Project 2 - A finite volume program for solving the velocity potential equation

Rushi Faldu

MAE 456

May 1, 2026

Contents

List of Figures

List of Tables

1 Introduction

The objective of this project was to develop a 2D finite volume program for solving the velocity potential equation for 2 structured grids. Two geometries were studied in this project which include a grid bump and a half cylinder. The geometry of the flow for each was solved at free stream Mach number 0.01, 0.5, and 0.8. The solver used ghost-cell approach to solve for the boundary conditions, applied the Green-Gauss technique to solve for velocity using the potential fields, and used the iterative Guass-Seidel approach to converge the solution meaningfully.

2 Governing Equation and Flow Model

For this project, the governing equation for the compressible potential flow is:

$$\nabla \cdot (\rho \nabla \phi) = 0, \tag{1}$$

In the above mentioned equation, ϕ is the velocity potential for the grids and ρ is the density. The velocity components for finding the density are obtained from the velocity potential fields using the following equation:

$$u = \frac{\partial \phi}{\partial x}, \qquad v = \frac{\partial \phi}{\partial y}. \tag{2}$$

The density for each cell center was obtained using local velocity components using the following compressible flow relation:

$$\rho = \rho_\infty \left[1 + \frac{\gamma - 1}{2} \left(M_\infty^2 - \frac{u^2 + v^2}{a_\infty^2} \right) \right]^{\frac{1}{\gamma - 1}}, \tag{3}$$

In the above equation, ρ_∞ is the free-stream density of the flow, a_∞ is the free-stream speed of sound, M_∞ is the free-stream Mach number, and γ is the ratio of specific heats.

The free-stream conditions used in the completion of this project are standard atmospheric conditions:

$$T_\infty = 300K, \qquad P_\infty = 101325Pa, \qquad \gamma = 1.4, \qquad R = 287J/(kg \cdot K). \tag{4}$$

From these,

$$\rho_\infty = \frac{P_\infty}{RT_\infty}, \qquad a_\infty = \sqrt{\gamma R T_\infty}. \tag{5}$$

3 Numerical Method

3.1 Mesh and Geometric Quantities

The grids used in this project were read from the provided text files. Here is a plot of both the grids.

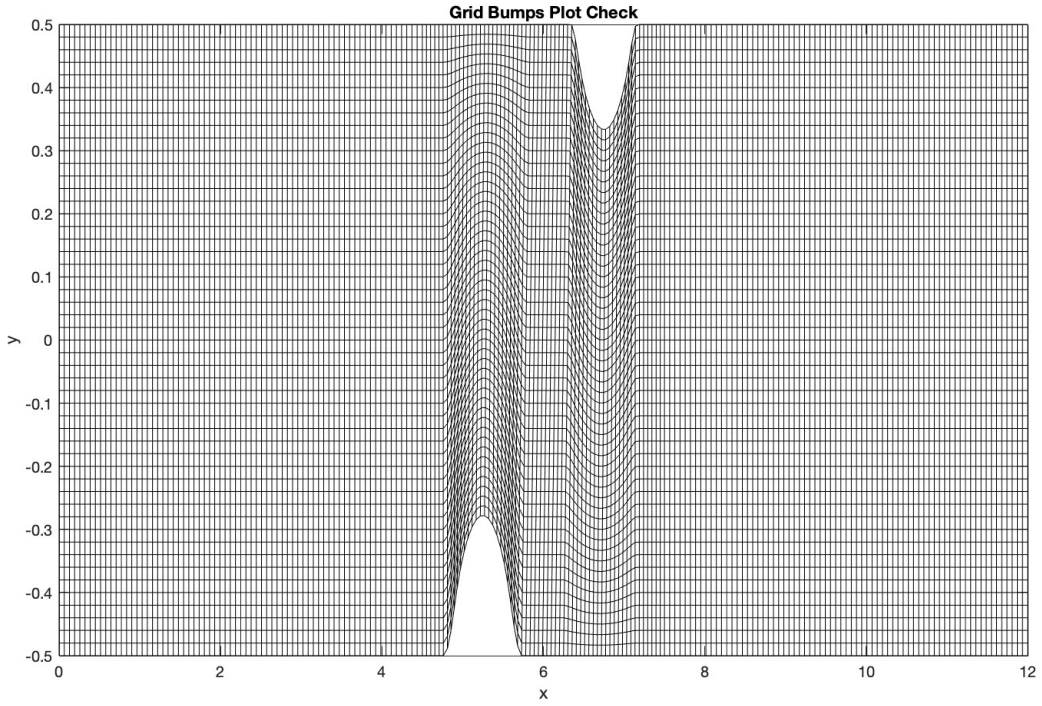

Figure 1: Grid Bumps Plot

These grid files were then structured into nodal arrays from where their geometric quantities were computed. Ghost cells were generated along all the outer boundaries so that proper boundary conditions could be enforced. The following geometric quantities were computed:

- cell-center coordinates,
- cell area,
- area-weighted outward face normal vectors,
- ghost-cell-center coordinates and areas.

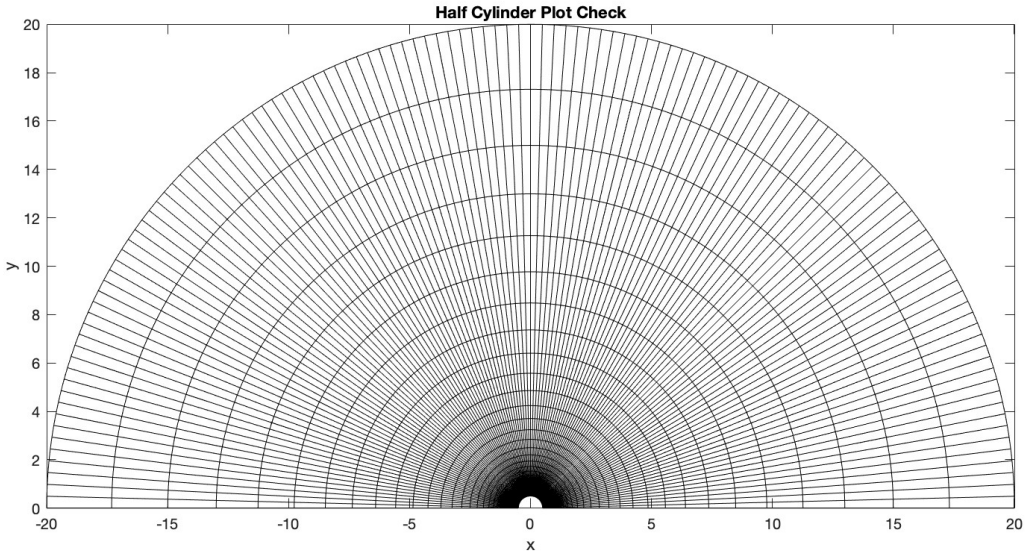

Figure 2: Half Cylinder Plot

3.2 Boundary Conditions

3.2.1 Grid 1: Bumps

For the bumps grid, the left and right boundaries were treated as far-field boundaries because of fluid flow and the top and bottom boundaries were treated as wall tangency. On the slip walls, ghost-cell potential values were assigned so that the normal velocity component for the wall is zero.

3.2.2 Grid 2: Half-Cylinder

For the half cylinder case, the curved body surface was treated as wall tengency whereas the outer boundaries were treated as far-field conditions. This allowed the body wall to enforce no penetration while the external boundaries represented approach to free-stream flow.

3.3 Velocity Calculation

The velocity was obtained from the potential field using Green-Gauss method. Face values of the potential were computed by averaging the values of all the adjacent cell centers. These were then used in the following equation:

$$\nabla\phi \approx \frac{1}{A}\sum_{f}\phi_f\vec{n}_f, \tag{6}$$

where A is the cell area, ϕ_f is the face-averaged potential, and $\vec{n}_f$ is the area-weighted face normal vector. This gave the velocity components directly as

$$u = \frac{\partial\phi}{\partial x}, \qquad v = \frac{\partial\phi}{\partial y}. \tag{7}$$

3.4 Residual and Iterative Updates

For each face, the flux coefficient was based on the average density, average adjacent cell area, and squared face-normal magnitude. The residual for each control volume was then formed from the sum of the face fluxes.

The potential was updated iteratively using a Gauss-Seidel method. The solver was considered converged when the relative residual dropped by at least four orders of magnitude.

4 program Structure

The program was organized into several functions each with their own individual use. All of the functions were then compiled in a main project code that had a similar core structure for both cases with minor differences. The main project code put the functions into an iterative loop till the required convergence was achieved.

- a geometry function for cell and ghost-cell geometry,
- boundary-condition functions for the bump and half-cylinder cases,
- a function to compute velocity and density from the potential,
- a residual/coefficient function,
- a potential-update function using Gauss-Seidel method,
- post-processing and plotting sections inside the main scripts.

5 Results and Discussion

5.1 Grid 1: Bumps

5.1.1 Mach 0.01

The grid bump solution at Mach = 0.01 shows almost incompressible flow. The residual decreases smoothly to convergence, while the velocity magnitude increases only locally in the constricted bump region to 4.4 m/s and remains close to the free-stream value elsewhere. The wall Mach-number distributions are close to the free-stream Mach number, and the pressure coefficient shows moderate suction in the narrow passages where the flow accelerates.

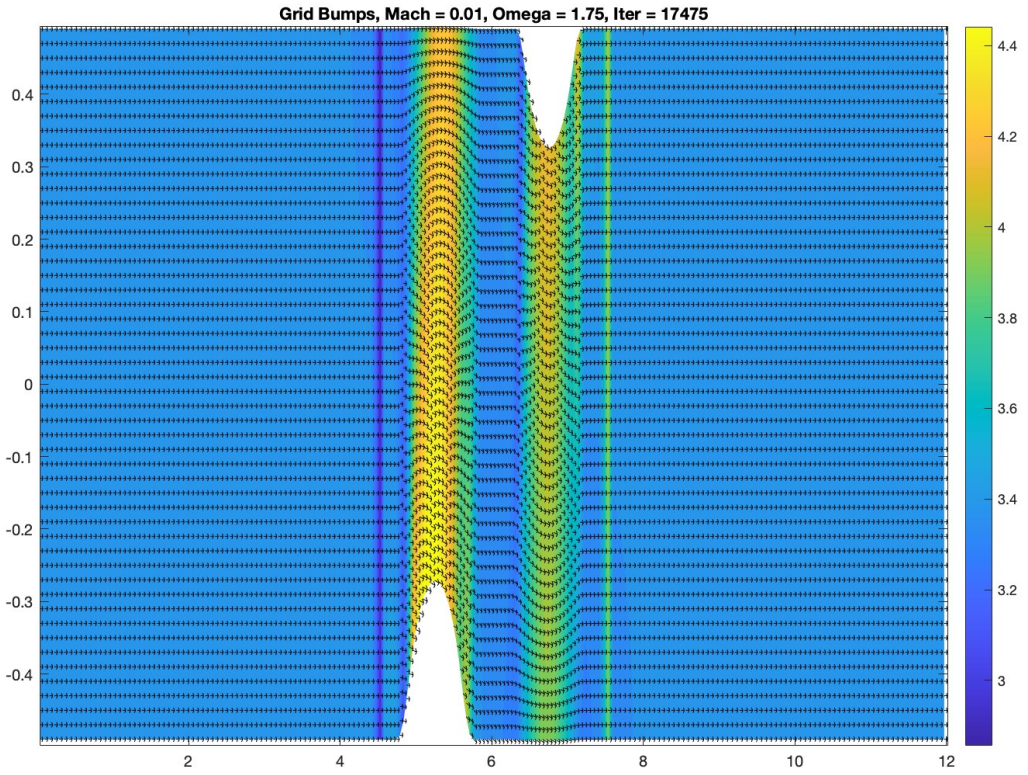

Figure 3: Grid Bumps Velocity Plot

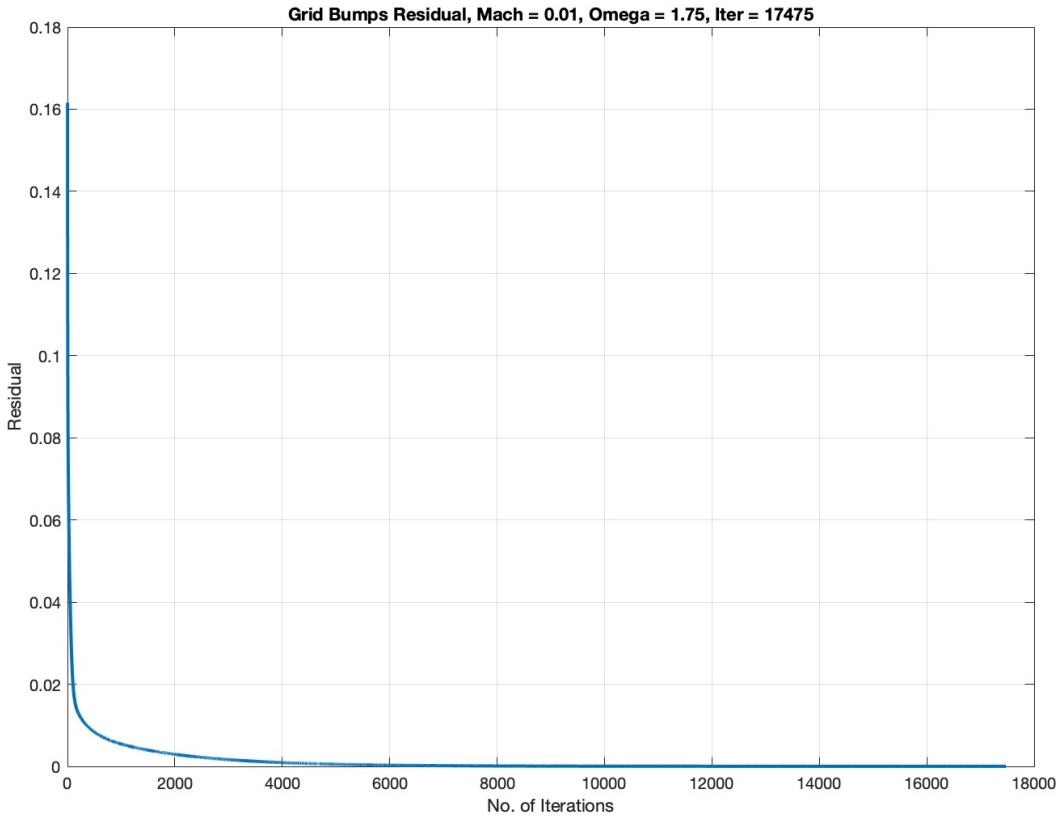

Figure 4: Grid Bumps Residual Plot

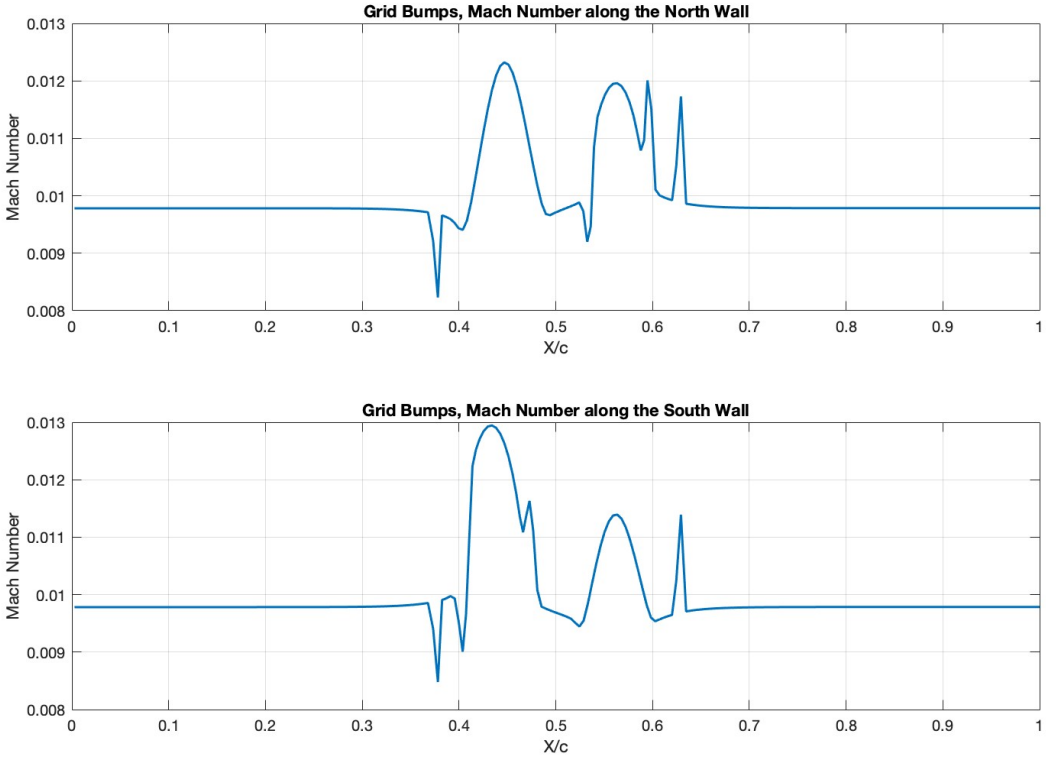

Figure 5: Grid Bumps Mach Plot

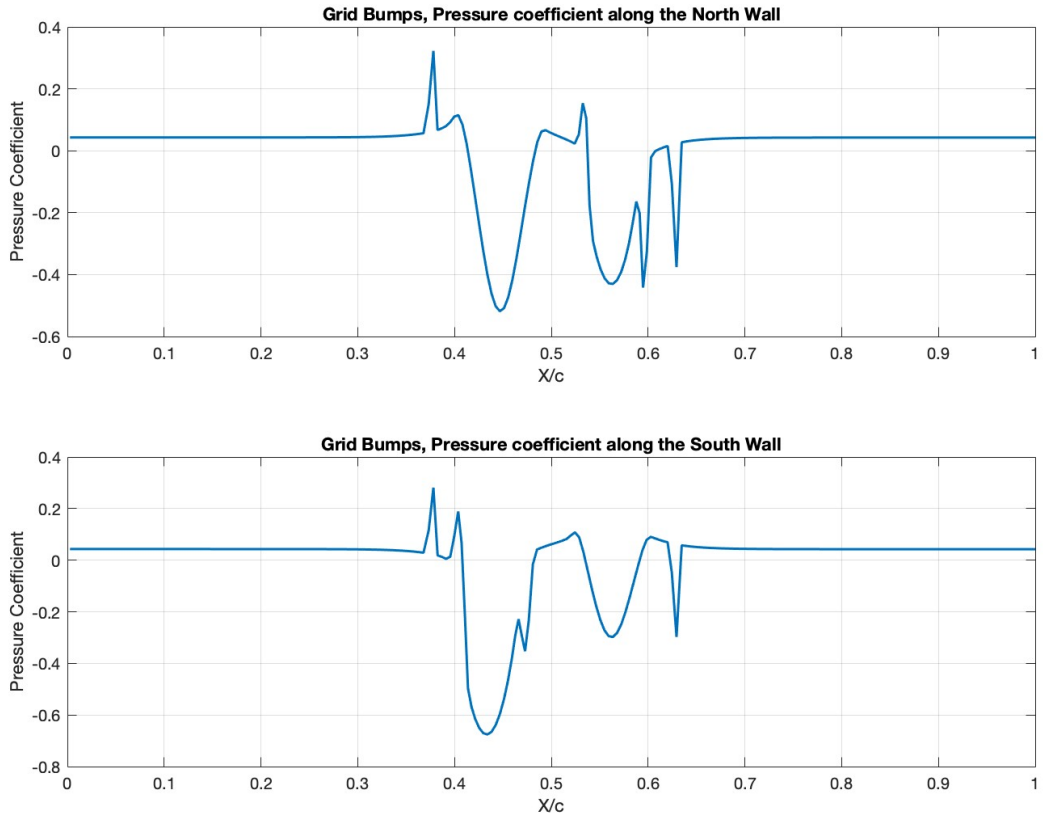

Figure 6: Grid Bumps Coefficient of Pressure Plot

5.1.2 Mach 0.5

At Mach = 0.5, the same overall flow structure is maintained, but the local acceleration through the bump region becomes much stronger with upto 240 m/s. The wall Mach-number peaks are significantly larger than in the low-Mach case, and the pressure coefficient shows stronger negative values in the same regions.

