Cairo University
Faculty of Engineering
Aerospace Department

Computational Fluid Dynamics

Project 1

Submitted to *Eng. Mina*

Name: ***Mohamed Ahmed Hassan Ahmed*** **BN: 10** **Section: 2**

Date:2022/ 5 / ***22***

Contents

Table of Figures

Problem Description

The governing equation of an incompressible potential two-dimensional flow past a Joukowski airfoil section can be written as: (Laplace equation) where "ψ" is the stream function.

Givens

BN.	Angle of Attach	% Camber / chord	% Thickness / chord
10	Sec(2) alpha = 8	5	7

Airfoil

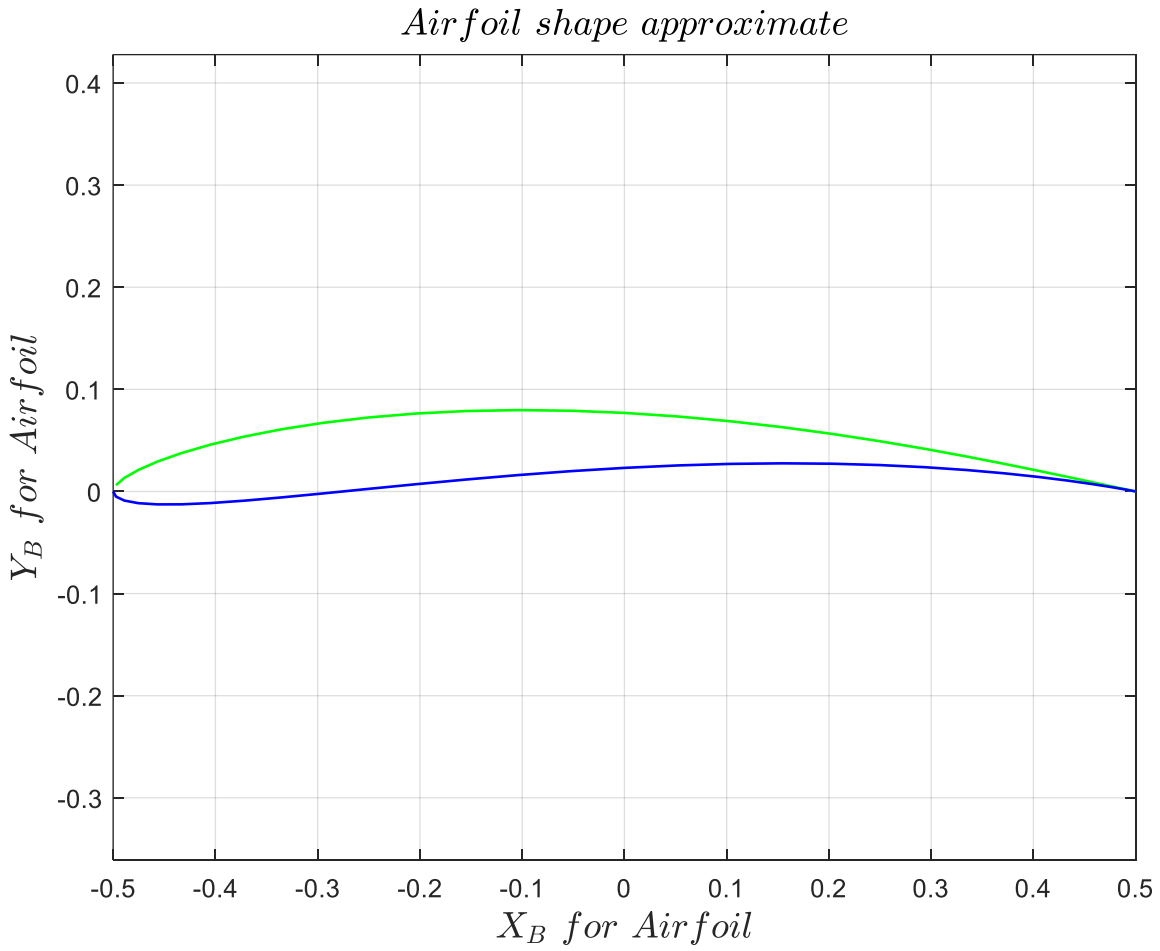


Figure 1. Airfoil

A. Construct a suitable boundary fitted grid (η1, η2) using (H-grid) or (***O-grid***) or (C-grid)

Constructing O-Grid was by linear interpolation of the values in the direction of η_2 from the surface of the Joukowski Airfoil to the surface of the outer Circle R.

