

ENGR 6251 Finite Difference Methods in Computational Fluid Dynamics

Assignment # 4

Done by

Yusri Al-Sanaani

ID:40070232

yosritaha@yahoo.com

Submitted to

Dr. Brian Vermeire

November 25, 2019

**Question 1**

A 2D solver to solve lid-driven cavity flow problem has been implemented here. The discretization is performed by using a second-order central scheme in space in addition to the fourth-order RK method for time integration. The solution is advanced to steady state using different grid refinements starting from nx=ny=33 to 260. The results of solution at nx=ny=33 are shown in figure 1. By comparing the line plots of velocity components, this solver gives similar behavior to those results in the reference with significant difference in the magnitude. Also, the contours are not smooth. By performing further refinement to the grid size, the solution slowly closes to those in reference as shown in figure 2 and figure 3. At nx=ny=260, which is the double of 129, the line plots of velocity components are very similar and close to those results reported in the reference. Also, the contours are finer and more organized as shown in figure 4.

|  |  |
| --- | --- |
|  |  |
| (a) | (b) |
|  |  |
| (c) | (d) |
| Figure 1: Velocity & pressure contours and line plot of velocity components at nx=ny =33. | |

|  |  |
| --- | --- |
|  |  |
| (a) | (b) |
|  |  |
| (c) | (d) |
| Figure 2: Velocity & pressure contours and line plot of velocity components at nx=ny =65. | |

|  |  |
| --- | --- |
|  |  |
| (a) | (b) |
|  |  |
| (c) | (d) |
| Figure 3: Velocity & pressure contours and line plot of velocity components at nx=ny =129. | |

|  |  |
| --- | --- |
|  |  |
| (a) | (b) |
|  |  |
| (c) | (d) |
| Figure 4: Velocity & pressure contours and line plot of velocity components at nx=ny =260. | |

***Main code:***

========================================================================

ENGR 6251 Finite Difference Methods in Computational Fluid Dynamics  
Assignment # 4, Question# 1  
Done by Yusri Al-Sanaani, ID:40070232  
MATLAB program to solve Lid driven cavity.

## %% Initializations

clc; clear all; close all  
L=1; nx=33; ny=nx; dx=L/(nx-1); dy=L/(ny-1); x=0:dx:L;  
y=0:dy:L; tmax=20; t=0; dt=0.001; Re=100; Ma=0.1;  
Pr=0.71; rhoi=1; gamma=1.4; R=1; U=1;  
cp=(gamma\*R)/(gamma-1);  
Tw=(U/Ma)^2/(gamma\*R);  
pi=rhoi\*R\*Tw;  
mu=(rhoi\*L\*U)/Re;  
k=(gamma\*mu)/Pr;  
sigma=-dt/(2\*dx\*dy);

## %% Obtain initial W=[rho, rho\*u, rho\*v, rho\*E]

u=zeros(nx); v=zeros(ny);  
p=pi\*ones(nx); rho=rhoi\*ones(nx);  
rhoE=(p/(gamma-1))+0.5\*rho.\*(u.^2+v.^2);  
W(:,:,1)=rho; W(:,:,2)=rho.\*u;  
W(:,:,3)=rho.\*v; W(:,:,4)=rhoE;

## %% Compute primitive variables Boundary conditions

[T,p,u,v]=variables(W,gamma,R); [T,p,u,v]=BCs(T,p,u,v,nx,ny,Tw);

## %% Compute the initial flow, W=[rho, rho\*u, rho\*v, rhoE]

W=LDC(gamma,u,v,p,T,R);