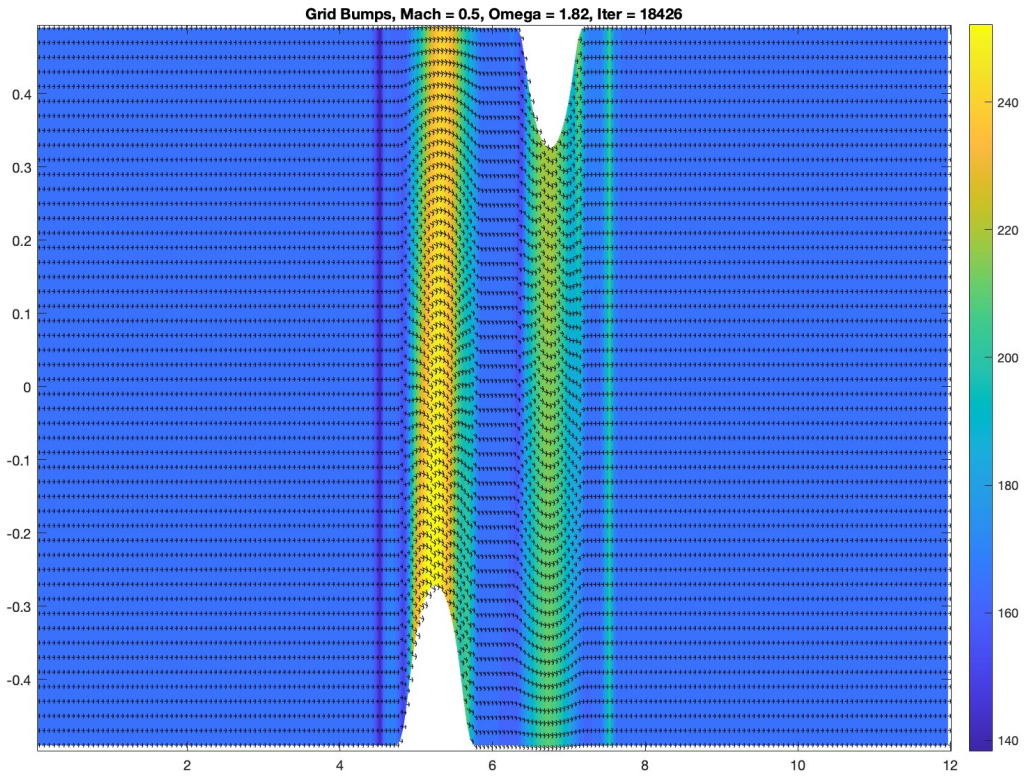

Figure 7: Grid Bumps Velocity Plot

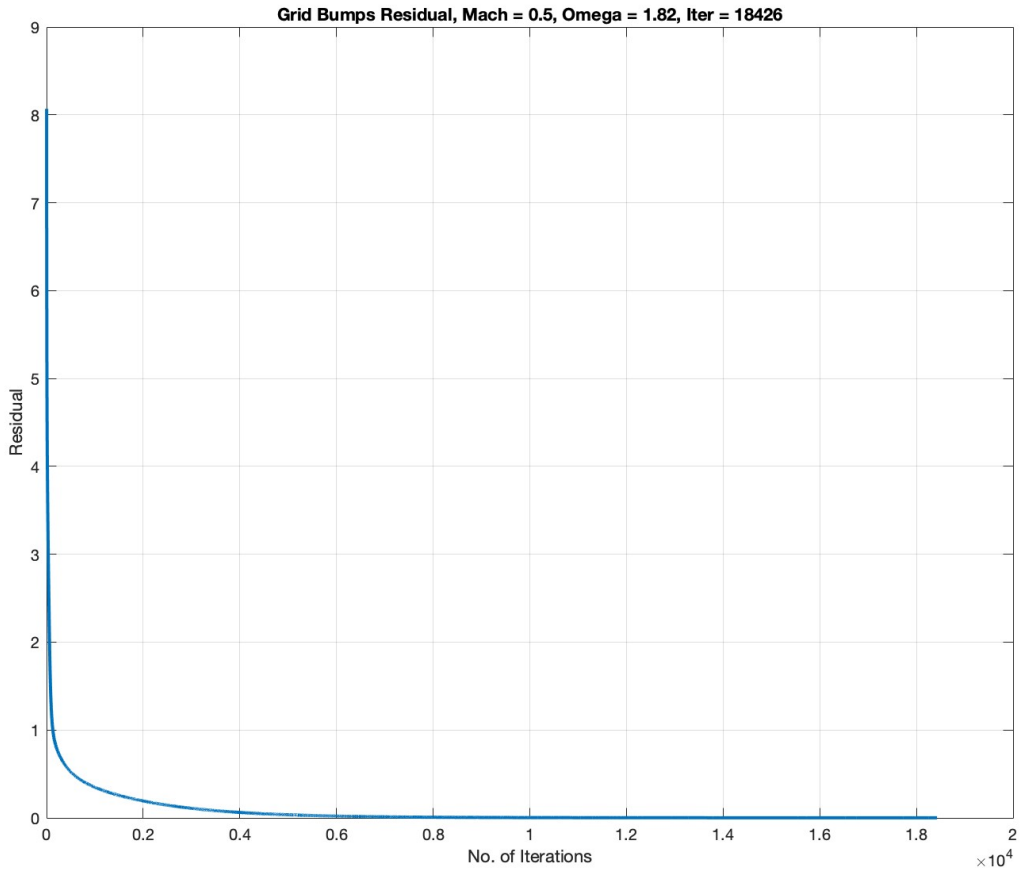

Figure 8: Grid Bumps Residual Plot

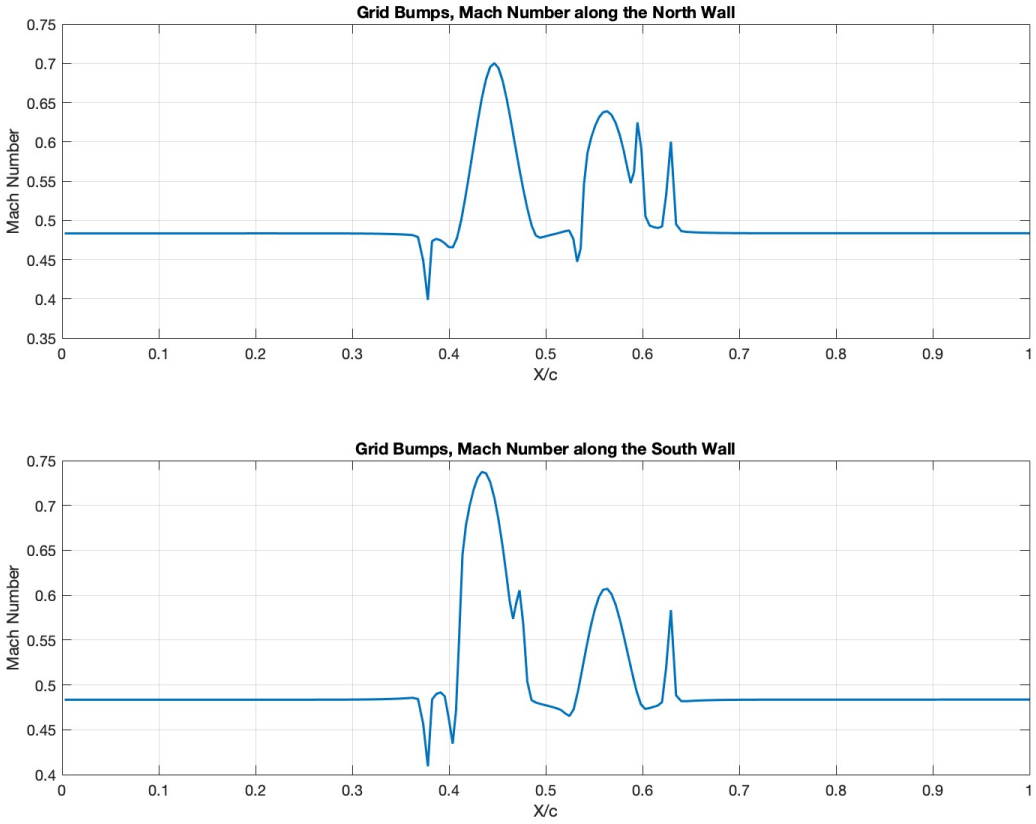

Figure 9: Grid Bumps Mach Plot

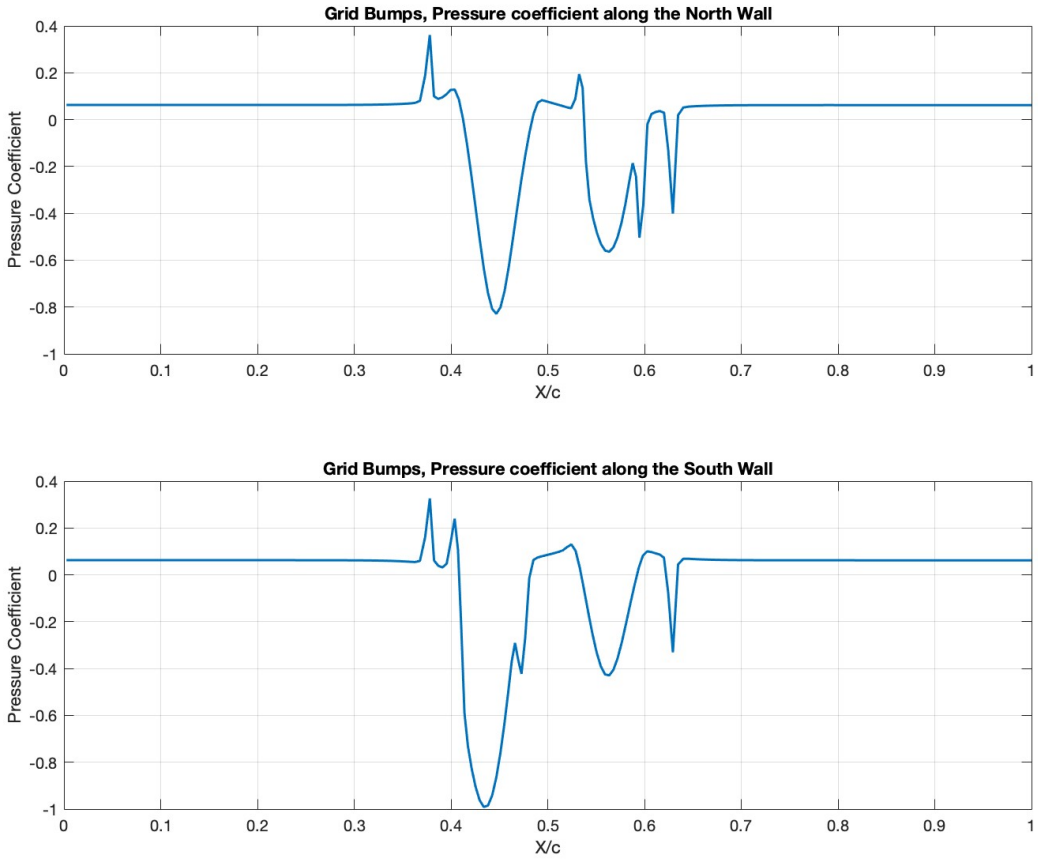

Figure 10: Grid Bumps Coefficient of Pressure Plot

5.1.3 Mach 0.8

At Mach = 0.8, the strongest compressibility effects are observed in the bump channel with velocities of 340 m/s. The wall Mach-number distributions reach the largest peaks and approach or slightly exceed unity locally, indicating very strong acceleration in the constricted passages. The pressure coefficient shows the largest suction levels, especially near the bump region. Although the flow field remains physically reasonable, the residual converges much more slowly than in the lower-Mach cases, confirming that the high-Mach solution is the most numerically difficult of the three bump-channel runs.

