)] for x in range(N)] # sound velocity
Points_Matrix_x = [[0 for x in range(N)] for x in range(N)] # position of intersection
Points_Matrix_t = [[0 for x in range(N)] for x in range(N)] # time at that position

```

```
# copying the old first row into new matrix

for i in range(0,N):
    Points_Matrix_u[0][i] = u_ew[i+1]
    Points_Matrix_a[0][i] = a_ew[i+1]
    Points_Matrix_x[0][i] = xexp_point[i+1]
    Points_Matrix_t[0][i] = texp_point[i+1]

# u,a,x,t are the four important properties for determining each point
for i in range(1,N):
    for j in range(i,N):
        if j == i:
            Points_Matrix_u[i][i] = 0
            Points_Matrix_x[i][i] = -Ll
            Points_Matrix_a[i][i] = Points_Matrix_a[i-1][i] - Points_Matrix_u[i-1][i] * (y-1)/2
            Points_Matrix_t[i][i] = Points_Matrix_t[i-1][i] + (Points_Matrix_x[i][i] - Points_Matrix_x[i-1][i])*(1/(Points_Matrix_u[i-1][i]-Points_Matrix_a[i-1][i]))#+1/(Points_Matrix_u[i][i]-Points_Matrix_a[i][i]))*1/2

        else:
            # Get u, a
            coeff = np.array([[1,2/(y-1)],[1,-2/(y-1)]])
            ordinate = np.array([Points_Matrix_u[i][j-1]+2/(y-1)*Points_Matrix_a[i][j-1],Points_Matrix_u[i-1][j]-2/(y-1)*Points_Matrix_a[i-1][j]])
            x = np.linalg.solve(coeff, ordinate)
            Points_Matrix_u[i][j] = x[0]
            Points_Matrix_a[i][j] = x[1]

            # Get x, t
            large_coefft1 = -(1/(Points_Matrix_u[i][j-1]+Points_Matrix_a[i][j-1]))#+1/(Points_Matrix_u[i][j]+Points_Matrix_a[i][j]))*1/2
            large_coefft2 = -(1/(Points_Matrix_u[i-1][j]-Points_Matrix_a[i-1][j]))#+1/(Points_Matrix_u[i][j]-Points_Matrix_a[i][j]))*1/2
            coeff1 = np.array([[large_coefft1,1],[large_coefft2,1]])
            ordinate1 = np.array([Points_Matrix_t[i][j-1]+large_coefft1*Points_Matrix_x[i][j-1],Points_Matrix_t[i-1][j]+large_coefft2*Points_Matrix_x[i-1][j]])
            x1 = np.linalg.solve(coeff1, ordinate1)
            Points_Matrix_x[i][j] = x1[0]
            Points_Matrix_t[i][j] = x1[1]


```

```
    # %% X-T DIAGRAM %% #

#   xexp = np.arange(-Ll,1)
#   for i in range(1,N+1):
#       texp_befr = 1/speed_ewave[i] * xexp
#       plt.plot(xexp,texp_befr,linewidth=2.0)


for i in range(0,N):
    plt.plot([0,Points_Matrix_x[0][i]],[0,Points_Matrix_t[0][i]])

for i in range(0,N):
    for j in range(i,N):
        if i<(N-1) and j == i :
            plt.plot([Points_Matrix_x[i][i],Points_Matrix_x[i][i+1]],[Points_Matrix_t[i][i],Points_Matrix_t[i][i+1]])
            plt.plot([Points_Matrix_x[i][i],Points_Matrix_x[i+1][i+1]],[Points_Matrix_t[i][i],Points_Matrix_t[i+1][i+1]])
        elif j == (N-1):
            if i<(N-1):
                plt.plot([Points_Matrix_x[i][N-1],Points_Matrix_x[i+1][N-1]],[Points_Matrix_t[i][N-1],Points_Matrix_t[i+1][N-1]])
            xexp_aft_ref = np.arange(Points_Matrix_x[i][N-1],1)
            texp_aft_ref = Points_Matrix_t[i][N-1] + 1/(Points_Matrix_u[i][N-1]+Points_Matrix_a[i][N-1]) * (xexp_aft_ref - Points_Matrix_x[i][N-1])
            plt.plot(xexp_aft_ref,texp_aft_ref,linewidth=2.0)
        elif i<(N-1) :
            plt.plot([Points_Matrix_x[i][j],Points_Matrix_x[i+1][j]],[Points_Matrix_t[i][j],Points_Matrix_t[i+1][j]])
            plt.plot([Points_Matrix_x[i][j],Points_Matrix_x[i][j+1]],[Points_Matrix_t[i][j],Points_Matrix_t[i][j+1]])