## %% Second-order central FDM scheme in space and RK4 method in time

W\_i=W;  
while t<=tmax  
 W1=W\_i;  
 [F,G]=Tot\_Flux(W1,gamma,R,nx,ny,dx,dy,mu,k,Tw);  
 R1=RNS(F,G,nx,ny,sigma);  
 W2=W\_i+(dt/2)\*R1;  
 [F,G]=Tot\_Flux(W2,gamma,R,nx,ny,dx,dy,mu,k,Tw);  
 R2=RNS(F,G,nx,ny,sigma);  
 W3=W\_i+(dt/2)\*R2;  
 [F,G]=Tot\_Flux(W3,gamma,R,nx,ny,dx,dy,mu,k,Tw);  
 R3=RNS(F,G,nx,ny,sigma);  
 W4=W\_i+dt\*R3;  
 [F,G]=Tot\_Flux(W4,gamma,R,nx,ny,dx,dy,mu,k,Tw);  
 R4=RNS(F,G,nx,ny,sigma);  
 W\_RK4=W\_i+(dt/6)\*(R1+2\*R2+2\*R3+R4);  
 [T,p,u,v]=variables(W\_RK4,gamma,R);  
 [T,p,u,v]=BCs(T,p,u,v,nx,ny,Tw);  
 WNEW=LDC(gamma,u,v,p,T,R);  
 W\_i=WNEW;  
 t=t+dt;  
end

## Result Plots

Results(W\_i,gamma,R,x,y,nx,t,Re)

***The functions used in this solver***

# Function to compute primitive variables

function [T,p,u,v]=variables(W,gamma,R)  
u=W(:,:,2)./W(:,:,1); v=W(:,:,3)./W(:,:,1);  
E=W(:,:,4)./W(:,:,1); p=(gamma-1)\*W(:,:,1).\*(E-0.5\*(u.^2 +v.^2));  
T=p./(R\*W(:,:,1));  
end

## Function to update Boundary conditions for u,v,p,T

function [T,p,u,v]=BCs(T,p,u,v,nx,ny,Tw)

## % u Boundary conditions

Down % Right

u(:,1)=0; u(nx,:)=0;  
% Left % Up  
u(1,:)=0; u(:,ny)=1;

## % v Boundary conditions

%Down % Right

v(:,1)=0; v(nx,:)=0;  
% Left % Up  
v(1,:)=0; v(:,ny)=0;

## % p Boundary conditions

% Down % Right

p(2:nx-1,1)=p(2:nx-1,2); p(nx,2:ny-1)=p(nx-1,2:ny-1);  
% Left % Up  
p(1,2:ny-1)=p(2,2:ny-1); p(2:nx-1,ny)=p(2:nx-1,ny-1);  
% DL corner % UL corner  
p(1,1)=p(2,2); p(1,ny)=p(2,ny-1);  
% DR corner % UR corner  
p(nx,1)=p(nx-1,2); p(nx,ny)=p(nx-1,ny-1);

## % T Boundary conditions

% Down % Right

T(:,1)=Tw; T(nx,:)=Tw;  
% Left % Up  
T(1,:)=Tw; T(:,ny)=Tw;

end

# Function to compute W=[rho, rho\*u, rho\*v, rhoE]

function W=LDC(gamma,u,v,p,T,R)  
rho=p./(R\*T);  
rhoE=(p/(gamma-1))+0.5\*rho.\*(u.^2 + v.^2);  
W(:,:,1)=rho; W(:,:,2)=rho.\*u; W(:,:,3)= rho.\*v; W(:,:,4)=rhoE;  
end

# Function to obtain the total flux (inviscid & Viscus flux)

function [F,G]=Tot\_Flux(W,gamma,R,nx,ny,dx,dy,mu,k,Tw)  
[T,p,u,v]=variables(W,gamma,R);  
[T,p,u,v]=BCs(T,p,u,v,nx,ny,Tw);  
[Fi,Gi]=Flux\_Inviscid(W,gamma,p);  
[Fv,Gv]=Flux\_Viscus(u,v,T,nx,ny,dx,dy,mu,k);  
F=Fi+Fv;  
G=Gi+Gv;  
end

## Function to Evaluate the inviscid flux

function [Fi,Gi]=Flux\_Inviscid(W,gamma,p)

## % Conserved variables

rho=W(:,:,1); rhou=W(:,:,2); rhov=W(:,:,3); E=W(:,:,4)./rho;