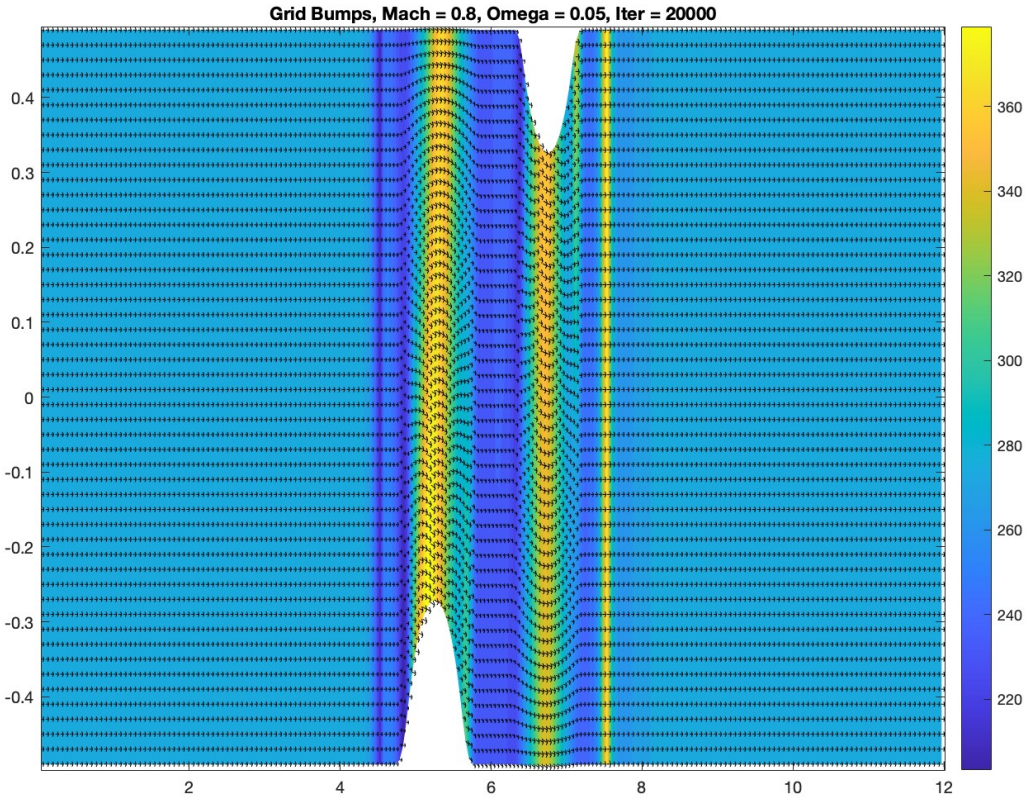

Figure 11: Grid Bumps Velocity Plot

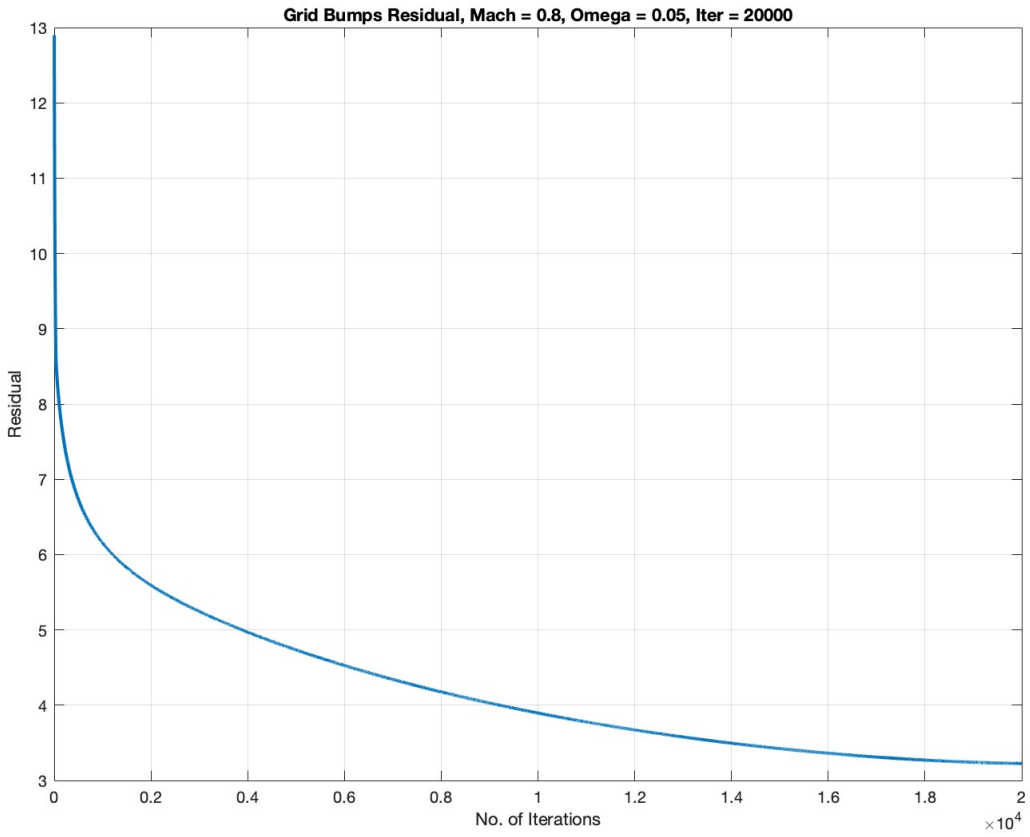

Figure 12: Grid Bumps Residual Plot

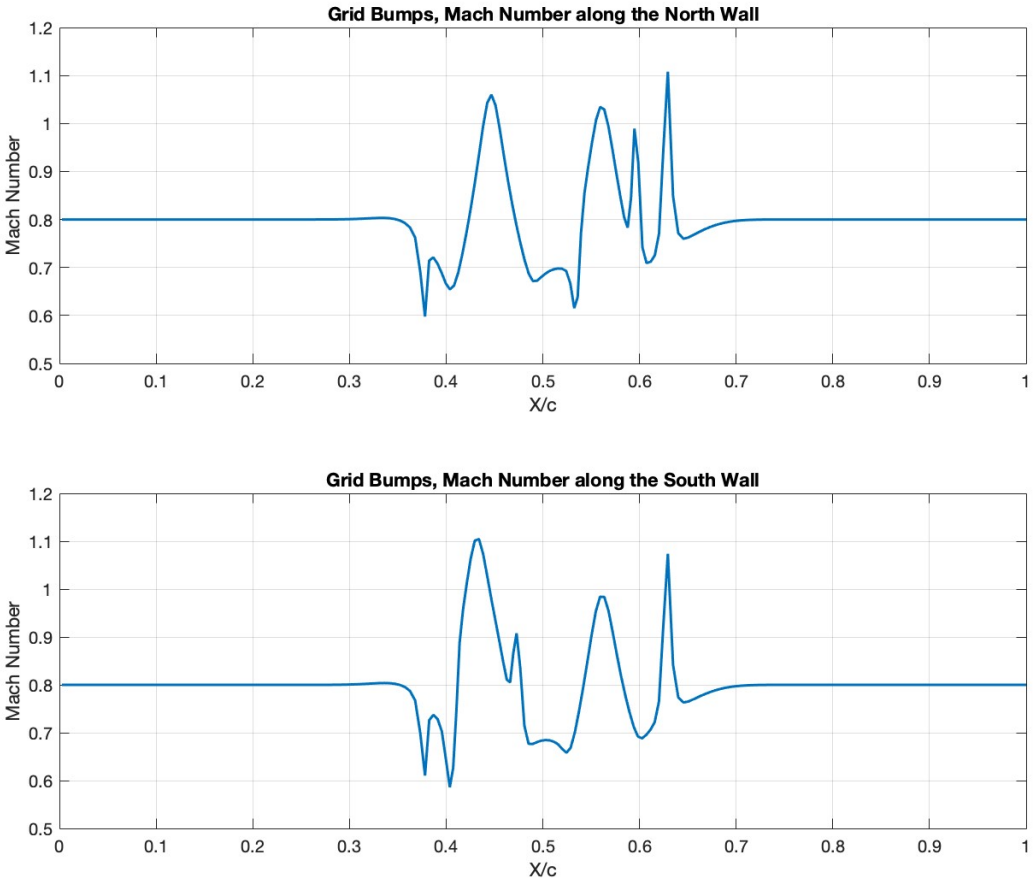

Figure 13: Grid Bumps Mach Plot

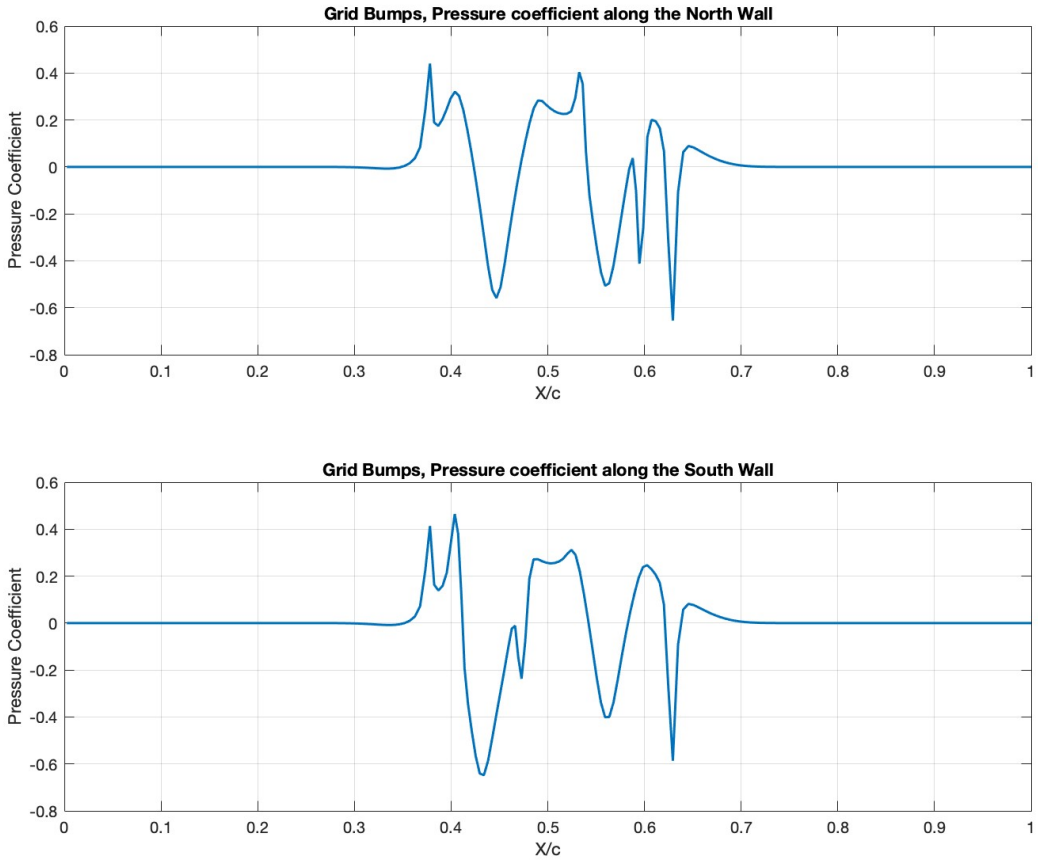

Figure 14: Grid Bumps Coefficient of Pressure Plot

5.2 Grid 2: Half-Cylinder

5.2.1 Mach 0.01

5.2.2 Mach 0.5

5.2.3 Mach 0.8

6 Code

The code starts from next page onwards.