plt.plot(Points_Matrix_x,Points_Matrix_t,'ro')
plt.show()
print("%f %f" %(a3,a4))
#print(Points_Matrix_u,Points_Matrix_a,Points_Matrix_x,Points_Matrix_t)
print(xexp_aft_ref)
# END OF PROGRAM
```

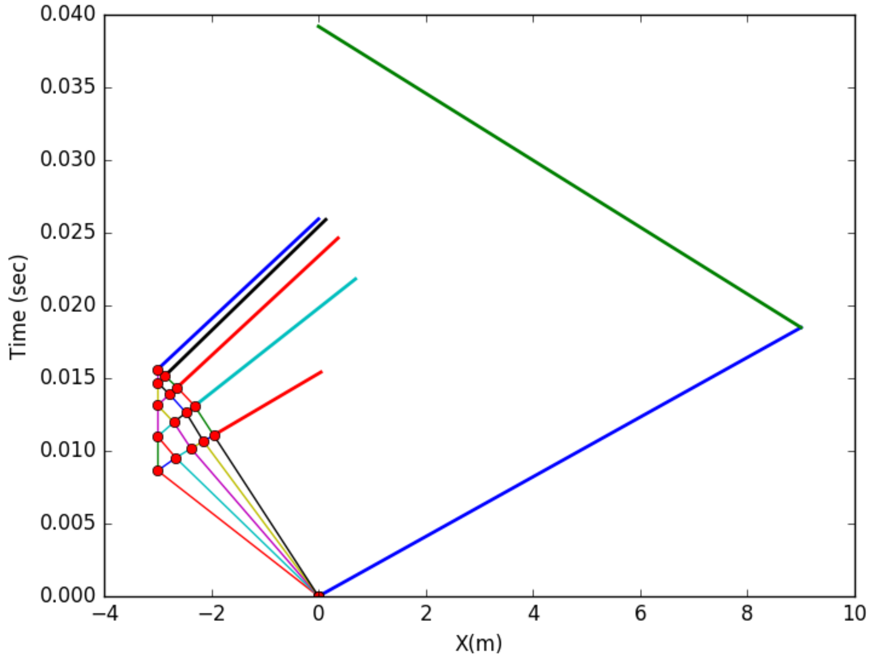
Following Listing 1... We observe the following after performing some experiments by changing number(N) of expansion wavefronts.

Figures & Conclusions

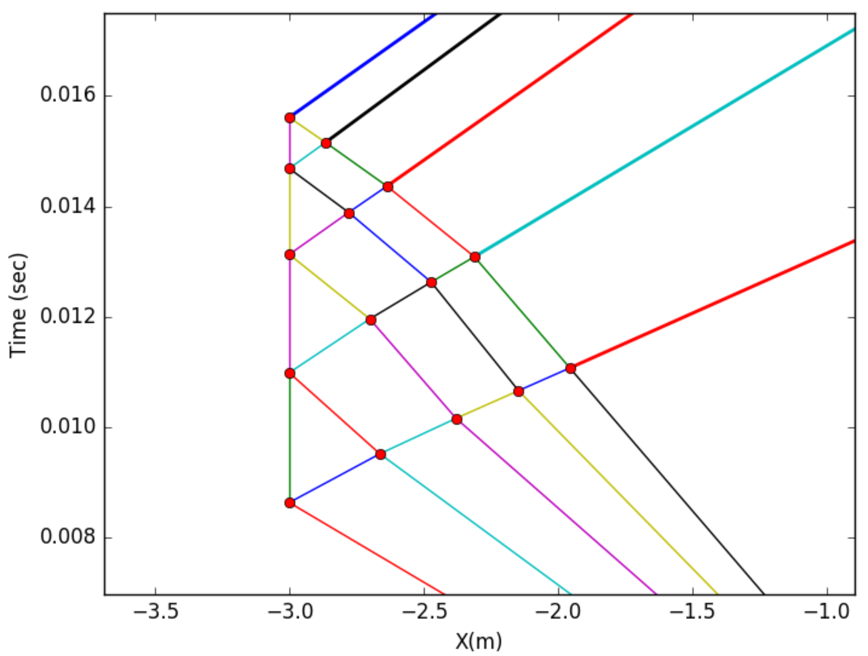
These figures are obtained directly from running the python code. We observe that as the number of expansion wave fronts increases the