## % compute primitive variables

u=rhou./rho; v=rhov./rho;

## % Compute flux functions

Fi=zeros(size(W));  
Fi(:,:,1)=rhou; Fi(:,:,2)=rhou.\*u+p; Fi(:,:,3)=rhov.\*u; Fi(:,:,4)=u.\*(rho.\*E+p);  
Gi=zeros(size(W));  
Gi(:,:,1)=rhov; Gi(:,:,2)=rhou.\*v; Gi(:,:,3)=rhov.\*v+p; Gi(:,:,4)=v.\*(rho.\*E+p);

end

## Function to obtain the viscus flux

function [Fv,Gv]=Flux\_Viscus(u,v,T,nx,ny,dx,dy,mu,k)

## % Find du/dx,du/dy,dv/dx,dv/dy

dudx=zeros(nx); dudy=zeros(nx);  
dvdy=zeros(nx); dvdx=zeros(nx);  
for i=2:nx-1  
 for j=2:ny-1  
 dudx(i,j)=(u(i+1,j)-u(i-1,j))/(2\*dx);  
 dudy(i,j)=(u(i,j+1)-u(i,j-1))/(2\*dy);  
 dvdx(i,j)=(v(i+1,j)-v(i-1,j))/(2\*dx);  
 dvdy(i,j)=(v(i,j+1)-v(i,j-1))/(2\*dy);  
 end  
end

for i=2:nx-1  
 dudx(i,1)=(u(i+1,1)-u(i-1,1))/(2\*dx); % Down  
 dudx(i,ny)=(u(i+1,ny)-u(i-1,ny))/(2\*dx); % Up  
 dudy(1,i)=(u(1,i+1)-u(1,i-1))/(2\*dy); % Left  
 dudy(nx,i)=(u(nx,i+1)-u(nx,i-1))/(2\*dy); % Top  
  
 dvdx(i,1)=(v(i+1,1)-v(i-1,1))/(2\*dx); % Down  
 dvdx(i,ny)=(v(i+1,ny)-v(i-1,ny))/(2\*dx); % Up  
 dvdy(1,i)=(v(1,i+1)-v(1,i-1))/(2\*dy); % Left  
 dvdy(nx,i)=(v(nx,i+1)-v(nx,i-1))/(2\*dy); % Top  
end  
% Three points one-sided stencil finite difference schemes at walls  
for i=1:nx  
 dudx(1,i)=(-3\*u(1,i)+4\*u(2,i)-u(3,i))/(2\*dx); % Left  
 dudx(nx,i)=(u(nx-2,i)-4\*u(nx-1,i)+3\*u(nx,i))/(2\*dx); % Right  
 dudy(i,1)=(-3\*u(i,1)+4\*u(i,2)-u(i,3))/(2\*dx); % down  
 dudy(i,ny)=(u(i,ny-2)-4\*u(i,ny-1)+3\*u(i,ny))/(2\*dx); % Top  
  
 dvdx(1,i)=(-3\*v(1,i)+4\*v(2,i)-v(3,i))/(2\*dx); % Left  
 dvdx(nx,i)=(v(nx-2,i)-4\*v(nx-1,i)+3\*v(nx,i))/(2\*dx); % Right  
 dvdy(i,1)=(-3\*v(i,1)+4\*v(i,2)-v(i,3))/(2\*dx); % down  
 dvdy(i,ny)=(v(i,ny-2)-4\*v(i,ny-1)+3\*v(i,ny))/(2\*dx); % Top  
end

## %% find stresses

ss\_xx=(2\*mu/3)\*(2\*dudx-dvdy);  
ss\_yy=(2\*mu/3)\*(2\*dvdy-dudx);  
ss\_yx=mu\*(dudy+dvdx);  
ss\_xy=ss\_yx;