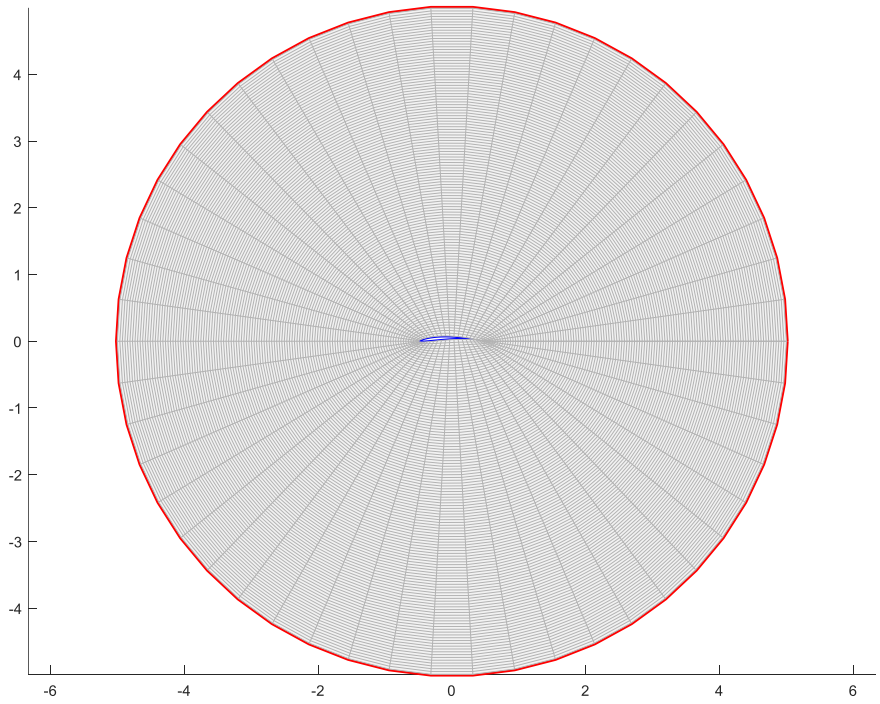


Figure 2. O-Grid Hole Figure

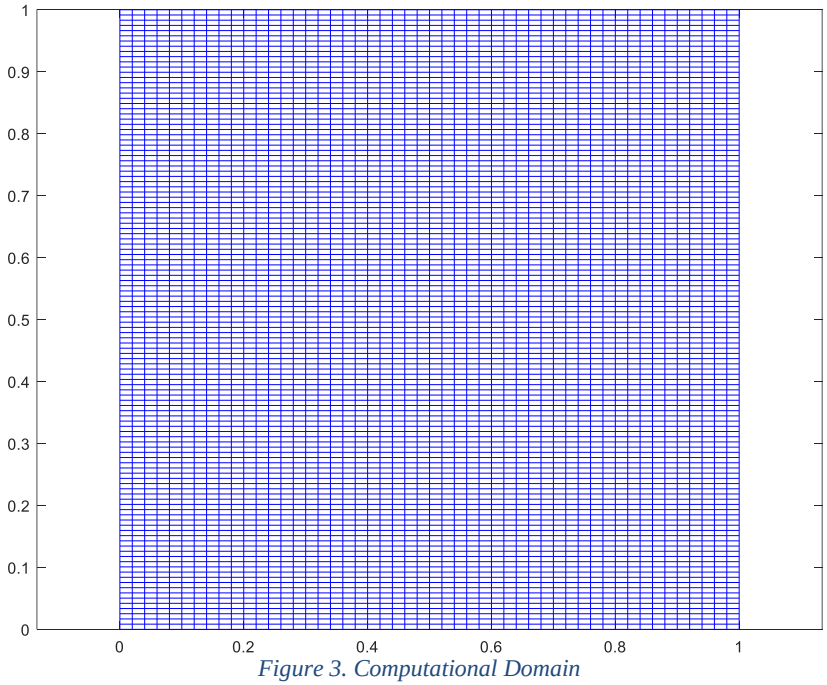


Figure 3. Computational Domain

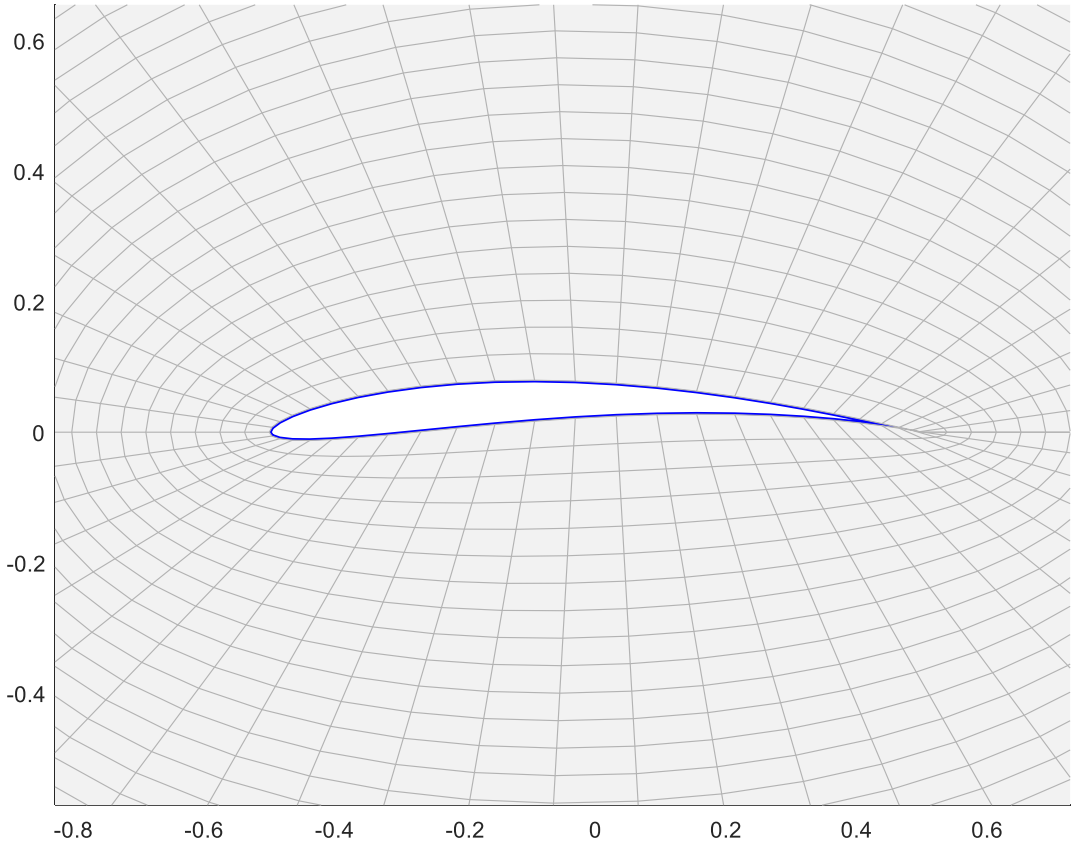


Figure 4. O-Grid Zoomed-in

B. Write the governing equation in the proposed body fitted grid (η1, η2)

$$\because \nabla^2\psi = \frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = \frac{\partial}{\partial \eta_1}\left[C_{11}\left(\frac{\partial\psi}{\partial \eta_1}\right) + C_{12}\left(\frac{\partial\psi}{\partial \eta_2}\right)\right] + \frac{\partial}{\partial \eta_2}\left[C_{12}\left(\frac{\partial\psi}{\partial \eta_1}\right) + C_{22}\left(\frac{\partial\psi}{\partial \eta_2}\right)\right]$$

$$= 0$$

$$\text{Where } \left\{ \begin{matrix} C_{11} = \frac{1}{J}\left[\left(\frac{\partial x}{\partial \eta_2}\right)^2 + \left(\frac{\partial y}{\partial \eta_2}\right)^2\right] & C_{12} = \frac{-1}{J}\left[\left(\frac{\partial x}{\partial \eta_1}\right)\left(\frac{\partial x}{\partial \eta_2}\right) + \left(\frac{\partial y}{\partial \eta_1}\right)\left(\frac{\partial y}{\partial \eta_2}\right)\right] \\ C_{22} = \frac{1}{J}\left[\left(\frac{\partial x}{\partial \eta_1}\right)^2 + \left(\frac{\partial y}{\partial \eta_1}\right)^2\right] & J = \left[\left(\frac{\partial x}{\partial \eta_1}\right)\left(\frac{\partial y}{\partial \eta_2}\right) - \left(\frac{\partial y}{\partial \eta_1}\right)\left(\frac{\partial x}{\partial \eta_2}\right)\right] \end{matrix} \right\}$$