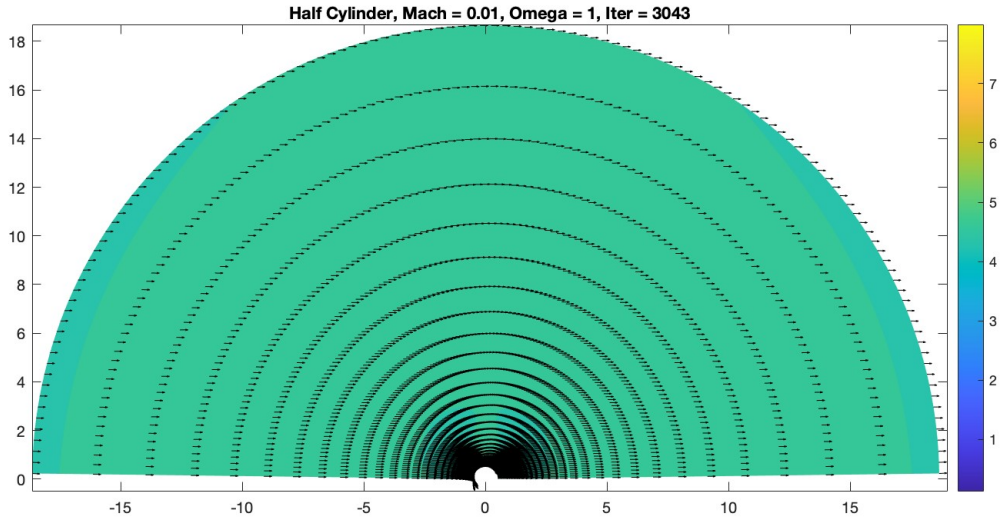

Figure 15: Half Cylinder Velocity Plot

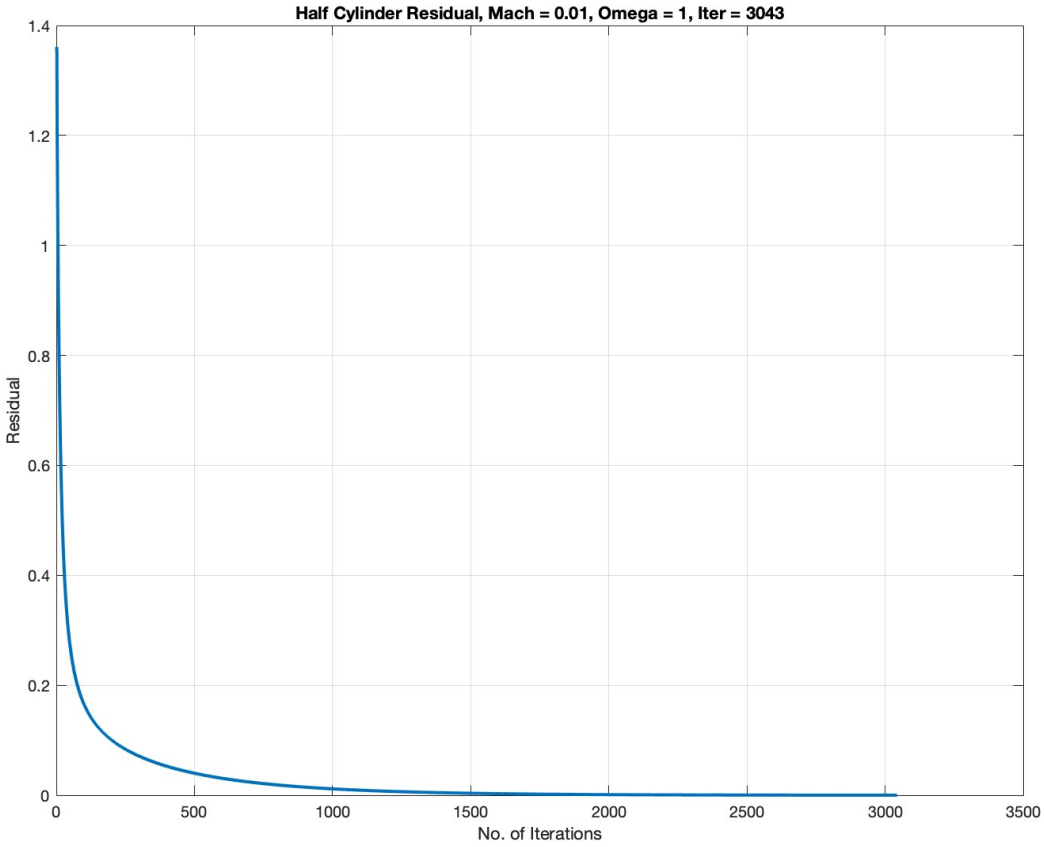

Figure 16: Half Cylinder Residual Plot

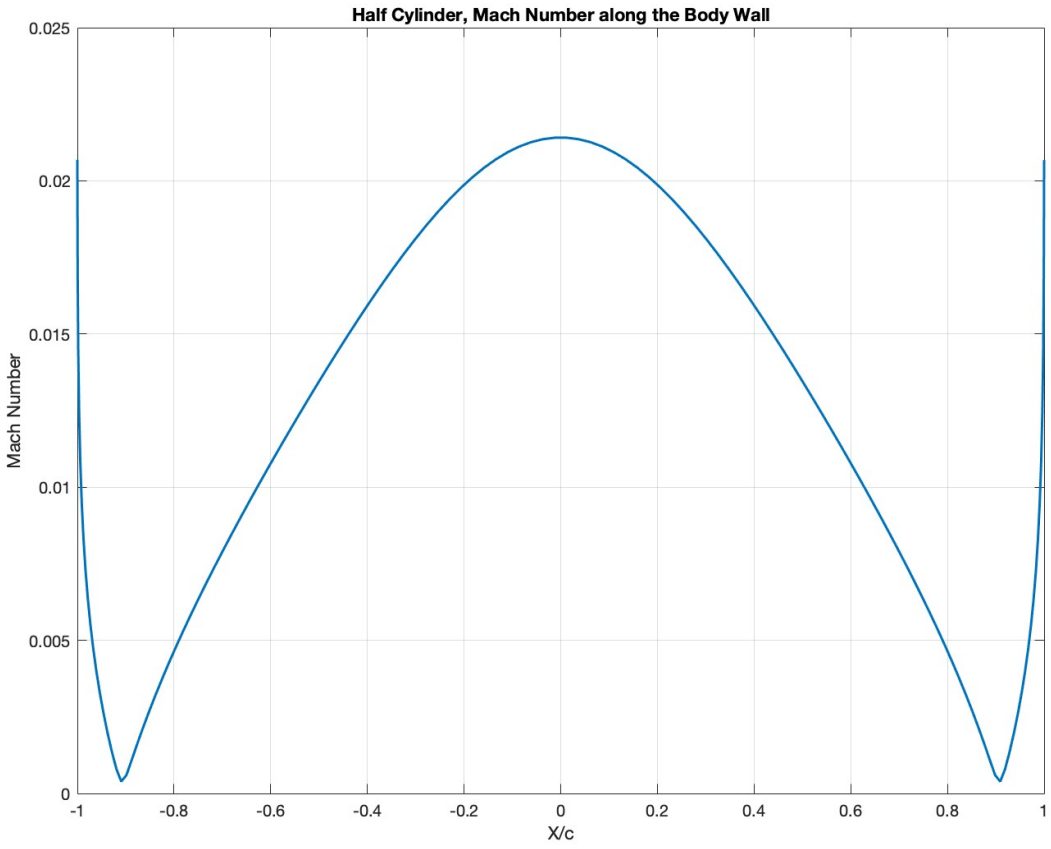

Figure 17: Half Cylinder Mach Plot

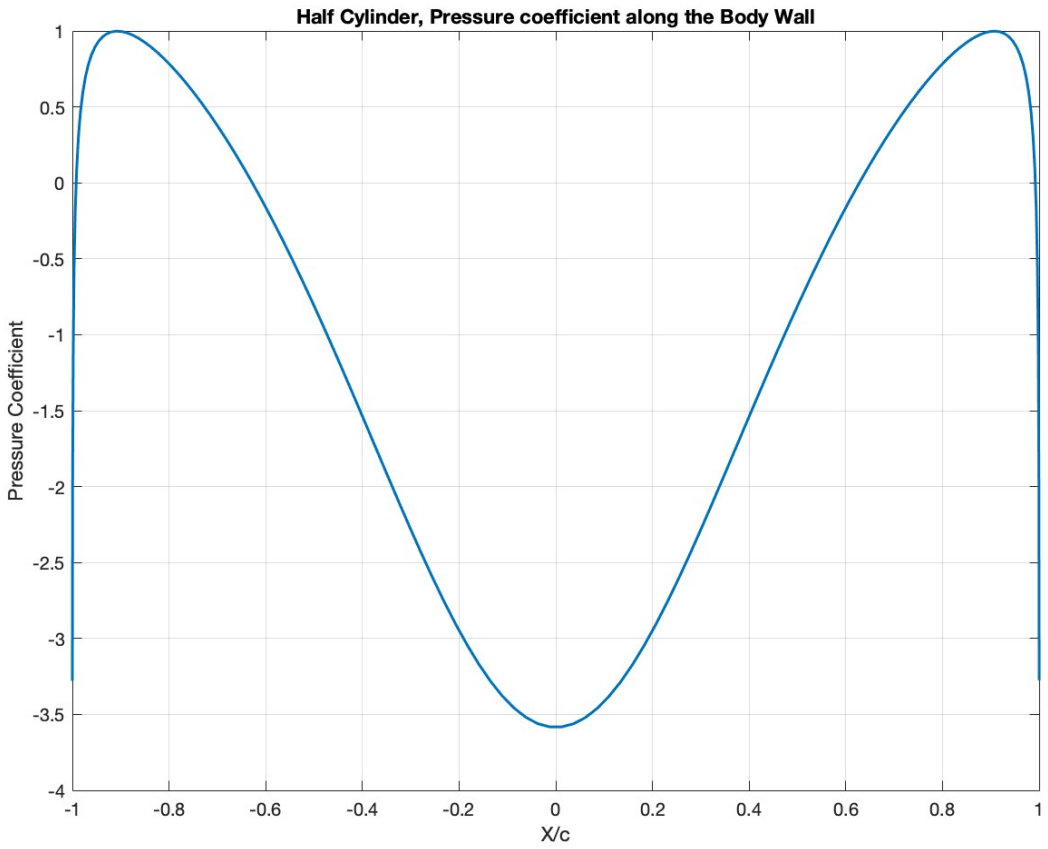

Figure 18: Half Cylinder Coefficient of Pressure Plot

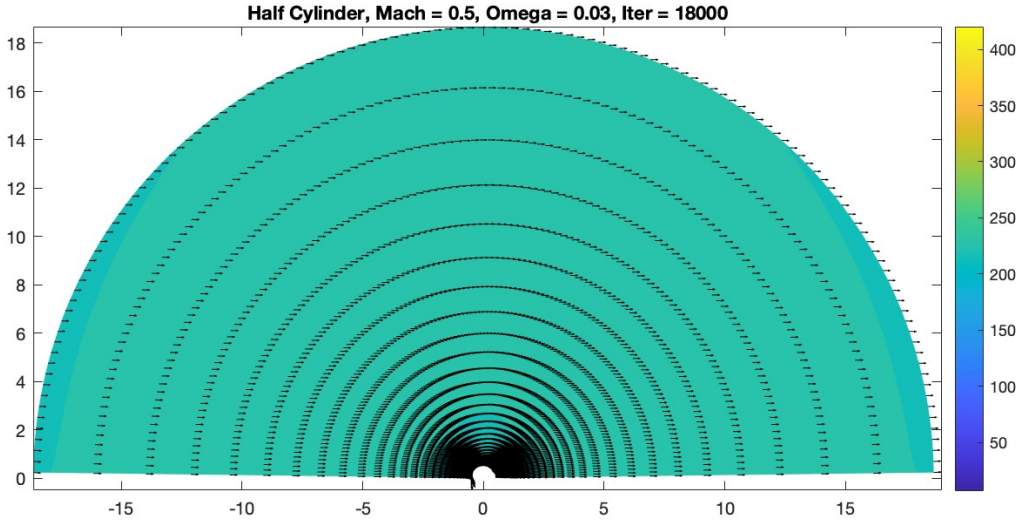

Figure 19: Half Cylinder Velocity Plot

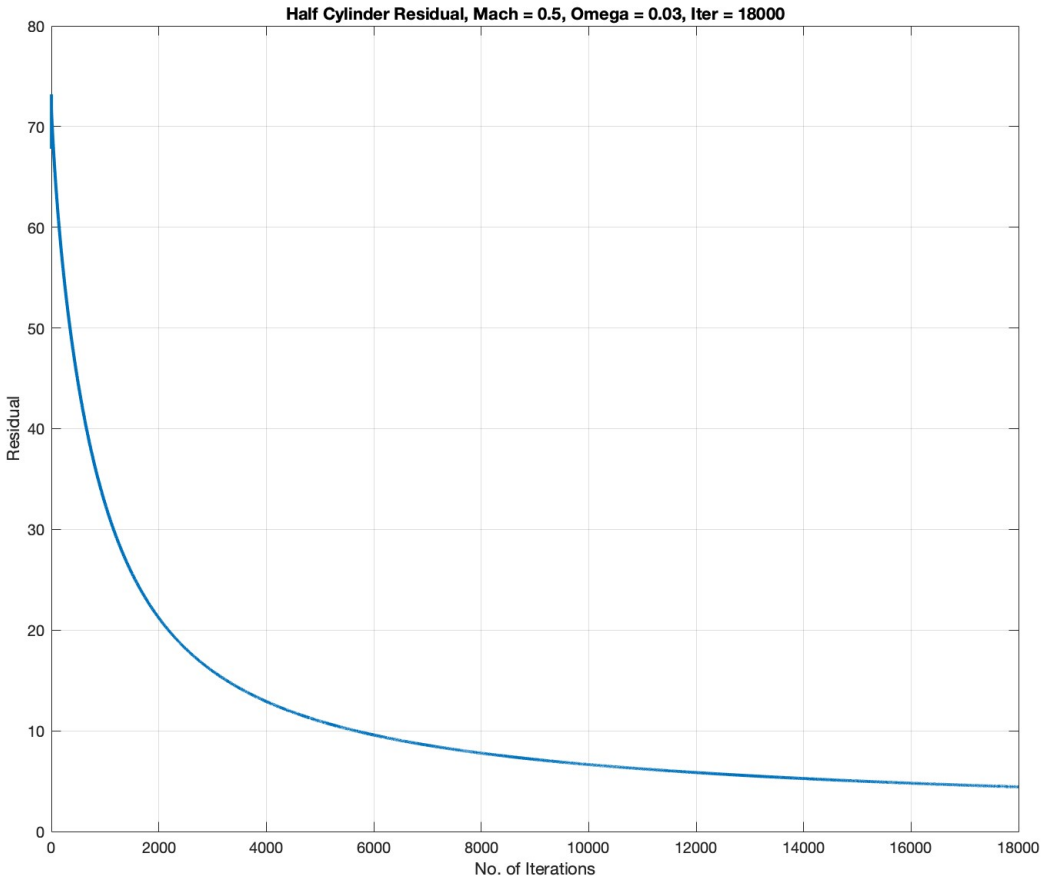

Figure 20: Half Cylinder Residual Plot

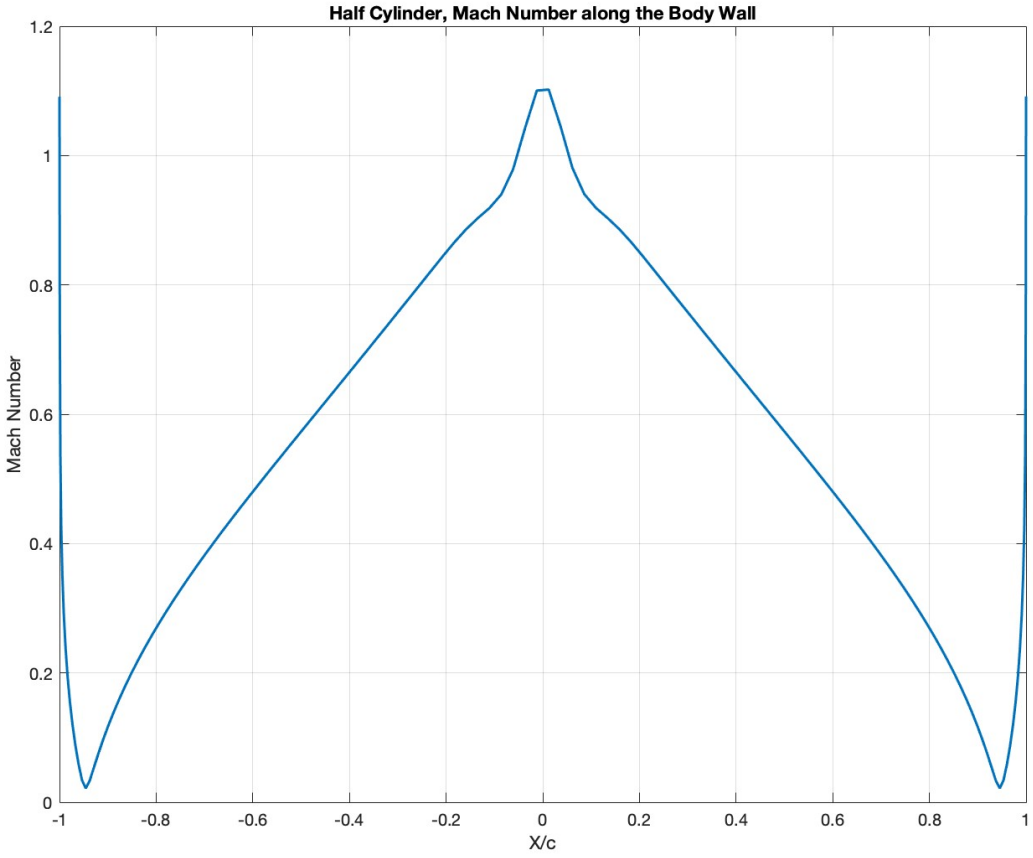

Figure 21: Half Cylinder Mach Plot

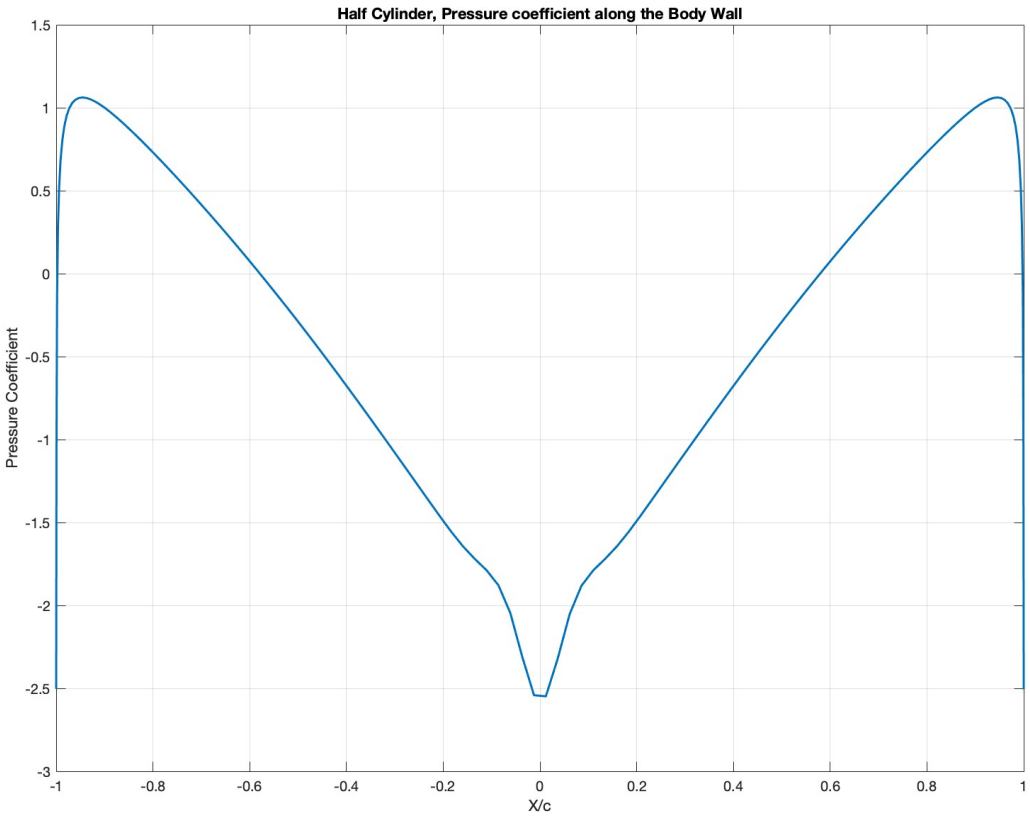

Figure 22: Half Cylinder Coefficient of Pressure Plot

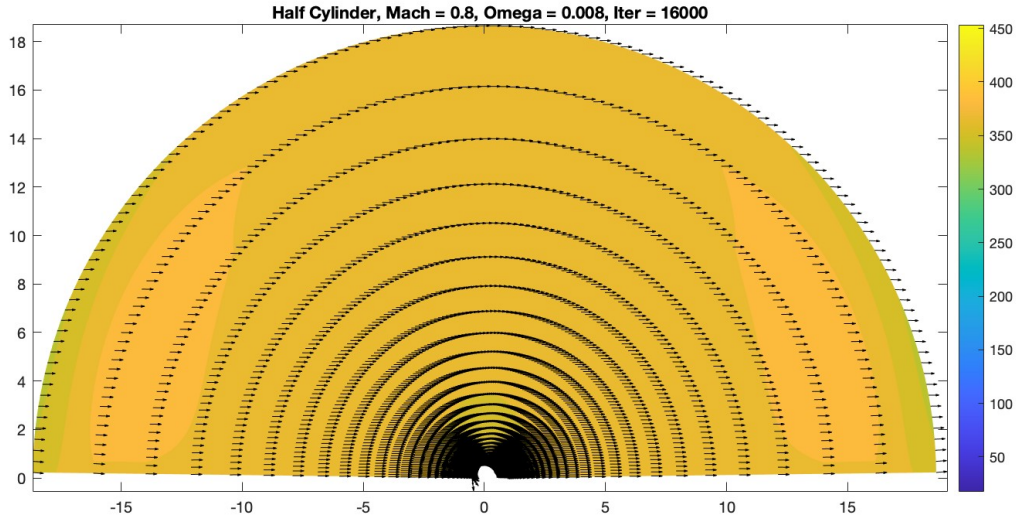

Figure 23: Half Cylinder Velocity Plot

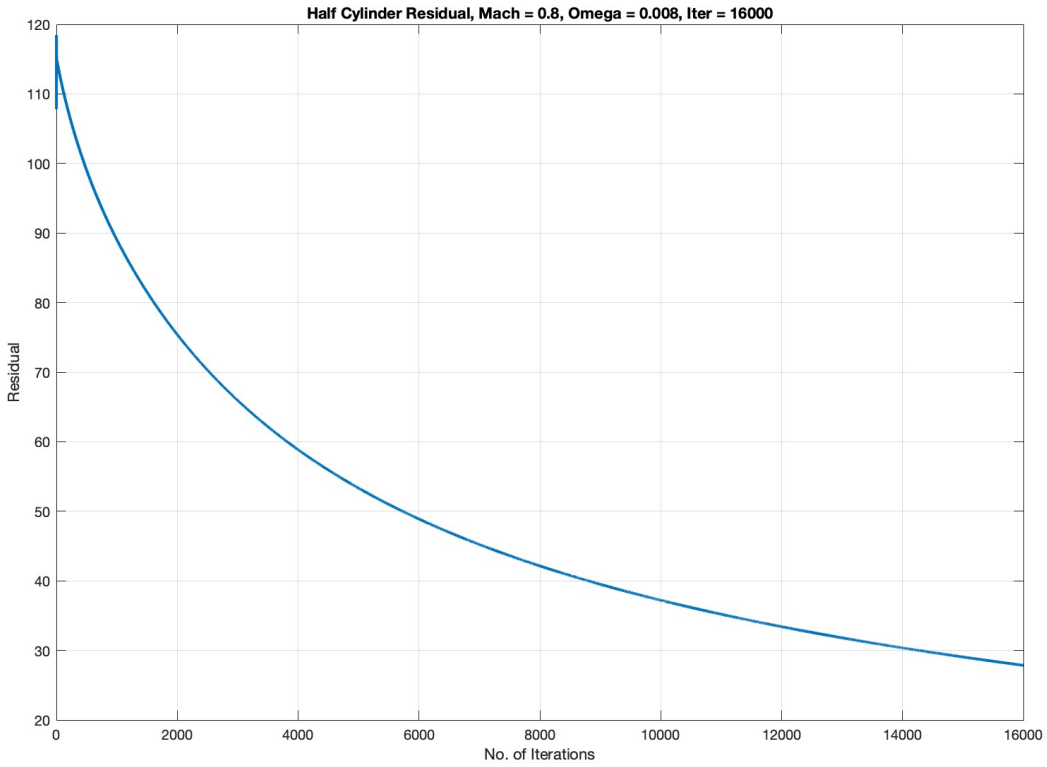

Figure 24: Half Cylinder Residual Plot

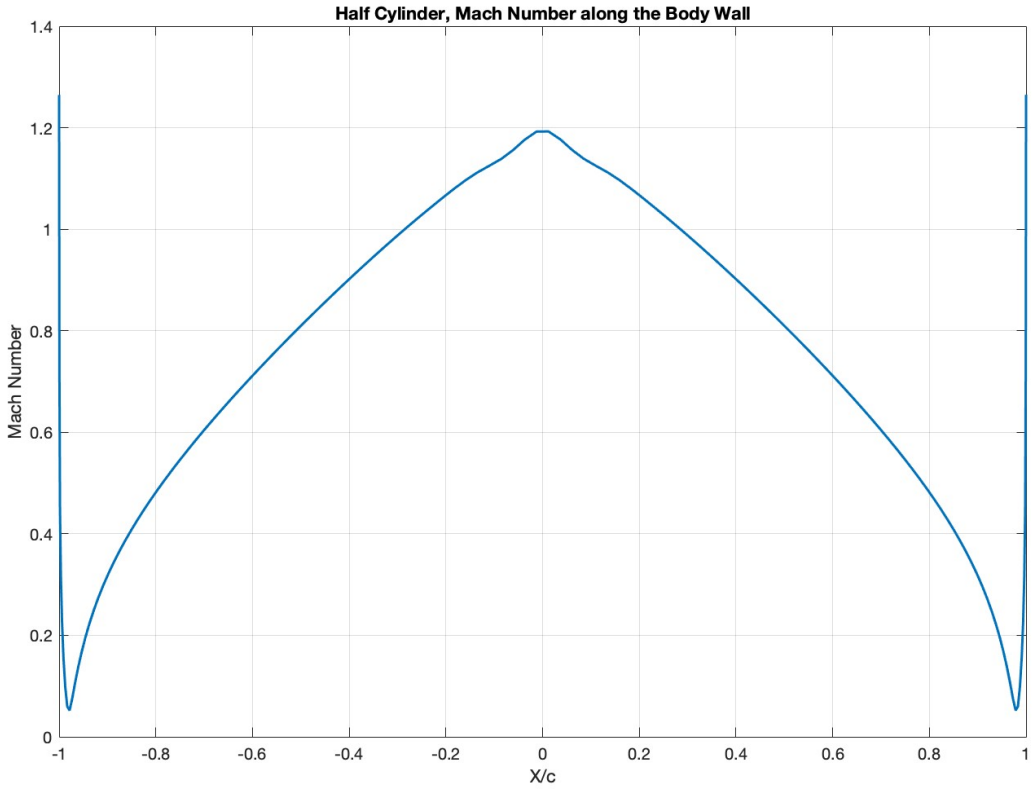

Figure 25: Half Cylinder Mach Plot

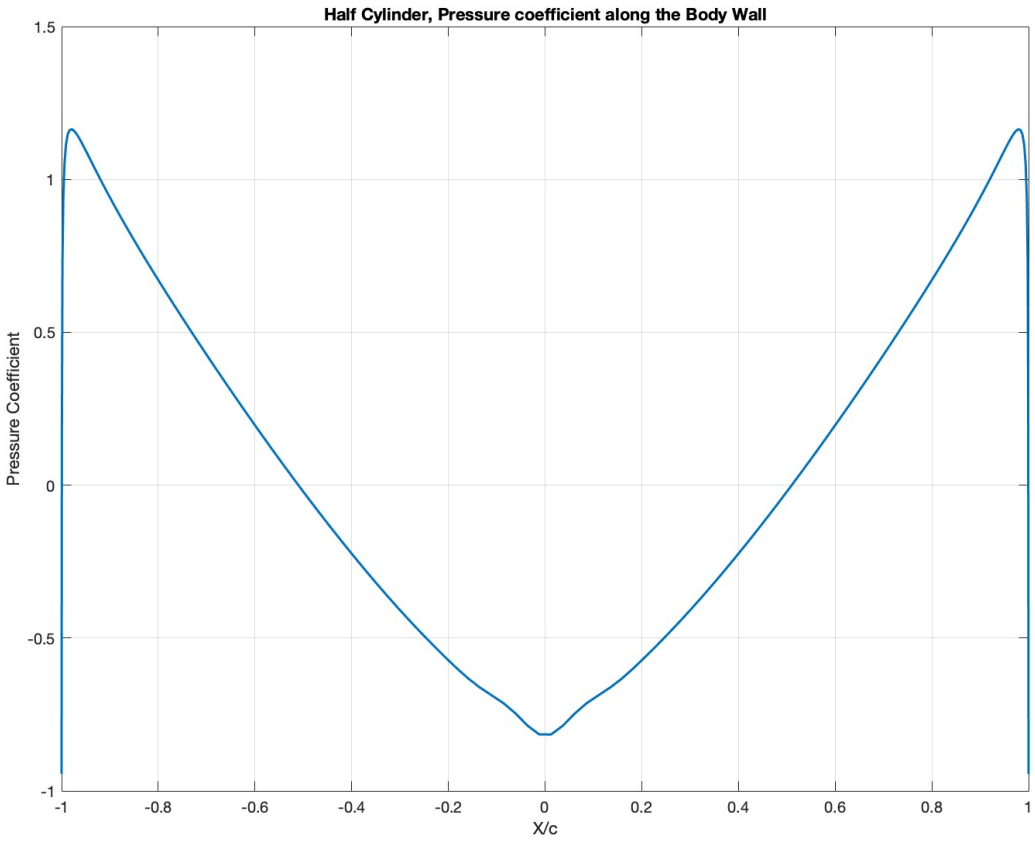

Figure 26: Half Cylinder Coefficient of Pressure Plot

Table of Contents


```
clear; clc; close all
```

Mach 0.01 Grid Bumps

```
clear; clc; close all

% Reading the mesh file
mesh = readmatrix("grid.bumps.txt");

% Reading mesh dimensions
npx = mesh(1,1); % Dimension of mesh
npy = mesh(1,2); % Dimension of mesh
mesh(1,:) = []; % deleting first row

% Sorting mesh coordinates
ndimn = 2;
coord = zeros(ndimn,npx,npy); % pre-allocating coordinate matrix

num = 1; % counts the iteration
for j = 1:npy
    for i = 1:npx
        for k = 1:ndimn
            coord(k,i,j) = mesh(num,k);
        end
        num = num + 1; % updating variable
    end
end

% Subtracting one from number of nodes given in rows in and columns to
% obtain the number of cells in row and column
ncx = npx - 1; % number of cells in row
ncy = npy - 1; % number of cells in column

%Sanity grid plot check
x_val = squeeze(coord(1,:,:));
y_val = squeeze(coord(2,:,:));

figure(1)
plot(x_val,y_val,'k-');
hold on
plot(x_val',y_val','k-');
xlabel('x')
```

```
ylabel('y')
title('Grid Bumps Plot Check')

% Geometry Calculations Using geo_calc.m function
geoel = geo_calc(coord);

% Setting up inital conditions
% Using standard atmospheric conditions as instructed. Gamma for air is
% about 1.4 and R for air is 287 J/Kg*K. Temperature and pressure at
% freestream conidtions are given along with a Mach range.

T_inf = 300; % K
P_inf = 101325; % Pa
gamma = 1.4; % For air
R = 287; % J/Kg*K
Mach = 0.01;

% Freestream density
rho_inf = P_inf/(R*T_inf); % ideal gas law kg/m^3

% Iteration settings
omega = 1.75;
iter = 20000;
tol = 1e-4; % convergence criterion

% Speed of sound calculations
a_inf = sqrt(gamma*R*T_inf); % m/s

% Finding initial speed using freestream speed of sound and Mach numbers.u_inf
% is non-zero because I assuming the fluid freestream moves from left to
% right along the x-axis. v_inf is zero because I am assuming zero
% freestream velocity upwards or in the y-axis.
u_inf = a_inf*Mach; % m/s
v_inf = 0;

% Creating array for unknowns such as phi and rho and velocity. Adding
% 2 to the number of cells for ghost cell layer around all the walls.
var = 4; % 1 phi, 2 u-comp, 3 v-comp, 4 density
unkno = zeros(var, ncx+2, ncy+2);

% Initial guess for velocity potential based on freestream velocity
% components. The velocity potential for each cell will be at the cell
% centeroid. Would help this way because the array is cell centered in
% scrap code.
for k = 1:ncy
    for j = 1:ncx
        x_cent = geoel(1,j+1,k+1); % centroid for cells
        y_cent = geoel(2,j+1,k+1); % centroid for cells

        % Same initial guess for all real cells.
        unkno(1,j+1,k+1) = u_inf*x_cent + v_inf*y_cent;
    end
end
```

```
% Initializing density values
unkno(4,2:ncx+1,2:ncy+1) = rho_inf;

% Initializing residual history
resi_hist = zeros(iter,1);

% Iteration loop for convergence
for num = 1:iter

    % Ghost cell boundary conditions application
    unkno = boundary_cond(unkno,geoel,u_inf,v_inf);

    % Calculating velocity and density values from phi values for both real
    % and ghost cells
    unkno = DensityfromPhi(unkno,geoel,gamma,Mach,a_inf,rho_inf);

    % Calculating the residuals and the coefficients
    [resi,coeff] = resi_calc(unkno,geoel);

    % Updating phi using Gauss-Seidel Method
    [unkno,dphi,resi_n] = updatePhi(unkno,resi,coeff,omega);

    % Storing residual history
    resi_hist(num) = resi_n;

    % Convergence check
    if num == 1
        resi0 = resi_n;
    end

    resi_temp = resi_n/resi0;

    if resi_temp < tol
        resi_hist = resi_hist(1:num);
        break
    end
end

if num == iter
    resi_hist = resi_hist(1:iter);
end

% Extracting data for plotting
Xc = squeeze(geoel(1,2:ncx+1,2:ncy+1));
Yc = squeeze(geoel(2,2:ncx+1,2:ncy+1));
Uc = squeeze(unkno(2,2:ncx+1,2:ncy+1));
Vc = squeeze(unkno(3,2:ncx+1,2:ncy+1));
rhoc = squeeze(unkno(4,2:ncx+1,2:ncy+1));

% Velocity magnitude
Vmag = sqrt(Uc.^2 + Vc.^2);

% Local mach number
Mach_local = Vmag./a_inf;
```

```
% Pressure coefficient calculation
numerator = 1 + (gamma-1)/2 * Mach^2;
denomenator = 1 + (gamma-1)/2 * Mach_local.^2;
Cp = 2/(gamma*Mach^2).*((numerator./denomenator).^(gamma/(gamma-1)) - 1);

% Velocity and mach for slip walls
VN = sqrt(Uc(:,end).^2 + Vc(:,end).^2);    % North wall
VS = sqrt(Uc(:,1).^2   + Vc(:,1).^2);      % South wall

MN = VN ./ a_inf;
MS = VS ./ a_inf;

denominatorN = 1 + (gamma-1)/2 * MN.^2;
denominatorS = 1 + (gamma-1)/2 * MS.^2;

CpN = 2/(gamma*Mach^2) .* ((numerator ./ denominatorN).^(gamma/(gamma-1)) -
1);
CpS = 2/(gamma*Mach^2) .* ((numerator ./ denominatorS).^(gamma/(gamma-1)) -
1);

% X/c along the walls
XN = Xc(:,end) ./ max(Xc(:,end));
XS = Xc(:,1)   ./ max(Xc(:,1));

% sort for clean plotting
[XN, indN] = sort(XN);
MN = MN(indN);
CpN = CpN(indN);

[XS, indS] = sort(XS);
MS = MS(indS);
CpS = CpS(indS);

% Plots
figure()
plot(1:length(resi_hist),resi_hist,'LineWidth',2)
grid on
xlabel('No. of Iterations')
ylabel('Residual')
title('Grid Bumps Residual, Mach = 0.01, Omega = 1.75, Iter = 17475')

figure()
contourf(Xc',Yc',Vmag',30,'LineStyle','none')
hold on
colorbar
quiver(Xc',Yc',Uc',Vc','k')
%axis equal tight
title('Grid Bumps, Mach = 0.01, Omega = 1.75, Iter = 17475')

figure()
subplot(2,1,1)
plot(XN,MN,'LineWidth',1.5)
xlabel('X/c')
```

```
ylabel('Mach Number')
title('Grid Bumps, Mach Number along the North Wall')
grid on

subplot(2,1,2)
plot(XS,MS,'LineWidth',1.5)
xlabel('X/c')
ylabel('Mach Number')
title('Grid Bumps, Mach Number along the South Wall')
grid on

figure()
subplot(2,1,1)
plot(XN,CpN,'LineWidth',1.5)
xlabel('X/c')
ylabel('Pressure Coefficient')
title('Grid Bumps, Pressure coefficient along the North Wall')
grid on

subplot(2,1,2)
plot(XS,CpS,'LineWidth',1.5)
xlabel('X/c')
ylabel('Pressure Coefficient')
title('Grid Bumps, Pressure coefficient along the South Wall')
grid on
```

Mach 0.5 Grid Bumps

```
clear; clc; close all

% Reading the mesh file
mesh = readmatrix("grid.bumps.txt");

% Reading mesh dimensions
npx = mesh(1,1); % Dimension of mesh
npy = mesh(1,2); % Dimension of mesh
mesh(1,:) = []; % deleting first row

% Sorting mesh coordinates
ndimn = 2;
coord = zeros(ndimn,npx,npy); % pre-allocating coordinate matrix

num = 1; % counts the iteration
for j = 1:npy
    for i = 1:npx
        for k = 1:ndimn
            coord(k,i,j) = mesh(num,k);
        end
        num = num + 1; % updating variable
    end
end

% Subtracting one from number of nodes given in rows in and columns to
```

```
% obtain the number of cells in row and column
ncx = npx - 1; % number of cells in row
ncy = npy - 1; % number of cells in column

%Sanity grid plot check
x_val = squeeze(coord(1,:,:));
y_val = squeeze(coord(2,:,:));

figure(1)
plot(x_val,y_val,'k-');
hold on
plot(x_val',y_val','k-');
xlabel('x')
ylabel('y')
title('Grid Bumps Plot Check')

% Geometry Calculations Using geo_calc.m function
geoel = geo_calc(coord);

% Setting up inital conditions
% Using standard atmospheric conditions as instructed. Gamma for air is
% about 1.4 and R for air is 287 J/Kg*K. Temperature and pressure at
% freestream conidtions are given along with a Mach range.

T_inf = 300; % K
P_inf = 101325; % Pa
gamma = 1.4; % For air
R = 287; % J/Kg*K
Mach = 0.5;

% Freestream density
rho_inf = P_inf/(R*T_inf); % ideal gas law kg/m^3

% Iteration settings
omega = 1.82;
iter = 20000;
tol = 1e-4; % convergence criterion

% Speed of sound calculations
a_inf = sqrt(gamma*R*T_inf); % m/s

% Finding initial speed using freestream speed of sound and Mach numbers.u_inf
% is non-zero because I assuming the fluid freestream moves from left to
% right along the x-axis. v_inf is zero because I am assuming zero
% freestream velocity upwards or in the y-axis.
u_inf = a_inf*Mach; % m/s
v_inf = 0;

% Creating array for unknowns such as phi and rho and velocity. Adding
% 2 to the number of cells for ghost cell layer around all the walls.
var = 4; % 1 phi, 2 u-comp, 3 v-comp, 4 density
unkno = zeros(var, ncx+2, ncy+2);

% Initial guess for velocity potential based on freestream velocity
```

```
% components. The velocity potential for each cell will be at the cell
% centeroid. Would help this way because the array is cell centered in