clarity of the "Non-Simple Region" increases. Mathematically it gives better and accurate results but large number of wavefronts is not good for simple viewing of image.

Another interesting observation is the effect of wall on the characteristics. There is steep change in slope near the wall as compared to locations far off from the wall. This is well in line with intuition. The effect of wall decays with distance from it.

Concluding this assignment, We note that for a simple problem which can be solved easily using modern softwares, it is indeed quite an effort to delve deep and solve for each wave front individually. It is still a worthwhile and fun exercise for this class!

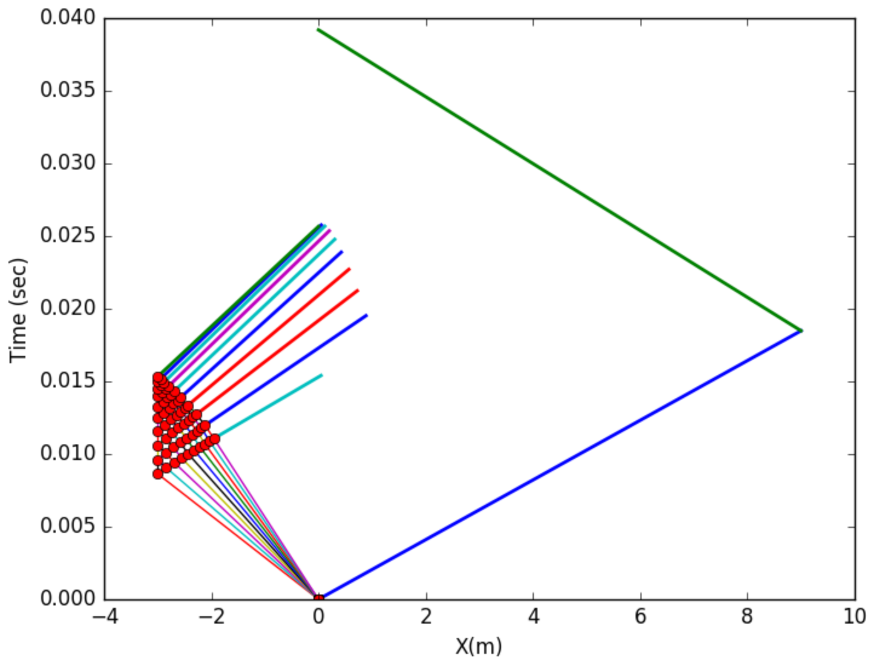


(a) Overall Picture

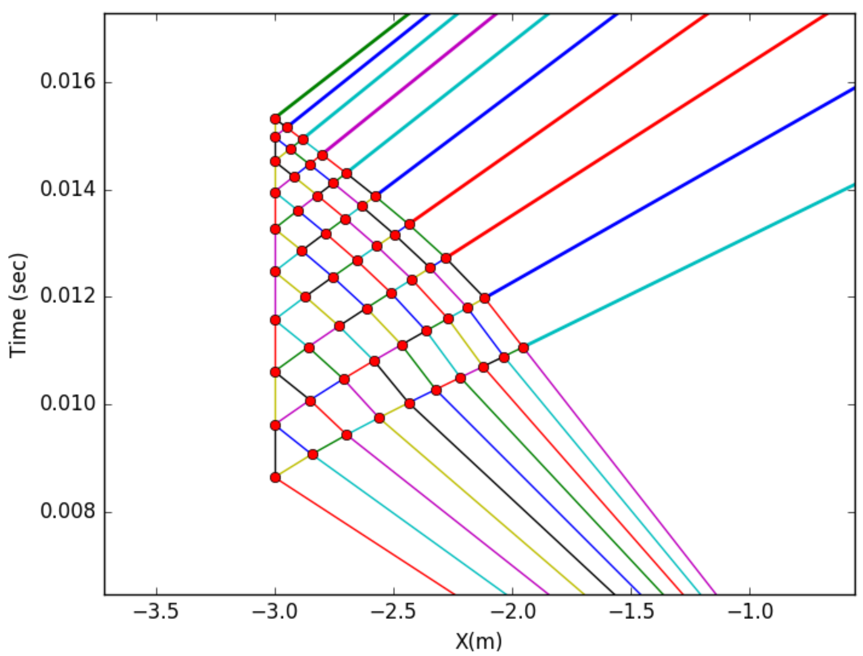


(b) Expansion wave Zoomed

Figure 1: x-t diagram for N=5 wavefronts

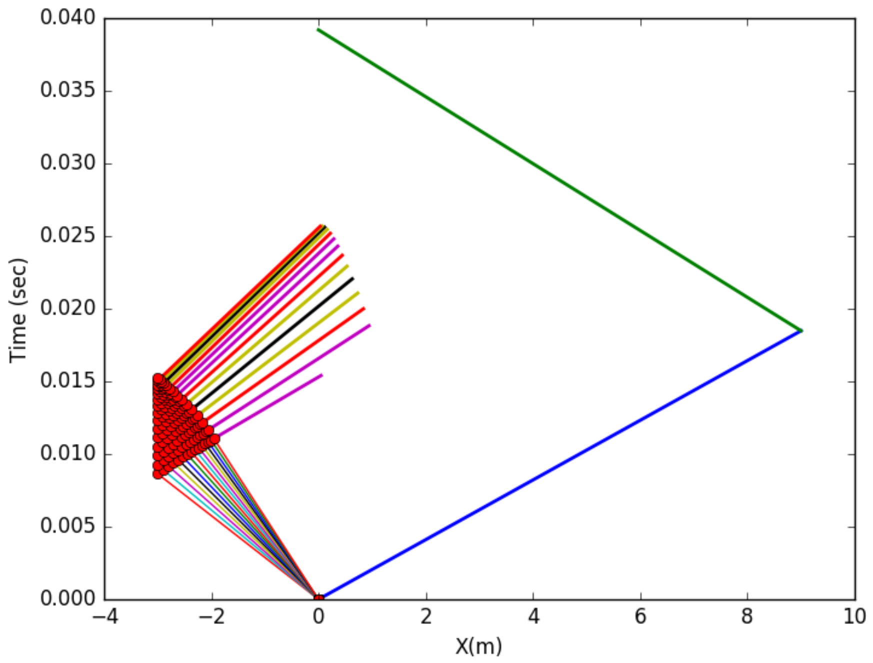


(a) Overall Picture

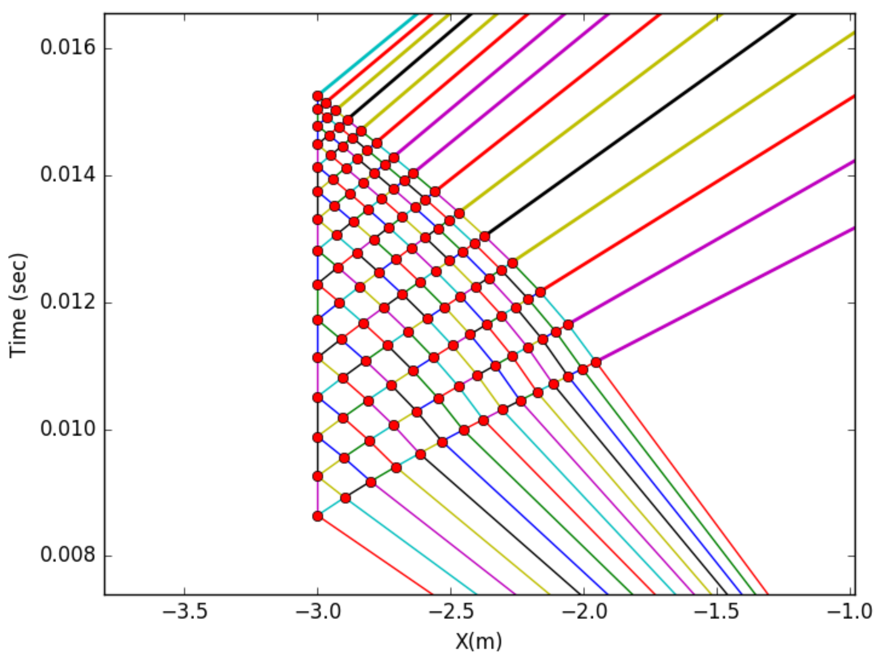


(b) Expansion wave Zoomed

Figure 2: x-t diagram for N=10 wavefronts



(a) Overall Picture



(b) Expansion wave Zoomed

Figure 3: x-t diagram for N=15 wavefronts