## %% Find Qx=-k\*dT/dx and Qy=-k\*dT/dy

Qx=zeros(nx);  
Qy=zeros(nx);  
for i=2:nx-1  
 for j=2:ny-1  
 Qx(i,j)=(-k/(2\*dx))\*(T(i+1,j)-T(i-1,j));  
 Qy(i,j)=(-k/(2\*dy))\*(T(i,j+1)-T(i,j-1));  
 end  
end

for i=2:nx-1  
 Qx(i,1)=(-k/(2\*dx))\*(T(i+1,1)-T(i-1,1)); % Down  
 Qx(i,ny)=(-k/(2\*dx))\*(T(i+1,ny)-T(i-1,ny)); % Up  
 Qy(1,i)=(-k/(2\*dx))\*(T(1,i+1)-T(1,i-1)); % L  
 Qy(nx,i)=(-k/(2\*dx))\*(T(nx,i+1)-T(nx,i-1)); % T  
end

% Three points stencil finite difference schemes  
for i=1:nx  
 Qx(1,i)=(-k/(2\*dx))\*(-3\*T(1,i)+4\*T(2,i)-T(3,i));  
 Qx(nx,i)=(-k/(2\*dx))\*(T(nx-2,i)-4\*T(nx-1,i)+3\*T(nx,i));  
 Qy(i,1)=(-k/(2\*dy))\*(-3\*T(i,1)+4\*T(i,2)-T(i,3));  
 Qy(i,ny)=(-k/(2\*dy))\*(T(i,nx-2)-4\*T(i,nx-1)+3\*T(i,ny));  
end

## %% Calculate viscus flux

Fv=zeros(nx,ny,4);  
Fv(:,:,1)=zeros(nx,ny); Fv(:,:,2)=-ss\_xx; Fv(:,:,3)=-ss\_xy; Fv(:,:,4)=Qx-u.\*ss\_xx-v.\*ss\_yx;  
Gv=zeros(nx,ny,4);  
Gv(:,:,1)=zeros(nx,ny); Gv(:,:,2)=-ss\_xy; Gv(:,:,3)=-ss\_yy; Gv(:,:,4)=Qy-u.\*ss\_xy-v.\*ss\_yy;

end

# Function to calculate the RHS of NS equations

function RW=RNS(F,G,nx,ny,sigma)  
RF=zeros(size(F));  
RG=zeros(size(F));  
for n=1:4  
 for i=2:nx-1  
 for j=2:nx-1  
 R11=(F(i+1,j,n)-F(i-1,j,n));  
 R22=(G(i,j+1,n)-G(i,j-1,n));  
 RF(i,j,n)=R11;  
 RG(i,j,n)=R22;  
 end  
 end  
end  
 % Three points stencil finite difference schemes  
for k=1:4  
 for i=1:nx  
 RF(1,i,k)=((-3\*F(1,i,k)+4\*F(2,i,k)-F(3,i,k)));%L  
 RF(nx,i,k)=((F(nx-2,i,k)-4\*F(nx-1,i,k)+3\*F(nx,i,k)));%R  
 RG(i,1,k)=((-3\*G(i,1,k)+4\*G(i,2,k)-G(i,3,k)));%B  
 RG(i,ny,k)=((G(i,nx-2,k)-4\*G(i,nx-1,k)+3\*G(i,ny,k)));%T  
 end  
end  
for n=1:4  
 for i=2:nx-1  
 RF(i,1,n)=(F(i+1,1,n)-F(i-1,1,n));  
 RF(i,ny,n)=(F(i+1,ny,n)-F(i-1,ny,n));  
 RG(1,i,n)=(G(1,i+1,n)-G(1,i-1,n));  
 RG(nx,i,n)=(G(nx,i+1,n)-G(nx,i-1,n));  
 end  
end  
RW=sigma\*(RF+RG);  
end

## Function to plot the results

function Results(W\_i,gamma,R,x,y,nx,t,Re)