% scrap code.
for k = 1:ncy
    for j = 1:ncx
        x_cent = geoel(1,j+1,k+1); % centroid for cells
        y_cent = geoel(2,j+1,k+1); % centroid for cells

        % Same initial guess for all real cells.
        unkno(1,j+1,k+1) = u_inf*x_cent + v_inf*y_cent;
    end
end

% Initializing density values
unkno(4,2:ncx+1,2:ncy+1) = rho_inf;

% Initializing residual history
resi_hist = zeros(iter,1);

% Iteration loop for convergence
for num = 1:iter

    % Ghost cell boundary conditions application
    unkno = boundary_cond(unkno,geoel,u_inf,v_inf);

    % Calculating velocity and density values from phi values for both real
    % and ghost cells
    unkno = DensityfromPhi(unkno,geoel,gamma,Mach,a_inf,rho_inf);

    % Calculating the residuals and the coefficients
    [resi,coeff] = resi_calc(unkno,geoel);

    % Updating phi using Gauss-Seidel Method
    [unkno,dphi,resi_n] = updatePhi(unkno,resi,coeff,omega);

    % Storing residual history
    resi_hist(num) = resi_n;

    % Convergence check
    if num == 1
        resi0 = resi_n;
    end

    resi_temp = resi_n/resi0;

    if resi_temp < tol
        resi_hist = resi_hist(1:num);
        break
    end
end

if num == iter
    resi_hist = resi_hist(1:iter);
end
```

```
% Extracting data for plotting
Xc = squeeze(geoel(1,2:ncx+1,2:ncy+1));
Yc = squeeze(geoel(2,2:ncx+1,2:ncy+1));
Uc = squeeze(unkno(2,2:ncx+1,2:ncy+1));
Vc = squeeze(unkno(3,2:ncx+1,2:ncy+1));
rhoc = squeeze(unkno(4,2:ncx+1,2:ncy+1));

% Velocity magnitude
Vmag = sqrt(Uc.^2 + Vc.^2);

% Local mach number
Mach_local = Vmag./a_inf;

% Pressure coefficient calculation
numerator = 1 + (gamma-1)/2 * Mach^2;
denomenator = 1 + (gamma-1)/2 * Mach_local.^2;
Cp = 2/(gamma*Mach^2).*((numerator./denomenator).^(gamma/(gamma-1)) - 1);

% Velocity and mach for slip walls
VN = sqrt(Uc(:,end).^2 + Vc(:,end).^2);   % North wall
VS = sqrt(Uc(:,1).^2   + Vc(:,1).^2);     % South wall

MN = VN ./ a_inf;
MS = VS ./ a_inf;

denominatorN = 1 + (gamma-1)/2 * MN.^2;
denominatorS = 1 + (gamma-1)/2 * MS.^2;

CpN = 2/(gamma*Mach^2) .* ((numerator ./ denominatorN).^(gamma/(gamma-1)) -
1);
CpS = 2/(gamma*Mach^2) .* ((numerator ./ denominatorS).^(gamma/(gamma-1)) -
1);

% X/c along the walls
XN = Xc(:,end) ./ max(Xc(:,end));
XS = Xc(:,1)   ./ max(Xc(:,1));

% sort for clean plotting
[XN, indN] = sort(XN);
MN = MN(indN);
CpN = CpN(indN);

[XS, indS] = sort(XS);
MS = MS(indS);
CpS = CpS(indS);

% Plots
figure()
plot(1:length(resi_hist),resi_hist,'LineWidth',2)
grid on
xlabel('No. of Iterations')
ylabel('Residual')
title('Grid Bumps Residual, Mach = 0.5, Omega = 1.82, Iter = 18426')
```

```
figure()
contourf(Xc',Yc',Vmag',30,'LineStyle','none')
hold on
colorbar
quiver(Xc',Yc',Uc',Vc','k')
%axis equal tight
title('Grid Bumps, Mach = 0.5, Omega = 1.82, Iter = 18426')

figure()
subplot(2,1,1)
plot(XN,MN,'LineWidth',1.5)
xlabel('X/c')
ylabel('Mach Number')
title('Grid Bumps, Mach Number along the North Wall')
grid on

subplot(2,1,2)
plot(XS,MS,'LineWidth',1.5)
xlabel('X/c')
ylabel('Mach Number')
title('Grid Bumps, Mach Number along the South Wall')
grid on

figure()
subplot(2,1,1)
plot(XN,CpN,'LineWidth',1.5)
xlabel('X/c')
ylabel('Pressure Coefficient')
title('Grid Bumps, Pressure coefficient along the North Wall')
grid on

subplot(2,1,2)
plot(XS,CpS,'LineWidth',1.5)
xlabel('X/c')
ylabel('Pressure Coefficient')
title('Grid Bumps, Pressure coefficient along the South Wall')
grid on
```

Mach 0.8 Grid Bumps

```
clear; clc; close all

% Reading the mesh file
mesh = readmatrix("grid.bumps.txt");

% Reading mesh dimensions
npx = mesh(1,1); % Dimension of mesh
npy = mesh(1,2); % Dimension of mesh
mesh(1,:) = []; % deleting first row

% Sorting mesh coordinates
ndimn = 2;
```

```
coord = zeros(ndimn,npx,npy); % pre-allocating coordinate matrix

num = 1; % counts the iteration
for j = 1:npy
    for i = 1:npx
        for k = 1:ndimn
            coord(k,i,j) = mesh(num,k);
        end
        num = num + 1; % updating variable
    end
end

% Subtracting one from number of nodes given in rows in and columns to
% obtain the number of cells in row and column
ncx = npx - 1; % number of cells in row
ncy = npy - 1; % number of cells in column

%Sanity grid plot check
x_val = squeeze(coord(1,:,:));
y_val = squeeze(coord(2,:,:));

figure(1)
plot(x_val,y_val,'k-');
hold on
plot(x_val',y_val','k-');
xlabel('x')
ylabel('y')
title('Grid Bumps Plot Check')

% Geometry Calculations Using geo_calc.m function
geoel = geo_calc(coord);

% Setting up inital conditions
% Using standard atmospheric conditions as instructed. Gamma for air is
% about 1.4 and R for air is 287 J/Kg*K. Temperature and pressure at
% freestream conidtions are given along with a Mach range.

T_inf = 300; % K
P_inf = 101325; % Pa
gamma = 1.4; % For air
R = 287; % J/Kg*K
Mach = 0.8;

% Freestream density
rho_inf = P_inf/(R*T_inf); % ideal gas law kg/m^3

% Iteration settings
omega = 0.05;
iter = 20000;
tol = 1e-4; % convergence criterion

% Speed of sound calculations
a_inf = sqrt(gamma*R*T_inf); % m/s
```

```
% Finding initial speed using freestream speed of sound and Mach numbers.u_inf
% is non-zero because I assuming the fluid freestream moves from left to
% right along the x-axis. v_inf is zero because I am assuming zero
% freestream velocity upwards or in the y-axis.
u_inf = a_inf*Mach; % m/s
v_inf = 0;

% Creating array for unknowns such as phi and rho and velocity. Adding
% 2 to the number of cells for ghost cell layer around all the walls.
var = 4; % 1 phi, 2 u-comp, 3 v-comp, 4 density
unkno = zeros(var, ncx+2, ncy+2);

% Initial guess for velocity potential based on freestream velocity
% components. The velocity potential for each cell will be at the cell
% centeroid. Would help this way because the array is cell centered in
% scrap code.
for k = 1:ncy
    for j = 1:ncx
        x_cent = geoel(1,j+1,k+1); % centroid for cells
        y_cent = geoel(2,j+1,k+1); % centroid for cells

        % Same initial guess for all real cells.
        unkno(1,j+1,k+1) = u_inf*x_cent + v_inf*y_cent;
    end
end

% Initializing density values
unkno(4,2:ncx+1,2:ncy+1) = rho_inf;

% Initializing residual history
resi_hist = zeros(iter,1);

% Iteration loop for convergence
for num = 1:iter

    % Ghost cell boundary conditions application
    unkno = boundary_cond(unkno,geoel,u_inf,v_inf);

    % Calculating velocity and density values from phi values for both real
    % and ghost cells
    unkno = DensityfromPhi(unkno,geoel,gamma,Mach,a_inf,rho_inf);

    % Calculating the residuals and the coefficients
    [resi,coeff] = resi_calc(unkno,geoel);

    % Updating phi using Gauss-Seidel Method
    [unkno,dphi,resi_n] = updatePhi(unkno,resi,coeff,omega);

    % Storing residual history
    resi_hist(num) = resi_n;

    % Convergence check
    if num == 1
        resi0 = resi_n;
```

```
    end

    resi_temp = resi_n/resi0;

    if resi_temp < tol
        resi_hist = resi_hist(1:num);
        break
    end
end

if num == iter
    resi_hist = resi_hist(1:iter);
end

% Extracting data for plotting
Xc = squeeze(geoel(1,2:ncx+1,2:ncy+1));
Yc = squeeze(geoel(2,2:ncx+1,2:ncy+1));
Uc = squeeze(unkno(2,2:ncx+1,2:ncy+1));
Vc = squeeze(unkno(3,2:ncx+1,2:ncy+1));
rhoc = squeeze(unkno(4,2:ncx+1,2:ncy+1));

% Velocity magnitude
Vmag = sqrt(Uc.^2 + Vc.^2);

% Local mach number
Mach_local = Vmag./a_inf;

% Pressure coefficient calculation
numerator = 1 + (gamma-1)/2 * Mach^2;
denomenator = 1 + (gamma-1)/2 * Mach_local.^2;
Cp = 2/(gamma*Mach^2).*((numerator./denomenator).^(gamma/(gamma-1)) - 1);

% Velocity and mach for slip walls
VN = sqrt(Uc(:,end).^2 + Vc(:,end).^2);   % North wall
VS = sqrt(Uc(:,1).^2   + Vc(:,1).^2);     % South wall

MN = VN ./ a_inf;
MS = VS ./ a_inf;

denominatorN = 1 + (gamma-1)/2 * MN.^2;
denominatorS = 1 + (gamma-1)/2 * MS.^2;

CpN = 2/(gamma*Mach^2) .* ((numerator ./ denominatorN).^(gamma/(gamma-1)) -
1);
CpS = 2/(gamma*Mach^2) .* ((numerator ./ denominatorS).^(gamma/(gamma-1)) -
1);

% X/c along the walls
XN = Xc(:,end) ./ max(Xc(:,end));
XS = Xc(:,1)   ./ max(Xc(:,1));

% sort for clean plotting
[XN, indN] = sort(XN);
MN = MN(indN);
```

```
CpN = CpN(indN);

[XS, indS] = sort(XS);
MS = MS(indS);
CpS = CpS(indS);

% Plots
figure()
plot(1:length(resi_hist),resi_hist,'LineWidth',2)
grid on
xlabel('No. of Iterations')
ylabel('Residual')
title('Grid Bumps Residual, Mach = 0.8, Omega = 0.05, Iter = 20000')

figure()
contourf(Xc',Yc',Vmag',30,'LineStyle','none')
hold on
colorbar
quiver(Xc',Yc',Uc',Vc','k')
%axis equal tight
title('Grid Bumps, Mach = 0.8, Omega = 0.05, Iter = 20000')

figure()
subplot(2,1,1)
plot(XN,MN,'LineWidth',1.5)
xlabel('X/c')
ylabel('Mach Number')
title('Grid Bumps, Mach Number along the North Wall')
grid on

subplot(2,1,2)
plot(XS,MS,'LineWidth',1.5)
xlabel('X/c')
ylabel('Mach Number')
title('Grid Bumps, Mach Number along the South Wall')
grid on

figure()
subplot(2,1,1)
plot(XN,CpN,'LineWidth',1.5)
xlabel('X/c')
ylabel('Pressure Coefficient')
title('Grid Bumps, Pressure coefficient along the North Wall')
grid on

subplot(2,1,2)
plot(XS,CpS,'LineWidth',1.5)
xlabel('X/c')
ylabel('Pressure Coefficient')
title('Grid Bumps, Pressure coefficient along the South Wall')
grid on
```