C. Choose the numerical method used to solve the governing equation (***PSOR***) or (LSOR) or (ADI)

We shall use **Point – SOR** with value of $\boldsymbol{\omega = 1.1}$

The general form of the transformed Laplace equation

$$S_{ij}\psi_{ij} = S_{i-1j}\psi_{i-1j} + S_{i+1j}\psi_{i+1j} + S_{ij+1}\psi_{ij+1} + S_{ij-1}\psi_{ij-1} + S_{i-1j-1}\psi_{i-1j-1} + S_{i-1j+1}\psi_{i-1j+1} + S_{i+1j-1}\psi_{i+1j-1} + S_{i+1j+1}\psi_{i+1j+1}$$

Where,

$S_{ij} = (c_{11})_{i+1/2j} + (c_{11})_{i-1/2j} + \left[(c_{22})_{ij+1/2} + (c_{22})_{ij-1/2}\right]\left(\frac{\Delta\eta_1}{\Delta\eta_2}\right)^2$	$S_{i-1j-1} = \left(\frac{\Delta\eta_1}{4\Delta\eta_2}\right)\left[(c_{12})_{i-1/2j} + (c_{12})_{ij-1/2}\right]$	$S_{i-1j+1} = -\left(\frac{\Delta\eta_1}{4\Delta\eta_2}\right)\left[(c_{12})_{i-1/2j} + (c_{12})_{ij+1/2}\right]$
$S_{i+1j-1} = -\left(\frac{\Delta\eta_1}{4\Delta\eta_2}\right)\left[(c_{12})_{i+1/2j} + (c_{12})_{ij-1/2}\right]$	$S_{i+1j+1} = \left(\frac{\Delta\eta_1}{4\Delta\eta_2}\right)\left[(c_{12})_{i+1/2j} + (c_{12})_{ij+1/2}\right]$	$S_{i-1j} = (c_{11})_{i-1/2j} - \left(\frac{\Delta\eta_1}{2\Delta\eta_2}\right)^2\left[(c_{12})_{ij+1/2} - (c_{12})_{ij-1/2}\right]$
$S_{i+1j} = (c_{11})_{i+1/2j} + \left(\frac{\Delta\eta_1}{2\Delta\eta_2}\right)^2\left[(c_{12})_{ij+1/2} - (c_{12})_{ij-1/2}\right]$	$S_{ij-1} = \left(\frac{\Delta\eta_1}{\Delta\eta_2}\right)^2(c_{22})_{ij-1/2} - \left(\frac{\Delta\eta_1}{4\Delta\eta_2}\right)\left[(c_{12})_{i+1/2j} - (c_{12})_{i-1/2j}\right]$	$S_{ij+1} = \left(\frac{\Delta\eta_1}{\Delta\eta_2}\right)^2(c_{22})_{ij+1/2} + \left(\frac{\Delta\eta_1}{4\Delta\eta_2}\right)\left[(c_{12})_{i+1/2j} - (c_{12})_{i-1/2j}\right]$

D. Determine the values of the stream function ψ at the outer boundaries

$$d\psi = \frac{\partial \psi}{\partial x}\, c + \frac{\partial \psi}{\partial y}\, dy\,, \qquad \frac{\partial \psi}{\partial x} = -v = -V_\infty \sin(\alpha)\,, \qquad \frac{\partial \psi}{\partial y} = u = V_\infty \cos(\alpha)$$

$$\therefore d\psi = -V_\infty \sin(\alpha)\ dx + V_\infty \cos(\alpha)\ dy$$

Let ψ at start point "$\psi_{1,1}$" = 0

Then, Along the outer circle of radius R,

$$\therefore \psi_{i+1,j+1} = \psi_{i,j} + V_\infty \cos(\alpha)\left(y_{j+1} - y_j\right) - V_\infty \cos(\alpha)\left(x_{i+1} - x_i\right)$$

$$i \in [1, i_{max}]\,, \qquad j \in [1, j_{max}]$$

E. Choose a suitable initial value of the stream function ψ for all points in the grid points

Thus, we could do linear interpolation from the inner airfoil surface values of $\psi = 0$ to the outer surface values of the boundary conditions, in the direction of η_2.