## %% Evaluate primitive variables, [rho,u,v,p,E]

[T,p,u,v]=variables(W\_i,gamma,R);  
figure(1)  
contourf(x,y,sqrt(u'.^2+v'.^2),40);colormap(jet)  
xlabel('x','fontweight','bold'); ylabel('y','fontweight','bold');  
set(gca,'FontName', 'Times New Roman','FontSize',12,'linewidth',1);  
title({['Velocity Field Contour at \itt = ' num2str(t),' and nx = ny = ',num2str(nx)]})  
  
figure(2)  
contourf(x,y,p',30); xlabel('x','fontweight','bold'); ylabel('y','fontweight','bold')  
set(gca,'FontName', 'Times New Roman','FontSize',12,'linewidth',1);colormap(jet)  
title({['Pressure Contour, p at \itt = ' num2str(t),' and nx = ny = ',num2str(nx)]})  
figure(3)  
quiver(x,y,u',v','r');figure(gcf); axis([0 1 0 1]);  
xlabel('x','fontweight','bold');ylabel('y','fontweight','bold');  
set(gca,'FontName', 'Times New Roman','FontSize',12,'linewidth',1);  
title({['Velocity Vectors at \itt = ' num2str(t),' and nx = ny = ',num2str(nx)]})

## %% get the data of the velocity components of Ghia et al.

yref=[0 0.0546875 0.0625 0.0703125 0.1015625 0.171875 0.28125 0.453125 0.5...  
 0.6171875 0.734375 0.8515625 0.953125 0.9609375 0.96875 0.9765625 1];  
uref=[0 -0.03717 -0.04192 -0.04775 -0.06434 -0.1015 -0.15662 -0.2109...  
 -0.20581 -0.13641 0.00332 0.23151 0.68717 0.73722 0.78871 0.84123 1];  
  
xref=[0 0.0625 0.0703 0.0781 0.0938 0.1563 0.2266 0.2344 0.5 0.8047 0.8594 ...  
 0.9063 0.9453 0.9531 0.9609 0.9688 1];  
vref=[0 0.09233 0.10091 0.10890 0.12317 0.16077 0.17507 0.17527 0.05454 ...  
 -0.24533 -0.22445 -0.16914 -0.10313 -0.08864 -0.07391 -0.05906 0];

## %% Line Plots for u & v velocities with reference data

figure(4)  
u=u';  
plot(y,u(:,round(nx/2)),yref,uref,'--','LineWidth',2.0);  
legend('My Result','Ghia Result');legend('boxoff')  
title({['Velocity Component in x at \itt = ' num2str(t),' and nx = ny = ',num2str(nx)]})  
xlabel('x','fontweight','bold');ylabel('u','fontweight','bold');  
set(gca,'FontName', 'Times New Roman','FontSize',12,'linewidth',1.5,'box','off','XAxisLocation','origin');  
figure(5)  
plot(x,v(:,round(nx/2)),xref,vref,'--','LineWidth',2.0);  
legend('My Result','Ghia Result');legend('boxoff')  
title({['Velocity Component in y at \itt = ' num2str(t),' and nx = ny = ',num2str(nx)]})  
xlabel('x','fontweight','bold'); ylabel('u','fontweight','bold');  
set(gca,'FontName', 'Times New Roman','FontSize',12,'linewidth',1.5,'box','off','XAxisLocation','origin');

end

**Question 2**

The conservative form of 1D Euler equations [1]:

|  |  |
| --- | --- |
|  | (1) |

where is the density, p is the pressure, u is the velocity (horizontal), E is the internal energy, H is the static enthalpy, and is the ratio of specific heats. Equation (1) can be rewritten as follows:

|  |  |
| --- | --- |
|  | (2) |

A is the convective flux Jacobian matrix which is given as:

|  |  |
| --- | --- |
|  | (3) |

A can be expressed as where is the diagonal matrix of real eigenvalues of A, L and R are the left and right matrices of eigenvectors of A; .

|  |  |
| --- | --- |
|  | (4) |

where a is the speed of sound.