Functions

This function calculates basic geometry of cells within a mesh such as area, centroid, and finding normal vector. It was taken from HW_5 but 3rd and 4th corner definitions were changed so that it is counterclockwise as the number rises. It also will calculate the ghost cell geometry.

```
function geoel = geo_calc(coord)

[ndimn,npx,npy] = size(coord);

    ncx = npx - 1; % number of cells in row
    ncy = npy - 1; % number of cells in column

    ngeel = 11; % No. of geometric calculations
    geoel = zeros(ngeel,ncx+2,ncy+2);

    for j = 1:ncy
        for i = 1:ncx
            for k = 1:ndimn
                switch k
                    case 1
                    % X-coordinates
                    x1 = coord(k,i,j); % Bottom left corner
                    x2 = coord(k,i+1,j); % Bottom right corner
                    x3 = coord(k,i+1,j+1); % Top right corner
                    x4 = coord(k,i,j+1); % Top left corner

                    case 2
                    % Y-coordinates
                    y1 = coord(k,i,j); % Bottom left corner
                    y2 = coord(k,i+1,j); % Bottom right corner
                    y3 = coord(k,i+1,j+1); % Top right corner
                    y4 = coord(k,i,j+1); % Top left corner
                end
            end

            % Varibales to account for ghost cell layer.
            gi = i + 1;
            gj = j + 1;

            % Cell-centroid box average method
            xc = (x1 + x2 + x3 + x4)/4; % x-coordinate for centroid
            yc = (y1 + y2 + y3 + y4)/4; % y-coordinate for centroid

            % Cell area
            area = 0.5*((x3-x1)*(y4-y2)-(x4-x2)*(y3-y1));

            % Length weighted normal vector calculation
            % North
            % Swapping and negating to find the normal vector
            nNx = y4 - y3;
            nNy = -(x4 - x3);
```

```
            % East
            % Swapping and negating to find the normal vector
            nEx = y3 - y2;
            nEy = -(x3 - x2);

            % South
            % Swapping and negating to find the normal vector
            nSx = y2 - y1;
            nSy = -(x2 - x1);

            % West
            % Swapping and negating to find the normal vector
            nWx = y1 - y4;
            nWy = -(x1 - x4);

            % Storing values
            geoel(1,gi,gj) = xc; % Cell centroid x-axis
            geoel(2,gi,gj) = yc; % Cell centroid y-axis
            geoel(3,gi,gj) = area; % Cell area
            geoel(4,gi,gj) = nEx; % East length weighted normal vector
            geoel(5,gi,gj) = nEy; % East length weighted normal vector
            geoel(6,gi,gj) = nNx; % North length weighted normal vector
            geoel(7,gi,gj) = nNy; % North length weighted normal vector
            geoel(8,gi,gj) = nWx; % West length weighted normal vector
            geoel(9,gi,gj) = nWy; % West length weighted normal vector
            geoel(10,gi,gj) = nSx; % South length weighted normal vector
            geoel(11,gi,gj) = nSy; % South length weighted normal vector
        end
    end

    % South Boundary ghost cells

    for i = 1:ncx

        x1 = coord(1,i,1);
        x2 = coord(1,i+1,1);
        y1 = coord(2,i,1);
        y2 = coord(2,i+1,1);

        % Centroid for cells above south ghost cell boundary
        xc = geoel(1,i+1,2);
        yc = geoel(2,i+1,2);

        % External area weight normal vector to this face
        rnx = y2 - y1;
        rny = x1 - x2;

        geoel(6,i+1,1) = -rnx;
        geoel(7,i+1,1) = -rny;

        % Magnitude of the normal vector
        rnmag = sqrt(rnx^2 + rny^2);

        % Unit Nomral vector
```

```
        rux = rnx/rnmag;
        ruy = rny/rnmag;

        % Normal distance of vector (x1 - xc)
        distn = 2.0*((x1 - xc)*rux + (y1 - yc)*ruy);

        % Ghost cell center coordinates
        geoel(1,i+1,1) = xc + distn*rux;
        geoel(2,i+1,1) = yc + distn*ruy;

        % Ghost cell area
        geoel(3,i+1,1) = geoel(3,i+1,2);
    end

    % North Boundary ghost cells

    for i = 1:ncx

        x1 = coord(1,i,npy);
        x2 = coord(1,i+1,npy);
        y1 = coord(2,i,npy);
        y2 = coord(2,i+1,npy);

        % Centroid for cells below north ghost cell boundary
        xc = geoel(1,i+1,ncy+1);
        yc = geoel(2,i+1,ncy+1);

        % External area weight normal vector to this face
        rnx = y1 - y2;
        rny = x2 - x1;

        geoel(10,i+1,ncy+2) = -rnx;
        geoel(11,i+1,ncy+2) = -rny;

        % Magnitude of the normal vector
        rnmag = sqrt(rnx^2 + rny^2);

        % Unit Nomral vector
        rux = rnx/rnmag;
        ruy = rny/rnmag;

        % Normal distance of vector (x1 - xc)
        distn = 2.0*((x1 - xc)*rux + (y1 - yc)*ruy);

        % Ghost cell center coordinates
        geoel(1,i+1,ncy+2) = xc + distn*rux;
        geoel(2,i+1,ncy+2) = yc + distn*ruy;

        % Ghost cell area
        geoel(3,i+1,ncy+2) = geoel(3,i+1,ncy+1);
    end

    % West Boundary ghost cells
```

```
for j = 1:ncy

    x1 = coord(1,1,j);
    x2 = coord(1,1,j+1);
    y1 = coord(2,1,j);
    y2 = coord(2,1,j+1);

    % Centroid for cells beside west ghost cell boundary
    xc = geoel(1,2,j+1);
    yc = geoel(2,2,j+1);

    % External area weight normal vector to this face
    rnx = y1 - y2;
    rny = x2 - x1;

    geoel(4,1,j+1) = -rnx;
    geoel(5,1,j+1) = -rny;

    % Magnitude of the normal vector
    rnmag = sqrt(rnx^2 + rny^2);

    % Unit Nomral vector
    rux = rnx/rnmag;
    ruy = rny/rnmag;

    % Normal distance of vector (x1 - xc)
    distn = 2.0*((x1 - xc)*rux + (y1 - yc)*ruy);

    % Ghost cell center coordinates
    geoel(1,1,j+1) = xc + distn*rux;
    geoel(2,1,j+1) = yc + distn*ruy;

    % Ghost cell area
    geoel(3,1,j+1) = geoel(3,2,j+1);
end

% East Boundary ghost cells

for j = 1:ncy

    x1 = coord(1,npx,j);
    x2 = coord(1,npx,j+1);
    y1 = coord(2,npx,j);
    y2 = coord(2,npx,j+1);

    % Centroid for cells beside east ghost cell boundary
    xc = geoel(1,ncx+1,j+1);
    yc = geoel(2,ncx+1,j+1);

    % External area weight normal vector to this face
    rnx = y2 - y1;
    rny = x1 - x2;

    geoel(8,ncx+2,j+1) = -rnx;
```

```
        geoel(9,ncx+2,j+1) = -rny;

        % Magnitude of the normal vector
        rnmag = sqrt(rnx^2 + rny^2);

        % Unit Nomral vector
        rux = rnx/rnmag;
        ruy = rny/rnmag;

        % Normal distance of vector (x1 - xc)
        distn = 2.0*((x1 - xc)*rux + (y1 - yc)*ruy);

        % Ghost cell center coordinates
        geoel(1,ncx+2,j+1) = xc + distn*rux;
        geoel(2,ncx+2,j+1) = yc + distn*ruy;

        % Ghost cell area
        geoel(3,ncx+2,j+1) = geoel(3,ncx+1,j+1);
    end

    % Ghost cell corner geometry
    % South-west corner
    geoel(1,1,1) = 0.5*(geoel(1,1,2) + geoel(1,2,1));
    geoel(2,1,1) = 0.5*(geoel(2,1,2) + geoel(2,2,1));
    geoel(3,1,1) = geoel(3,2,2);

    % South-east corner
    geoel(1,ncx+2,1) = 0.5*(geoel(1,ncx+1,1) + geoel(1,ncx+2,2));
    geoel(2,ncx+2,1) = 0.5*(geoel(2,ncx+1,1) + geoel(2,ncx+2,2));
    geoel(3,ncx+2,1) = geoel(3,ncx+1,2);

    % North-east corner
    geoel(1,ncx+2,ncy+2) = 0.5*(geoel(1,ncx+1,ncy+2) + geoel(1,ncx+2,ncy+1));
    geoel(2,ncx+2,ncy+2) = 0.5*(geoel(2,ncx+1,ncy+2) + geoel(2,ncx+2,ncy+1));
    geoel(3,ncx+2,ncy+2) = geoel(3,ncx+1,ncy+1);

    % North-west corner
    geoel(1,1,ncy+2) = 0.5*(geoel(1,1,ncy+1) + geoel(1,2,ncy+2));
    geoel(2,1,ncy+2) = 0.5*(geoel(2,1,ncy+1) + geoel(2,2,ncy+2));
    geoel(3,1,ncy+2) = geoel(3,2,ncy+1);
end

% This function calculates the boundary condition for ghost cells.
% For this type of mesh, looking at the plot, the top and bottom walls
% for the ghost cells deform forming the boundary condition. However, it
% seems the boundary is solid for top and bottom with this deformation.
% Left and right walls would be far field boundary condition because it
% mimics freestreams conditions of the flow flowing from left to right.

function unkno = boundary_cond(unkno,geoel,u_inf,v_inf)

    % Getting number of cells including ghost cells
    [~,npx,npy] = size(geoel);
```

```
    % No. of real cells
    ncx = npx - 2;
    ncy = npy - 2;

    % North and South slip walls. Copying the interior values of these
    % walls to the ghost cells of the walls for the velocity potential.
    for i = 1:ncx

        % South Wall
        unkno(1,i+1,1) = unkno(1,i+1,2);

        % North Wall
        unkno(1,i+1,ncy+2) = unkno(1,i+1,ncy+1);
    end

    % West and east walls. For these walls, the value for the ghost cells
    % would be exactly the same as the freestream values. We multiply
    % velocity components with cell centroid coordinates.
    for j = 1:ncy

        % West Wall
        xcw = geoel(1,1,j+1);
        ycw = geoel(2,1,j+1);
        unkno(1,1,j+1) = u_inf*xcw + v_inf*ycw;

        % East Wall
        xce = geoel(1,ncx+2,j+1);
        yce = geoel(2,ncx+2,j+1);
        unkno(1,ncx+2,j+1) = u_inf*xce + v_inf*yce;
    end

    % Corner ghost cells only come into contact with other ghost cells
    % hence their value for phi can remain 0.
end

% This function uses the phi calculations for each cell center to calcuate
% the veolicty components and stores them in unkno array. Then it uses
% those components along with freestream flow vlaues to get the updated
% desnisty storing it in the unkno array as well. Ghost cell values for the
% all are just similar to the interior cells they share walls with.

function unkno = DensityfromPhi(unkno,geoel,gamma,Mach,a_inf,rho_inf)

    % Getting number of cells including ghost cells
    [~,npx,npy] = size(geoel);

    % No. of real cells
    ncx = npx - 2;
    ncy = npy - 2;

    % Velocity from Phi only for the real cells. Does not include ghost
    % cells. Ghost cell values can be copied from the real cells.
    for j = 1:ncy
        for i = 1:ncx
```

```
            % Taking area from previously calculated function
            area = geoel(3,i,j);

            % Face averaged Phi Values
            phiE = 0.5*(unkno(1,i+1,j+1) + unkno(1,i+2,j+1));
            phiN = 0.5*(unkno(1,i+1,j+1) + unkno(1,i+1,j+2));
            phiW = 0.5*(unkno(1,i+1,j+1) + unkno(1,i,j+1));
            phiS = 0.5*(unkno(1,i+1,j+1) + unkno(1,i+1,j));

            % Extracting outwards center normal vectors
            % East
            nEx = geoel(4,i+1,j+1);
            nEy = geoel(5,i+1,j+1);

            % North
            nNx = geoel(6,i+1,j+1);
            nNy = geoel(7,i+1,j+1);

            % West
            nWx = geoel(8,i+1,j+1);
            nWy = geoel(9,i+1,j+1);

            % South
            nSx = geoel(10,i+1,j+1);
            nSy = geoel(11,i+1,j+1);

            % Velcoity components u and v by multiplying phi values with
            % vector normals and dividing by the area
            % u
            unkno(2,i+1,j+1) = (phiE*nEx + phiN*nNx + phiW*nWx + phiS*nSx)/
area;

            %v
            unkno(3,i+1,j+1) = (phiE*nEy + phiN*nNy + phiW*nWy + phiS*nSy)/
area;
        end
    end

    % Density from velocity using the provided formula. This is
    % also only for the real cells.
    for j = 1:ncy
        for i = 1:ncx

            u = unkno(2,i+1,j+1);
            v = unkno(3,i+1,j+1);

            % Density Calculation
            rho = rho_inf*(1 + (gamma - 1)/2 * (Mach^2 - (u^2 + v^2)/
a_inf^2))^(1/(gamma - 1));

            % Putting it in the unknown array
            unkno(4,i+1,j+1) = rho;
        end
```

```
    end

    % Filling in the value of the ghost cells
    % North and south ghost cell boundary
    for i = 1:ncx

        % North
        unkno(2,i+1,npy) = unkno(2,i+1,npy-1);
        unkno(3,i+1,npy) = unkno(3,i+1,npy-1);
        unkno(4,i+1,npy) = unkno(4,i+1,npy-1);

        % South
        unkno(2,i+1,1) = unkno(2,i+1,2);
        unkno(3,i+1,1) = unkno(3,i+1,2);
        unkno(4,i+1,1) = unkno(4,i+1,2);
    end

    % East and west ghost cell boundary
    for j = 1:ncy

        % East
        unkno(2,npx,j+1) = unkno(2,npx-1,j+1);
        unkno(3,npx,j+1) = unkno(3,npx-1,j+1);
        unkno(4,npx,j+1) = unkno(4,npx-1,j+1);

        % West
        unkno(2,1,j+1) = unkno(2,2,j+1);
        unkno(3,1,j+1) = unkno(3,2,j+1);
        unkno(4,1,j+1) = unkno(4,2,j+1);
    end

    % Corner cells the velocity and other flow properties remain zero as
    % the surounding cells are ghost cells which do not depend on the
    % corner cell for their property calculations
end
% This function calculates the residuals and the flux coefficients for the
% each wall of the cell along with its center. Each face contribution is
% calculated seperately by taking average of density, cell volume, and square
% face area. This is used to calculate the flux and the residuals. The code
% is based on the provided scrap code.

function [resi,coeff] = resi_calc(unkno,geoel)

    % Getting number of cells including ghost cells
    [~,npx,npy] = size(geoel);

    % No. of real cells
    ncx = npx - 2;
    ncy = npy - 2;

    % Initialization of residual and coefficient matrices
    resi = zeros(ncx+2,ncy+2);
    coeff = zeros(5, ncx+2, ncy+2);
```

```
    for j = 2:ncy+1
        for i = 2:ncx+1

            % Density averages for cell face
            rhoE = 0.5*(unkno(4,i,j) + unkno(4,i+1,j));
            rhoN = 0.5*(unkno(4,i,j) + unkno(4,i,j+1));
            rhoS = 0.5*(unkno(4,i,j) + unkno(4,i,j-1));
            rhoW = 0.5*(unkno(4,i,j) + unkno(4,i-1,j));

            % Average cell volume
            volE = 0.5*(geoel(3,i,j) + geoel(3,i+1,j));
            volN = 0.5*(geoel(3,i,j) + geoel(3,i,j+1));
            volS = 0.5*(geoel(3,i,j) + geoel(3,i,j-1));
            volW = 0.5*(geoel(3,i,j) + geoel(3,i-1,j));

            % Squared face area
            area2E = geoel(4,i,j)^2 + geoel(5,i,j)^2;
            area2N = geoel(6,i,j)^2 + geoel(7,i,j)^2;
            area2S = geoel(10,i,j)^2 + geoel(11,i,j)^2;
            area2W = geoel(8,i,j)^2 + geoel(9,i,j)^2;

            % Coefficient calculations
            coeffE = rhoE*area2E/volE;
            coeffN = rhoN*area2N/volN;
            coeffS = rhoS*area2S/volS;
            coeffW = rhoW*area2W/volW;

            % Flux calculation
            FE = coeffE*(unkno(1,i+1,j) - unkno(1,i,j));
            FN = coeffN*(unkno(1,i,j+1) - unkno(1,i,j));
            FS = coeffS*(unkno(1,i,j-1) - unkno(1,i,j));
            FW = coeffW*(unkno(1,i-1,j) - unkno(1,i,j));

            % Residual calculation
            resi(i,j) = FE + FN + FS + FW;

            % Coeff matrix assignment
            coeff(1,i,j) = coeffE;
            coeff(2,i,j) = coeffN;
            coeff(3,i,j) = coeffW;
            coeff(4,i,j) = coeffS;
            coeff(5,i,j) = -(coeffE + coeffN + coeffW + coeffS); % centeral
        end
    end
end
% This function updates phi using Gauss Seidel method.

function [unkno,dphi,resi_n] = updatePhi(unkno,resi,coeff,omega)

    % Getting number of cells including ghost cells
    [~,npx,npy] = size(unkno);

    % No. of real cells
    ncx = npx - 2;
```

```
    ncy = npy - 2;

    % Intializing delta phi
    dphi = zeros(npx,npy);

    % Gauss-Seidel method
    for j = 2:ncy+1
        for i = 2:ncx+1

            coeffE = coeff(1,i,j); % East
            coeffN = coeff(2,i,j); % north
            coeffW = coeff(3,i,j); % West
            coeffS = coeff(4,i,j); % south
            coeffC = coeff(5,i,j); % center

            % temp storage
            dphitemp = dphi(i,j);

            % Gauss-Seidel calculation
            dphigs = (-resi(i,j) - coeffW*dphi(i-1,j) - coeffS*dphi(i,j-1) ...
                - coeffN*dphi(i,j+1) - coeffE*dphi(i+1,j))/coeffC;

            % SOR step
            dphi(i,j) = dphitemp + omega*(dphigs - dphitemp);
        end
    end

    for j = 2:ncy+1
        for i = 2:ncx+1
            unkno(1,i,j) = unkno(1,i,j) + dphi(i,j);
        end
    end

    % Residual normal
    resi_n = norm(resi(2:ncx+1,2:ncy+1),2);

end
```