F. Obtain the numerical solution until convergence

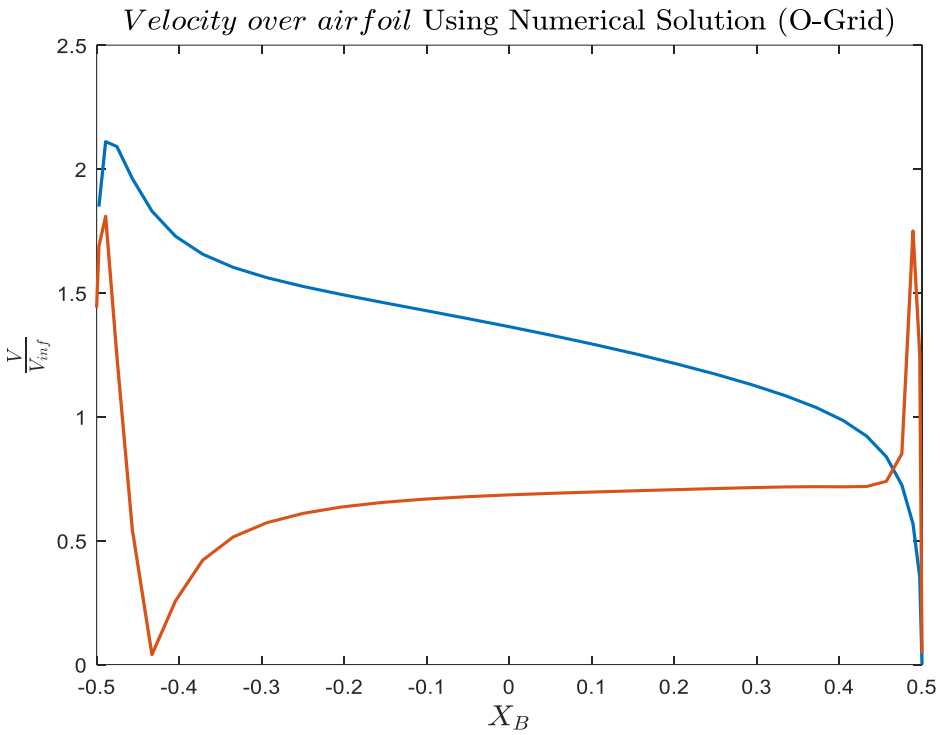


Figure 5. Numerical Solution