With , equation (1) can be written as:

|  |  |
| --- | --- |
|  | (5) |

is the cell-centered values stored at nods and are the fluxes at left and right cell interface.

|  |  |
| --- | --- |
|  | (6) |

is the diagonal matrix of absolute eigenvalues of A and L and R. The Roe solver is an approximate Riemann solver and it uses below formulation to find the numerical fluxes at the interface:

|  |  |
| --- | --- |
|  | (7) |

The bars denote the Roe’s averages between the left and right interfaces. To get , , , and should be calculated first using Roe average based on expansion of Taylor series for F about points.

|  |  |
| --- | --- |
| *Roe’s , goes to A(W) as .*  *Therefore:* | (8) |

The Roe averaging formulas to compute approximate values for constructing and are given below

|  |  |
| --- | --- |
| The eigenvalues of the Jacobian matrix are | (9) |

After obtaining , a first-order upwind Roe’s scheme is used. The conservative form of flux for Roe’s solver can be written as:

|  |  |
| --- | --- |
|  | (10) |

The first-order upwind Roe’s scheme can be written as:

|  |  |
| --- | --- |
|  | (11) |

Figure 5 shows the density, velocity, and pressure distributions at N=2000.‎ The resolution of the expansion fan, contact discontinuity, and shock increases as the level of mesh refinement increases. Contact discontinuities (surfaces that separate zones of different density and temperature) becomes sharper as N increases. The expansion fan becomes sharper too. The shock become sharp and its distance decreases as N increases.

|  |  |
| --- | --- |
|  |  |
|  |  |
| Figure 5: The density, velocity, and pressure distributions at N=80. | |

|  |  |
| --- | --- |
|  | 3 |
|  |  |
| Figure 6: The density, velocity, and pressure distributions at N=160. | |

|  |  |
| --- | --- |
|  |  |
|  |  |
| Figure 7: The density, velocity, and pressure distributions at N=320. | |

|  |  |
| --- | --- |
|  |  |
|  |  |
| Figure 8: The density, velocity, and pressure distributions at N=640. | |

|  |  |
| --- | --- |
|  |  |
|  |  |
| Figure 9: The density, velocity, and pressure distributions at N=2000. | |

***The main code***

# ==========================================================

ENGR 6251 Finite Difference Methods in Computational Fluid Dynamics  
Assignment # 4, Question# 2  
Done by Yusri Al-Sanaani, ID:40070232  
Matlab program to solve Shock Tubes problem.

## %% Initialization

clear all; close all; clc  
L=1.5; N=80; dx=L/(N-1); x=0:dx:L;  
tmax=0.3; dt=0.0001; sigma=-dt/dx; gamma=1.4; t=0;

## %% Define the initial left & right parameters of Sod's shock tube, [rho,u,p]

rhoR(1:N/2)=0.125; uR(1:N/2)=0; pR(1:N/2)=0.1;  
rhoL(1:N/2)=1; uL(1:N/2)=0; pL(1:N/2)=1;

## %% Define the left & right variables of W=[rho, rho\*u, rho\*E] based on ui=[rho,u, p]

wR(1,:)=rhoR; wR(2,:)=rhoR.\*uR; rhoER=(pR/(gamma-1))+0.5\*wR(2,:).\*uR; wR(3,:)=rhoER;  
wL(1,:)=rhoL; wL(2,:)=rhoL.\*uL; rhoEL=(pL/(gamma-1))+0.5\*wL(2,:).\*uL; wL(3,:)=rhoEL;  
w=[wL wR];

## %% Solve the first-order upwind Roe’s scheme using Forward Euler Methods

while t<=tmax  
 t=t+dt;  
 F=RoeFlux(w,gamma,N);  
 for i=2:N-1  
 w(:,i)=w(:,i)+sigma\*(F(:,i)-F(:,i-1));  
 end  
end