Published with MATLAB® R2024b

Table of Contents

Mach 0.01 Half Cylinder

```
clear; clc; close all

% Reading the mesh file
mesh = readmatrix("half_cylinder.txt");

% Reading mesh dimensions
npx = mesh(1,1); % Dimension of mesh
npy = mesh(1,2); % Dimension of mesh
mesh(1,:) = []; % deleting first row

% Sorting mesh coordinates
ndimn = 2;
coord = zeros(ndimn,npx,npy); % pre-allocating coordinate matrix

num = 1; % counts the iteration
for j = 1:npy
    for i = 1:npx
        for k = 1:ndimn
            coord(k,i,j) = mesh(num,k);
        end
        num = num + 1; % updating variable
    end
end

% Subtracting one from number of nodes given in rows in and columns to
% obtain the number of cells in row and column
ncx = npx - 1; % number of cells in row
ncy = npy - 1; % number of cells in column

%Sanity grid plot check
x_val = squeeze(coord(1,:,:));
y_val = squeeze(coord(2,:,:));

figure(1)
plot(x_val,y_val,'k-');
hold on
plot(x_val',y_val','k-');
xlabel('x')
ylabel('y')
title('Half Cylinder Plot Check')
axis equal tight
```

```
% Geometry Calculations Using geo_calc.m function
geoel = geo_calc(coord);

% Setting up inital conditions
% Using standard atmospheric conditions as instructed. Gamma for air is
% about 1.4 and R for air is 287 J/Kg*K. Temperature and pressure at
% freestream conidtions are given along with a Mach range.

T_inf = 300; % K
P_inf = 101325; % Pa
gamma = 1.4; % For air
R = 287; % J/Kg*K
Mach = 0.01;

% Freestream density
rho_inf = P_inf/(R*T_inf); % ideal gas law kg/m^3

% Iteration settings
omega = 1;
iter = 10000;
tol = 1e-4; % convergence criterion

% Speed of sound calculations
a_inf = sqrt(gamma*R*T_inf); % m/s

% Finding initial speed using freestream speed of sound and Mach numbers.u_inf
% is non-zero because I assuming the fluid freestream moves from left to
% right along the x-axis. v_inf is zero because I am assuming zero
% freestream velocity upwards or in the y-axis.
u_inf = a_inf*Mach; % m/s
v_inf = 0;

% Creating array for unknowns such as phi and rho and velocity. Adding
% 2 to the number of cells for ghost cell layer around all the walls.
var = 4; % 1 phi, 2 u-comp, 3 v-comp, 4 density
unkno = zeros(var, ncx+2, ncy+2);

% Initial guess for velocity potential based on freestream velocity
% components. The velocity potential for each cell will be at the cell
% centeroid. Would help this way because the array is cell centered in
% scrap code.
for k = 1:ncy
    for j = 1:ncx
        x_cent = geoel(1,j+1,k+1); % centroid for cells
        y_cent = geoel(2,j+1,k+1); % centroid for cells

        % Same initial guess for all real cells.
        unkno(1,j+1,k+1) = u_inf*x_cent + v_inf*y_cent;
    end
end

% Initializing density values
unkno(4,2:ncx+1,2:ncy+1) = rho_inf;
```

```
% Initializing residual history
resi_hist = zeros(iter,1);

% Iteration loop for convergence
for num = 1:iter

    % Ghost cell boundary conditions application
    unkno = boundary_cond_cylinder(unkno,geoel,u_inf,v_inf);

    % Calculating velocity and density values from phi values for both real
    % and ghost cells
    unkno = DensityfromPhi(unkno,geoel,gamma,Mach,a_inf,rho_inf);

    % Calculating the residuals and the coefficients
    [resi,coeff] = resi_calc(unkno,geoel);

    % Updating phi using Gauss-Seidel Method
    [unkno,dphi,resi_n] = updatePhi(unkno,resi,coeff,omega);

    % Storing residual history
    resi_hist(num) = resi_n;

    % Convergence check
    if num == 1
        resi0 = resi_n;
    end

    resi_temp = resi_n/resi0;

    if resi_temp < tol
        resi_hist = resi_hist(1:num);
        break
    end
end

if num == iter
    resi_hist = resi_hist(1:iter);
end

% Extracting data for plotting
Xc = squeeze(geoel(1,2:ncx+1,2:ncy+1));
Yc = squeeze(geoel(2,2:ncx+1,2:ncy+1));
Uc = squeeze(unkno(2,2:ncx+1,2:ncy+1));
Vc = squeeze(unkno(3,2:ncx+1,2:ncy+1));
rhoc = squeeze(unkno(4,2:ncx+1,2:ncy+1));

% Velocity magnitude
Vmag = sqrt(Uc.^2 + Vc.^2);

% Local mach number
Mach_local = Vmag./a_inf;

Vwall = sqrt(Uc(:,1).^2 + Vc(:,1).^2);
Mwall = Vwall/a_inf;
```

```
% Pressure coefficient calculation
numerator = 1 + (gamma-1)/2 * Mach^2;
denomenator = 1 + (gamma-1)/2 * Mwall.^2;
Cpwall = 2/(gamma*Mach^2).*((numerator./denomenator).^(gamma/(gamma-1)) - 1);

% X/c along the wall
Xwall = Xc(:,1);
Xwall = Xwall ./ max(abs(Xwall));

% Sort for clean plotting
[Xwall, indW] = sort(Xwall);
Mwall = Mwall(indW);
Cpwall = Cpwall(indW);

% Plots
figure()
plot(1:length(resi_hist),resi_hist,'LineWidth',2)
grid on
xlabel('No. of Iterations')
ylabel('Residual')
title('Half Cylinder Residual, Mach = 0.01, Omega = 1, Iter = 3043')

figure()
contourf(Xc',Yc',Vmag',30,'LineStyle','none')
hold on
colorbar
quiver(Xc',Yc',Uc',Vc','k')
axis equal tight
title('Half Cylinder, Mach = 0.01, Omega = 1, Iter = 3043')

figure()
plot(Xwall,Mwall,'LineWidth',1.5)
xlabel('X/c')
ylabel('Mach Number')
title('Half Cylinder, Mach Number along the Body Wall')
grid on

figure()
plot(Xwall,Cpwall,'LineWidth',1.5)
xlabel('X/c')
ylabel('Pressure Coefficient')
title('Half Cylinder, Pressure coefficient along the Body Wall')
grid on
```

Mach 0.5 Half Cylinder

```
clear; clc; close all

% Reading the mesh file
mesh = readmatrix("half_cylinder.txt");

% Reading mesh dimensions
```

```
npx = mesh(1,1); % Dimension of mesh
npy = mesh(1,2); % Dimension of mesh
mesh(1,:) = []; % deleting first row

% Sorting mesh coordinates
ndimn = 2;
coord = zeros(ndimn,npx,npy); % pre-allocating coordinate matrix

num = 1; % counts the iteration
for j = 1:npy
    for i = 1:npx
        for k = 1:ndimn
            coord(k,i,j) = mesh(num,k);
        end
        num = num + 1; % updating variable
    end
end

% Subtracting one from number of nodes given in rows in and columns to
% obtain the number of cells in row and column
ncx = npx - 1; % number of cells in row
ncy = npy - 1; % number of cells in column

%Sanity grid plot check
x_val = squeeze(coord(1,:,:));
y_val = squeeze(coord(2,:,:));

figure(1)
plot(x_val,y_val,'k-');
hold on
plot(x_val',y_val','k-');
xlabel('x')
ylabel('y')
title('Half Cylinder Plot Check')
axis equal tight

% Geometry Calculations Using geo_calc.m function
geoel = geo_calc(coord);

% Setting up inital conditions
% Using standard atmospheric conditions as instructed. Gamma for air is
% about 1.4 and R for air is 287 J/Kg*K. Temperature and pressure at
% freestream conidtions are given along with a Mach range.

T_inf = 300; % K
P_inf = 101325; % Pa
gamma = 1.4; % For air
R = 287; % J/Kg*K
Mach = 0.5;

% Freestream density
rho_inf = P_inf/(R*T_inf); % ideal gas law kg/m^3

% Iteration settings
```

```
omega = 0.03;
iter = 18000;
tol = 1e-4; % convergence criterion

% Speed of sound calculations
a_inf = sqrt(gamma*R*T_inf); % m/s

% Finding initial speed using freestream speed of sound and Mach numbers.u_inf
% is non-zero because I assuming the fluid freestream moves from left to
% right along the x-axis. v_inf is zero because I am assuming zero
% freestream velocity upwards or in the y-axis.
u_inf = a_inf*Mach; % m/s
v_inf = 0;

% Creating array for unknowns such as phi and rho and velocity. Adding
% 2 to the number of cells for ghost cell layer around all the walls.
var = 4; % 1 phi, 2 u-comp, 3 v-comp, 4 density
unkno = zeros(var, ncx+2, ncy+2);

% Initial guess for velocity potential based on freestream velocity
% components. The velocity potential for each cell will be at the cell
% centeroid. Would help this way because the array is cell centered in
% scrap code.
for k = 1:ncy
    for j = 1:ncx
        x_cent = geoel(1,j+1,k+1); % centroid for cells
        y_cent = geoel(2,j+1,k+1); % centroid for cells

        % Same initial guess for all real cells.
        unkno(1,j+1,k+1) = u_inf*x_cent + v_inf*y_cent;
    end
end

% Initializing density values
unkno(4,2:ncx+1,2:ncy+1) = rho_inf;

% Initializing residual history
resi_hist = zeros(iter,1);

% Iteration loop for convergence
for num = 1:iter

    % Ghost cell boundary conditions application
    unkno = boundary_cond_cylinder(unkno,geoel,u_inf,v_inf);

    % Calculating velocity and density values from phi values for both real
    % and ghost cells
    unkno = DensityfromPhi(unkno,geoel,gamma,Mach,a_inf,rho_inf);

    % Calculating the residuals and the coefficients
    [resi,coeff] = resi_calc(unkno,geoel);

    % Updating phi using Gauss-Seidel Method
    [unkno,dphi,resi_n] = updatePhi(unkno,resi,coeff,omega);
```

```
    % Storing residual history
    resi_hist(num) = resi_n;

    % Convergence check
    if num == 1
        resi0 = resi_n;
    end

    resi_temp = resi_n/resi0;

    if resi_temp < tol
        resi_hist = resi_hist(1:num);
        break
    end
end

if num == iter
    resi_hist = resi_hist(1:iter);
end

% Extracting data for plotting
Xc = squeeze(geoel(1,2:ncx+1,2:ncy+1));
Yc = squeeze(geoel(2,2:ncx+1,2:ncy+1));
Uc = squeeze(unkno(2,2:ncx+1,2:ncy+1));
Vc = squeeze(unkno(3,2:ncx+1,2:ncy+1));
rhoc = squeeze(unkno(4,2:ncx+1,2:ncy+1));

% Velocity magnitude
Vmag = sqrt(Uc.^2 + Vc.^2);

% Local mach number
Mach_local = Vmag./a_inf;

Vwall = sqrt(Uc(:,1).^2 + Vc(:,1).^2);
Mwall = Vwall/a_inf;

% Pressure coefficient calculation
numerator = 1 + (gamma-1)/2 * Mach^2;
denomenator = 1 + (gamma-1)/2 * Mwall.^2;
Cpwall = 2/(gamma*Mach^2).*((numerator./denomenator).^(gamma/(gamma-1)) - 1);

% X/c along the wall
Xwall = Xc(:,1);
Xwall = Xwall ./ max(abs(Xwall));

% Sort for clean plotting
[Xwall, indW] = sort(Xwall);
Mwall = Mwall(indW);
Cpwall = Cpwall(indW);

% Plots
figure()
plot(1:length(resi_hist),resi_hist,'LineWidth',2)
```

```
grid on
xlabel('No. of Iterations')
ylabel('Residual')
title('Half Cylinder Residual, Mach = 0.5, Omega = 0.03, Iter = 18000')

figure()
contourf(Xc',Yc',Vmag',30,'LineStyle','none')
hold on
colorbar
quiver(Xc',Yc',Uc',Vc','k')
axis equal tight
title('Half Cylinder, Mach = 0.5, Omega = 0.03, Iter = 18000')

figure()
plot(Xwall,Mwall,'LineWidth',1.5)
xlabel('X/c')
ylabel('Mach Number')
title('Half Cylinder, Mach Number along the Body Wall')
grid on

figure()
plot(Xwall,Cpwall,'LineWidth',1.5)
xlabel('X/c')
ylabel('Pressure Coefficient')
title('Half Cylinder, Pressure coefficient along the Body Wall')
grid on
```

Mach 0.8 Half Cylinder

```
clear; clc; close all

% Reading the mesh file
mesh = readmatrix("half_cylinder.txt");

% Reading mesh dimensions
npx = mesh(1,1); % Dimension of mesh
npy = mesh(1,2); % Dimension of mesh
mesh(1,:) = []; % deleting first row

% Sorting mesh coordinates
ndimn = 2;
coord = zeros(ndimn,npx,npy); % pre-allocating coordinate matrix

num = 1; % counts the iteration
for j = 1:npy
    for i = 1:npx
        for k = 1:ndimn
            coord(k,i,j) = mesh(num,k);
        end
        num = num + 1; % updating variable
    end
end
```

```
% Subtracting one from number of nodes given in rows in and columns to
% obtain the number of cells in row and column
ncx = npx - 1; % number of cells in row
ncy = npy - 1; % number of cells in column

%Sanity grid plot check
x_val = squeeze(coord(1,:,:));
y_val = squeeze(coord(2,:,:));

figure(1)
plot(x_val,y_val,'k-');
hold on
plot(x_val',y_val','k-');
xlabel('x')
ylabel('y')
title('Half Cylinder Plot Check')
axis equal tight

% Geometry Calculations Using geo_calc.m function
geoel = geo_calc(coord);

% Setting up inital conditions
% Using standard atmospheric conditions as instructed. Gamma for air is
% about 1.4 and R for air is 287 J/Kg*K. Temperature and pressure at
% freestream conidtions are given along with a Mach range.

T_inf = 300; % K
P_inf = 101325; % Pa
gamma = 1.4; % For air
R = 287; % J/Kg*K
Mach = 0.8;

% Freestream density
rho_inf = P_inf/(R*T_inf); % ideal gas law kg/m^3

% Iteration settings
omega = 0.008;
iter = 16000;
tol = 1e-4; % convergence criterion

% Speed of sound calculations
a_inf = sqrt(gamma*R*T_inf); % m/s

% Finding initial speed using freestream speed of sound and Mach numbers.u_inf
% is non-zero because I assuming the fluid freestream moves from left to
% right along the x-axis. v_inf is zero because I am assuming zero
% freestream velocity upwards or in the y-axis.
u_inf = a_inf*Mach; % m/s
v_inf = 0;

% Creating array for unknowns such as phi and rho and velocity. Adding
% 2 to the number of cells for ghost cell layer around all the walls.
var = 4; % 1 phi, 2 u-comp, 3 v-comp, 4 density
unkno = zeros(var, ncx+2, ncy+2);
```

```
% Initial guess for velocity potential based on freestream velocity
% components. The velocity potential for each cell will be at the cell
% centeroid. Would help this way because the array is cell centered in
% scrap code.
for k = 1:ncy
    for j = 1:ncx
        x_cent = geoel(1,j+1,k+1); % centroid for cells
        y_cent = geoel(2,j+1,k+1); % centroid for cells

        % Same initial guess for all real cells.
        unkno(1,j+1,k+1) = u_inf*x_cent + v_inf*y_cent;
    end
end

% Initializing density values
unkno(4,2:ncx+1,2:ncy+1) = rho_inf;

% Initializing residual history
resi_hist = zeros(iter,1);

% Iteration loop for convergence
for num = 1:iter

    % Ghost cell boundary conditions application
    unkno = boundary_cond_cylinder(unkno,geoel,u_inf,v_inf);

    % Calculating velocity and density values from phi values for both real
    % and ghost cells
    unkno = DensityfromPhi(unkno,geoel,gamma,Mach,a_inf,rho_inf);

    % Calculating the residuals and the coefficients
    [resi,coeff] = resi_calc(unkno,geoel);

    % Updating phi using Gauss-Seidel Method
    [unkno,dphi,resi_n] = updatePhi(unkno,resi,coeff,omega);

    % Storing residual history
    resi_hist(num) = resi_n;

    % Convergence check
    if num == 1
        resi0 = resi_n;
    end

    resi_temp = resi_n/resi0;

    if resi_temp < tol
        resi_hist = resi_hist(1:num);
        break
    end
end

if num == iter
```

```
    resi_hist = resi_hist(1:iter);
end

% Extracting data for plotting
Xc = squeeze(geoel(1,2:ncx+1,2:ncy+1));
Yc = squeeze(geoel(2,2:ncx+1,2:ncy+1));
Uc = squeeze(unkno(2,2:ncx+1,2:ncy+1));
Vc = squeeze(unkno(3,2:ncx+1,2:ncy+1));
rhoc = squeeze(unkno(4,2:ncx+1,2:ncy+1));

% Velocity magnitude
Vmag = sqrt(Uc.^2 + Vc.^2);

% Local mach number
Mach_local = Vmag./a_inf;

Vwall = sqrt(Uc(:,1).^2 + Vc(:,1).^2);
Mwall = Vwall/a_inf;

% Pressure coefficient calculation
numerator = 1 + (gamma-1)/2 * Mach^2;
denomenator = 1 + (gamma-1)/2 * Mwall.^2;
Cpwall = 2/(gamma*Mach^2).*((numerator./denomenator).^(gamma/(gamma-1)) - 1);

% X/c along the wall
Xwall = Xc(:,1);
Xwall = Xwall ./ max(abs(Xwall));

% Sort for clean plotting
[Xwall, indW] = sort(Xwall);
Mwall = Mwall(indW);
Cpwall = Cpwall(indW);

% Plots
figure()
plot(1:length(resi_hist),resi_hist,'LineWidth',2)
grid on
xlabel('No. of Iterations')
ylabel('Residual')
title('Half Cylinder Residual, Mach = 0.8, Omega = 0.008, Iter = 16000')

figure()
contourf(Xc',Yc',Vmag',30,'LineStyle','none')
hold on
colorbar
quiver(Xc',Yc',Uc',Vc','k')
axis equal tight
title('Half Cylinder, Mach = 0.8, Omega = 0.008, Iter = 16000')

figure()
plot(Xwall,Mwall,'LineWidth',1.5)
xlabel('X/c')
ylabel('Mach Number')
title('Half Cylinder, Mach Number along the Body Wall')
```

```
grid on

figure()
plot(Xwall,Cpwall,'LineWidth',1.5)
xlabel('X/c')
ylabel('Pressure Coefficient')
title('Half Cylinder, Pressure coefficient along the Body Wall')
grid on
```