of non-dimensional velocity using O-Grid

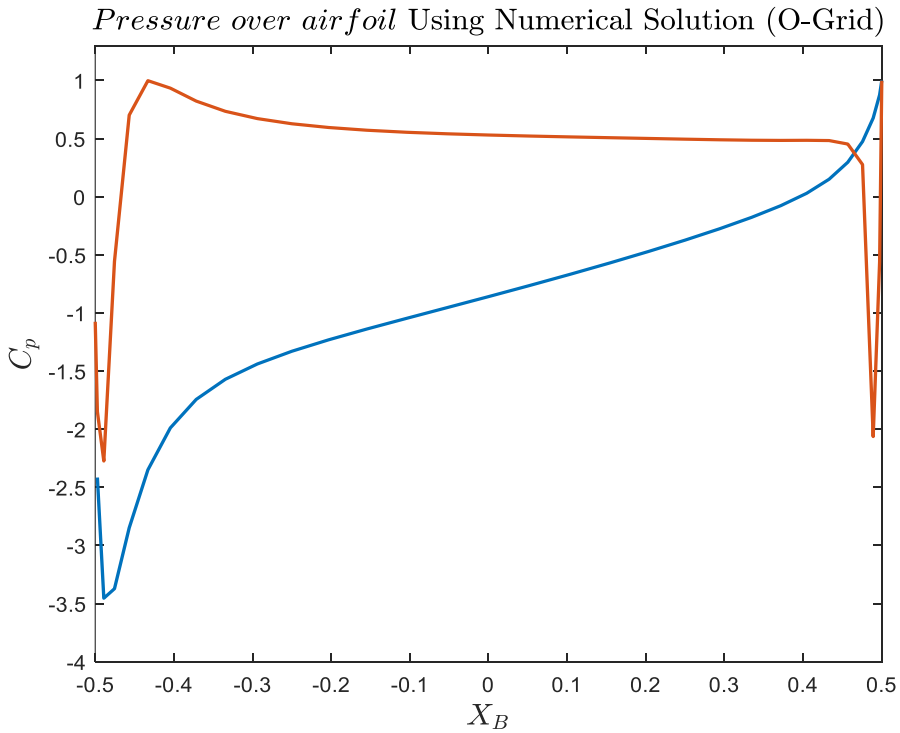
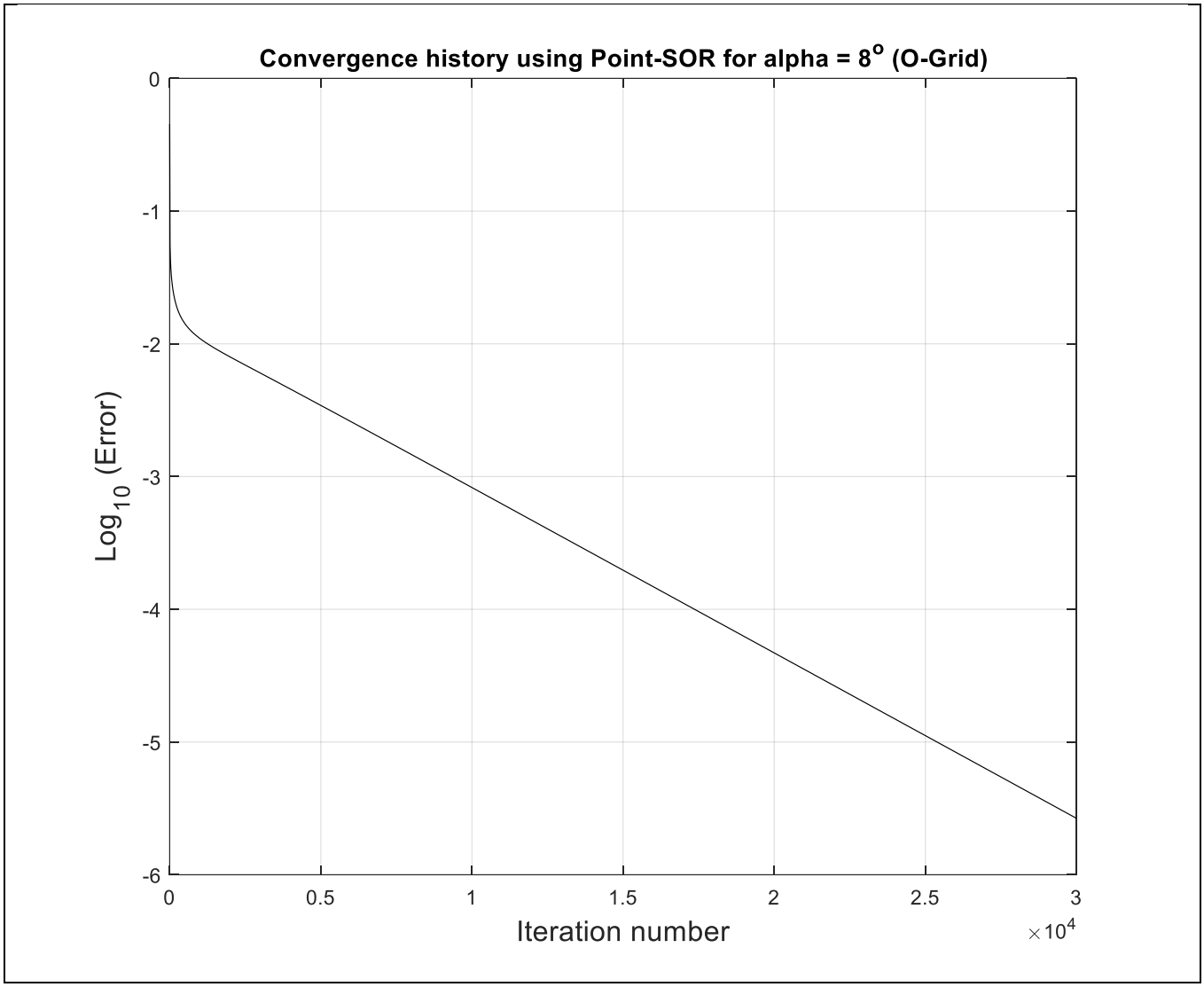