## %% obtain the rho, u, p, and energy from final W

rho=w(1,:); u=w(2,:)./rho; p=(gamma-1)\*(w(3,:)-0.5\*w(2,:).\*u); E=w(3,:)./rho;  
%--------------------------------------------------------------------------  
figure  
plot(x,rho,'LineWidth',2.0);grid off  
title({['Density at \itt = ' num2str(t),' and N = ',num2str(N)]})  
xlabel('x','fontweight','bold');ylabel('\rho','fontweight','bold');  
set(gca,'FontName', 'Times New Roman','FontSize',12,'linewidth',1.5,'box','off','XAxisLocation','origin');  
%--------------------------------------------------------------------------  
figure  
plot(x,u,'LineWidth',2.0);  
title({['Velocity at \itt = ' num2str(t),' and N = ',num2str(N)]})  
xlabel('x','fontweight','bold');ylabel('u','fontweight','bold');  
set(gca,'FontName', 'Times New Roman','FontSize',12,'linewidth',1.5,'box','off','XAxisLocation','origin');  
%--------------------------------------------------------------------------  
figure  
plot(x,p,'LineWidth',2.0);  
title({['Pressure at \itt = ' num2str(t),' and N = ',num2str(N)]})  
xlabel('x','fontweight','bold');ylabel('p','fontweight','bold');  
set(gca,'FontName', 'Times New Roman','FontSize',12,'linewidth',1.5,'box','off','XAxisLocation','origin');  
%--------------------------------------------------------------------------  
figure  
plot(x,E,'LineWidth',2.0);  
title({['Energy at \itt = ' num2str(t),' and N = ',num2str(N)]})  
xlabel('x','fontweight','bold');ylabel('E','fontweight','bold');  
set(gca,'FontName', 'Times New Roman','FontSize',12,'linewidth',1.5,'box','off','XAxisLocation','origin');

***The function for Roe’s solver***

function F=RoeFlux(w,gamma,N)

rho=w(1,:); % Get the densities  
u=w(2,:)./w(1,:); % Get the velocities  
p=(gamma-1)\*(w(3,:)-0.5\*w(2,:).\*u); % Get the pressures  
E=w(3,:)./rho; % Get the specific energy  
H=E+p./rho; % Get the Enthalpy, H=E+p/rho  
% Get the flux  
F\_w(1,:)= w(2,:);  
F\_w(2,:)= w(2,:).\*u+p;  
F\_w(3,:)= (w(3,:)+p).\*u;

## %% Calculate Roe average values

for j=1:N-1  
 u\_avg=(u(:,j)\*sqrt(rho(:,j))+u(:,j+1)\*sqrt(rho(:,j+1)))/(sqrt(rho(:,j))+sqrt(rho(:,j+1))); % Roe's Velocity  
 H\_avg=(H(j)\*sqrt(rho(:,j))+H(j+1)\*sqrt(rho(:,j+1)))/(sqrt(rho(:,j))+sqrt(rho(:,j+1))); % Roe's Enthalpy  
 a=sqrt((gamma-1.0)\*(H\_avg-0.5\*u\_avg^2)); % the speed of sound  
 beta=1/(2\*a^2);  
 R=[1 beta beta;u\_avg (u\_avg+a)\*beta (u\_avg-a)\*beta;0.5\*u\_avg^2 beta\*(H\_avg+a\*u\_avg) beta\*(H\_avg-a\*u\_avg)];  
 L=inv(R);  
 D=diag([abs(u\_avg) abs(u\_avg+a) abs(u\_avg-a)]);  
 %Apply Haartens entropy fix  
del=0.001;  
 for i=1:3  
 if(i==1||i==3)  
 if abs(D(i,i))<del  
 D(i,i)=0.5\*(D(i,i)^2/del+del);  
 end  
 end  
 end  
 wL=w(:,j+1);wR=w(:,j);  
 % Calculate F=0.5\*(F(WL}+F(WR))-R\*D\*L\*(WL-WR)  
 F(:,j)=0.5\*(F\_w(:,j)+F\_w(:,j+1))-0.5\*R\*D\*L\*(wL-wR);  
end

end