Functions

This function calculates basic geometry of cells within a mesh such as area, centroid, and finding normal vector. It was taken from HW_5 but 3rd and 4th corner definitions were changed so that it is counterclockwise as the number rises. It also will calculate the ghost cell geometry.

```
function geoel = geo_calc(coord)

[ndimn,npx,npy] = size(coord);

    ncx = npx - 1; % number of cells in row
    ncy = npy - 1; % number of cells in column

    ngeel = 11; % No. of geometric calculations
    geoel = zeros(ngeel,ncx+2,ncy+2);

    for j = 1:ncy
        for i = 1:ncx
            for k = 1:ndimn
                switch k
                    case 1
                    % X-coordinates
                    x1 = coord(k,i,j); % Bottom left corner
                    x2 = coord(k,i+1,j); % Bottom right corner
                    x3 = coord(k,i+1,j+1); % Top right corner
                    x4 = coord(k,i,j+1); % Top left corner

                    case 2
                    % Y-coordinates
                    y1 = coord(k,i,j); % Bottom left corner
                    y2 = coord(k,i+1,j); % Bottom right corner
                    y3 = coord(k,i+1,j+1); % Top right corner
                    y4 = coord(k,i,j+1); % Top left corner
                end
            end

            % Varibales to account for ghost cell layer.
            gi = i + 1;
            gj = j + 1;

            % Cell-centroid box average method
            xc = (x1 + x2 + x3 + x4)/4; % x-coordinate for centroid
            yc = (y1 + y2 + y3 + y4)/4; % y-coordinate for centroid
```

```
            % Cell area
            area = 0.5*((x3-x1)*(y4-y2)-(x4-x2)*(y3-y1));

            % Length weighted normal vector calculation
            % North
            % Swapping and negating to find the normal vector
            nNx = y4 - y3;
            nNy = -(x4 - x3);

            % East
            % Swapping and negating to find the normal vector
            nEx = y3 - y2;
            nEy = -(x3 - x2);

            % South
            % Swapping and negating to find the normal vector
            nSx = y2 - y1;
            nSy = -(x2 - x1);

            % West
            % Swapping and negating to find the normal vector
            nWx = y1 - y4;
            nWy = -(x1 - x4);

            % Storing values
            geoel(1,gi,gj) = xc; % Cell centroid x-axis
            geoel(2,gi,gj) = yc; % Cell centroid y-axis
            geoel(3,gi,gj) = area; % Cell area
            geoel(4,gi,gj) = nEx; % East length weighted normal vector
            geoel(5,gi,gj) = nEy; % East length weighted normal vector
            geoel(6,gi,gj) = nNx; % North length weighted normal vector
            geoel(7,gi,gj) = nNy; % North length weighted normal vector
            geoel(8,gi,gj) = nWx; % West length weighted normal vector
            geoel(9,gi,gj) = nWy; % West length weighted normal vector
            geoel(10,gi,gj) = nSx; % South length weighted normal vector
            geoel(11,gi,gj) = nSy; % South length weighted normal vector
        end
    end

    % South Boundary ghost cells

    for i = 1:ncx

        x1 = coord(1,i,1);
        x2 = coord(1,i+1,1);
        y1 = coord(2,i,1);
        y2 = coord(2,i+1,1);

        % Centroid for cells above south ghost cell boundary
        xc = geoel(1,i+1,2);
        yc = geoel(2,i+1,2);

        % External area weight normal vector to this face
        rnx = y2 - y1;
```

```
        rny = x1 - x2;

        geoel(6,i+1,1) = -rnx;
        geoel(7,i+1,1) = -rny;

        % Magnitude of the normal vector
        rnmag = sqrt(rnx^2 + rny^2);

        % Unit Nomral vector
        rux = rnx/rnmag;
        ruy = rny/rnmag;

        % Normal distance of vector (x1 - xc)
        distn = 2.0*((x1 - xc)*rux + (y1 - yc)*ruy);

        % Ghost cell center coordinates
        geoel(1,i+1,1) = xc + distn*rux;
        geoel(2,i+1,1) = yc + distn*ruy;

        % Ghost cell area
        geoel(3,i+1,1) = geoel(3,i+1,2);
    end

    % North Boundary ghost cells

    for i = 1:ncx

        x1 = coord(1,i,npy);
        x2 = coord(1,i+1,npy);
        y1 = coord(2,i,npy);
        y2 = coord(2,i+1,npy);

        % Centroid for cells below north ghost cell boundary
        xc = geoel(1,i+1,ncy+1);
        yc = geoel(2,i+1,ncy+1);

        % External area weight normal vector to this face
        rnx = y1 - y2;
        rny = x2 - x1;

        geoel(10,i+1,ncy+2) = -rnx;
        geoel(11,i+1,ncy+2) = -rny;

        % Magnitude of the normal vector
        rnmag = sqrt(rnx^2 + rny^2);

        % Unit Nomral vector
        rux = rnx/rnmag;
        ruy = rny/rnmag;

        % Normal distance of vector (x1 - xc)
        distn = 2.0*((x1 - xc)*rux + (y1 - yc)*ruy);

        % Ghost cell center coordinates
```

```
    geoel(1,i+1,ncy+2) = xc + distn*rux;
    geoel(2,i+1,ncy+2) = yc + distn*ruy;

    % Ghost cell area
    geoel(3,i+1,ncy+2) = geoel(3,i+1,ncy+1);
end

% West Boundary ghost cells

for j = 1:ncy

    x1 = coord(1,1,j);
    x2 = coord(1,1,j+1);
    y1 = coord(2,1,j);
    y2 = coord(2,1,j+1);

    % Centroid for cells beside west ghost cell boundary
    xc = geoel(1,2,j+1);
    yc = geoel(2,2,j+1);

    % External area weight normal vector to this face
    rnx = y1 - y2;
    rny = x2 - x1;

    geoel(4,1,j+1) = -rnx;
    geoel(5,1,j+1) = -rny;

    % Magnitude of the normal vector
    rnmag = sqrt(rnx^2 + rny^2);

    % Unit Nomral vector
    rux = rnx/rnmag;
    ruy = rny/rnmag;

    % Normal distance of vector (x1 - xc)
    distn = 2.0*((x1 - xc)*rux + (y1 - yc)*ruy);

    % Ghost cell center coordinates
    geoel(1,1,j+1) = xc + distn*rux;
    geoel(2,1,j+1) = yc + distn*ruy;

    % Ghost cell area
    geoel(3,1,j+1) = geoel(3,2,j+1);
end

% East Boundary ghost cells

for j = 1:ncy

    x1 = coord(1,npx,j);
    x2 = coord(1,npx,j+1);
    y1 = coord(2,npx,j);
    y2 = coord(2,npx,j+1);
```

```
        % Centroid for cells beside east ghost cell boundary
        xc = geoel(1,ncx+1,j+1);
        yc = geoel(2,ncx+1,j+1);

        % External area weight normal vector to this face
        rnx = y2 - y1;
        rny = x1 - x2;

        geoel(8,ncx+2,j+1) = -rnx;
        geoel(9,ncx+2,j+1) = -rny;

        % Magnitude of the normal vector
        rnmag = sqrt(rnx^2 + rny^2);

        % Unit Nomral vector
        rux = rnx/rnmag;
        ruy = rny/rnmag;

        % Normal distance of vector (x1 - xc)
        distn = 2.0*((x1 - xc)*rux + (y1 - yc)*ruy);

        % Ghost cell center coordinates
        geoel(1,ncx+2,j+1) = xc + distn*rux;
        geoel(2,ncx+2,j+1) = yc + distn*ruy;

        % Ghost cell area
        geoel(3,ncx+2,j+1) = geoel(3,ncx+1,j+1);
    end

    % Ghost cell corner geometry
    % South-west corner
    geoel(1,1,1) = 0.5*(geoel(1,1,2) + geoel(1,2,1));
    geoel(2,1,1) = 0.5*(geoel(2,1,2) + geoel(2,2,1));
    geoel(3,1,1) = geoel(3,2,2);

    % South-east corner
    geoel(1,ncx+2,1) = 0.5*(geoel(1,ncx+1,1) + geoel(1,ncx+2,2));
    geoel(2,ncx+2,1) = 0.5*(geoel(2,ncx+1,1) + geoel(2,ncx+2,2));
    geoel(3,ncx+2,1) = geoel(3,ncx+1,2);

    % North-east corner
    geoel(1,ncx+2,ncy+2) = 0.5*(geoel(1,ncx+1,ncy+2) + geoel(1,ncx+2,ncy+1));
    geoel(2,ncx+2,ncy+2) = 0.5*(geoel(2,ncx+1,ncy+2) + geoel(2,ncx+2,ncy+1));
    geoel(3,ncx+2,ncy+2) = geoel(3,ncx+1,ncy+1);

    % North-west corner
    geoel(1,1,ncy+2) = 0.5*(geoel(1,1,ncy+1) + geoel(1,2,ncy+2));
    geoel(2,1,ncy+2) = 0.5*(geoel(2,1,ncy+1) + geoel(2,2,ncy+2));
    geoel(3,1,ncy+2) = geoel(3,2,ncy+1);
end

function unkno = boundary_cond_cylinder(unkno,geoel,u_inf,v_inf)

    % Getting number of cells including ghost cells
```

```
    [~,npx,npy] = size(geoel);

    % No. of real cells
    ncx = npx - 2;
    ncy = npy - 2;

    % North Ghost Cells are freestream and south wall is tengency BC
    for i = 2:ncx+1
        % North
        xcn = geoel(1,i,ncy+2);
        ycn = geoel(2,i,ncy+2);

        unkno(1,i,ncy+2) = u_inf*xcn + v_inf*ycn;

        % South
        % Ghost cells share wall with real cells in North
        nxs = geoel(6,i,1);
        nys = geoel(7,i,1);

        distn = geoel(2,i,2) - geoel(2,i,1);

        u_local = unkno(2,i,2);

        if abs(nys) < 1e-12
           unkno(1,i,1) = unkno(1,i,2);
        else
           unkno(1,i,1) = unkno(1,i,2) + u_local*(nxs/nys)*distn;
        end

    end

    % East and west ghost cells are far field BC
    for j = 2:ncy+1
        % West
        xcw = geoel(1,1,j);
        ycw = geoel(2,1,j);

        unkno(1,1,j) = u_inf*xcw + v_inf*ycw;

        % East
        xce = geoel(1,ncx+2,j);
        yce = geoel(2,ncx+2,j);

        unkno(1,ncx+2,j) = u_inf*xce + v_inf*yce;
    end

end

% This function uses the phi calculations for each cell center to calcuate
% the veolicty components and stores them in unkno array. Then it uses
% those components along with freestream flow vlaues to get the updated
% desnisty storing it in the unkno array as well. Ghost cell values for the
% all are just similar to the interior cells they share walls with.
```

```
function unkno = DensityfromPhi(unkno,geoel,gamma,Mach,a_inf,rho_inf)

    % Getting number of cells including ghost cells
    [~,npx,npy] = size(geoel);

    % No. of real cells
    ncx = npx - 2;
    ncy = npy - 2;

    % Velocity from Phi only for the real cells. Does not include ghost
    % cells. Ghost cell values can be copied from the real cells.
    for j = 1:ncy
        for i = 1:ncx

            % Taking area from previously calculated function
            area = geoel(3,i,j);

            % Face averaged Phi Values
            phiE = 0.5*(unkno(1,i+1,j+1) + unkno(1,i+2,j+1));
            phiN = 0.5*(unkno(1,i+1,j+1) + unkno(1,i+1,j+2));
            phiW = 0.5*(unkno(1,i+1,j+1) + unkno(1,i,j+1));
            phiS = 0.5*(unkno(1,i+1,j+1) + unkno(1,i+1,j));

            % Extracting outwards center normal vectors
            % East
            nEx = geoel(4,i+1,j+1);
            nEy = geoel(5,i+1,j+1);

            % North
            nNx = geoel(6,i+1,j+1);
            nNy = geoel(7,i+1,j+1);

            % West
            nWx = geoel(8,i+1,j+1);
            nWy = geoel(9,i+1,j+1);

            % South
            nSx = geoel(10,i+1,j+1);
            nSy = geoel(11,i+1,j+1);

            % Velcoity components u and v by multiplying phi values with
            % vector normals and dividing by the area
            % u
            unkno(2,i+1,j+1) = (phiE*nEx + phiN*nNx + phiW*nWx + phiS*nSx)/
area;

            %v
            unkno(3,i+1,j+1) = (phiE*nEy + phiN*nNy + phiW*nWy + phiS*nSy)/
area;
        end
    end

    % Density from velocity using the provided formula. This is
    % also only for the real cells.
```

```
    for j = 1:ncy
        for i = 1:ncx

            u = unkno(2,i+1,j+1);
            v = unkno(3,i+1,j+1);

            % Density Calculation
            rho = rho_inf*(1 + (gamma - 1)/2 * (Mach^2 - (u^2 + v^2)/
a_inf^2))^(1/(gamma - 1));

            % Putting it in the unknown array
            unkno(4,i+1,j+1) = rho;
        end
    end

    % Filling in the value of the ghost cells
    % North and south ghost cell boundary
    for i = 1:ncx

        % North
        unkno(2,i+1,npy) = unkno(2,i+1,npy-1);
        unkno(3,i+1,npy) = unkno(3,i+1,npy-1);
        unkno(4,i+1,npy) = unkno(4,i+1,npy-1);

        % South
        unkno(2,i+1,1) = unkno(2,i+1,2);
        unkno(3,i+1,1) = unkno(3,i+1,2);
        unkno(4,i+1,1) = unkno(4,i+1,2);
    end

    % East and west ghost cell boundary
    for j = 1:ncy

        % East
        unkno(2,npx,j+1) = unkno(2,npx-1,j+1);
        unkno(3,npx,j+1) = unkno(3,npx-1,j+1);
        unkno(4,npx,j+1) = unkno(4,npx-1,j+1);

        % West
        unkno(2,1,j+1) = unkno(2,2,j+1);
        unkno(3,1,j+1) = unkno(3,2,j+1);
        unkno(4,1,j+1) = unkno(4,2,j+1);
    end

    % Corner cells the velocity and other flow properties remain zero as
    % the surounding cells are ghost cells which do not depend on the
    % corner cell for their property calculations
end
% This function calculates the residuals and the flux coefficients for the
% each wall of the cell along with its center. Each face contribution is
% calculated seperately by taking average of density, cell volume, and square
% face area. This is used to calculate the flux and the residuals. The code
% is based on the provided scrap code.
```

```
function [resi,coeff] = resi_calc(unkno,geoel)

    % Getting number of cells including ghost cells
    [~,npx,npy] = size(geoel);

    % No. of real cells
    ncx = npx - 2;
    ncy = npy - 2;

    % Initialization of residual and coefficient matrices
    resi = zeros(ncx+2,ncy+2);
    coeff = zeros(5, ncx+2, ncy+2);

    for j = 2:ncy+1
        for i = 2:ncx+1

            % Density averages for cell face
            rhoE = 0.5*(unkno(4,i,j) + unkno(4,i+1,j));
            rhoN = 0.5*(unkno(4,i,j) + unkno(4,i,j+1));
            rhoS = 0.5*(unkno(4,i,j) + unkno(4,i,j-1));
            rhoW = 0.5*(unkno(4,i,j) + unkno(4,i-1,j));

            % Average cell volume
            volE = 0.5*(geoel(3,i,j) + geoel(3,i+1,j));
            volN = 0.5*(geoel(3,i,j) + geoel(3,i,j+1));
            volS = 0.5*(geoel(3,i,j) + geoel(3,i,j-1));
            volW = 0.5*(geoel(3,i,j) + geoel(3,i-1,j));

            % Squared face area
            area2E = geoel(4,i,j)^2 + geoel(5,i,j)^2;
            area2N = geoel(6,i,j)^2 + geoel(7,i,j)^2;
            area2S = geoel(10,i,j)^2 + geoel(11,i,j)^2;
            area2W = geoel(8,i,j)^2 + geoel(9,i,j)^2;

            % Coefficient calculations
            coeffE = rhoE*area2E/volE;
            coeffN = rhoN*area2N/volN;
            coeffS = rhoS*area2S/volS;
            coeffW = rhoW*area2W/volW;

            % Flux calculation
            FE = coeffE*(unkno(1,i+1,j) - unkno(1,i,j));
            FN = coeffN*(unkno(1,i,j+1) - unkno(1,i,j));
            FS = coeffS*(unkno(1,i,j-1) - unkno(1,i,j));
            FW = coeffW*(unkno(1,i-1,j) - unkno(1,i,j));

            % Residual calculation
            resi(i,j) = FE + FN + FS + FW;

            % Coeff matrix assignment
            coeff(1,i,j) = coeffE;
            coeff(2,i,j) = coeffN;
            coeff(3,i,j) = coeffW;
            coeff(4,i,j) = coeffS;
```

```
            coeff(5,i,j) = -(coeffE + coeffN + coeffW + coeffS); % centeral
        end
    end
end
% This function updates phi using Gauss Seidel method.

function [unkno,dphi,resi_n] = updatePhi(unkno,resi,coeff,omega)

    % Getting number of cells including ghost cells
    [~,npx,npy] = size(unkno);

    % No. of real cells
    ncx = npx - 2;
    ncy = npy - 2;

    % Intializing delta phi
    dphi = zeros(npx,npy);

    % Gauss-Seidel method
    for j = 2:ncy+1
        for i = 2:ncx+1

            coeffE = coeff(1,i,j); % East
            coeffN = coeff(2,i,j); % north
            coeffW = coeff(3,i,j); % West
            coeffS = coeff(4,i,j); % south
            coeffC = coeff(5,i,j); % center

            % temp storage
            dphitemp = dphi(i,j);

            % Gauss-Seidel calculation
            dphigs = (-resi(i,j) - coeffW*dphi(i-1,j) - coeffS*dphi(i,j-1) ...
                - coeffN*dphi(i,j+1) - coeffE*dphi(i+1,j))/coeffC;

            % SOR step
            dphi(i,j) = dphitemp + omega*(dphigs - dphitemp);
        end
    end

    for j = 2:ncy+1
        for i = 2:ncx+1
            unkno(1,i,j) = unkno(1,i,j) + dphi(i,j);
        end
    end

    % Residual normal
    resi_n = norm(resi(2:ncx+1,2:ncy+1),2);

end
```

Published with MATLAB® R2024b