Figure 6. Numerical Solution of Pressure Coefficient using O-Grid

G. Show the results of the convergence history



H. Show the iso-velocity and iso-pressure lines in the entire domain

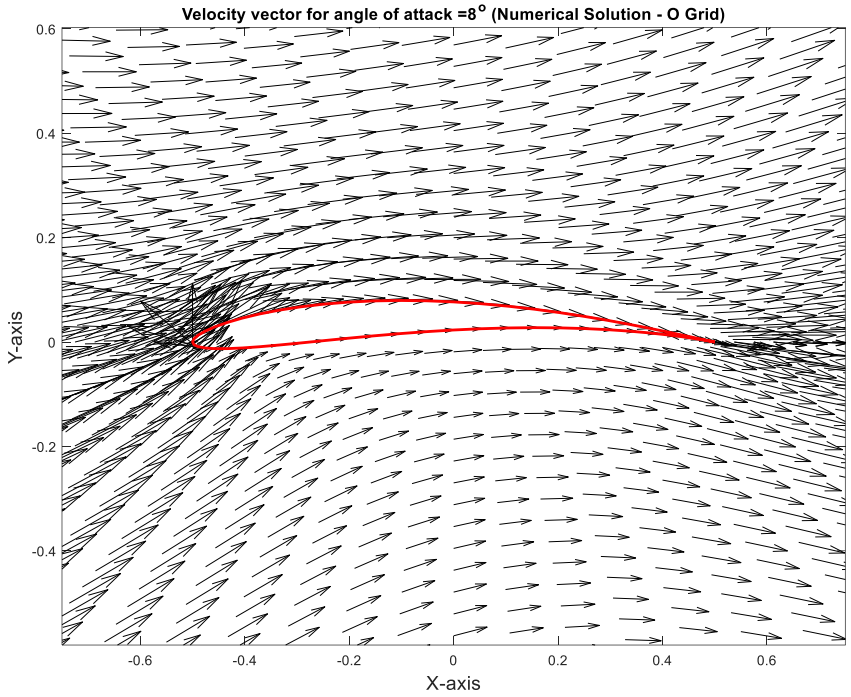


Figure 7. Iso-Velocity in Entire Domain

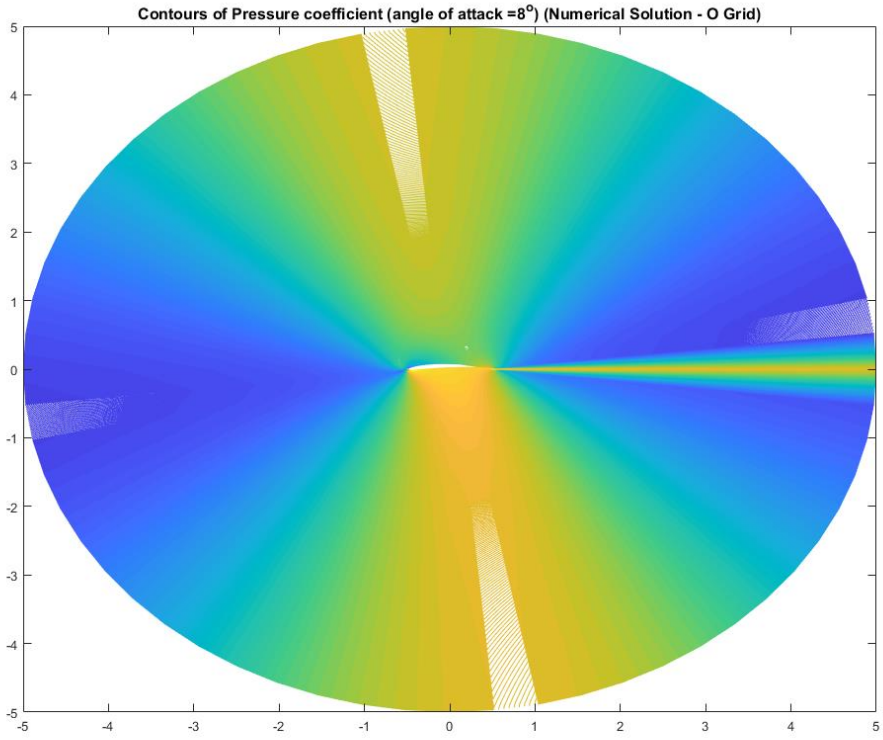


Figure 8. Iso-Pressure in Entire Domain

I. Comparing with the potential flow results obtained by the Joukowski transformation between the circle and the airfoil.

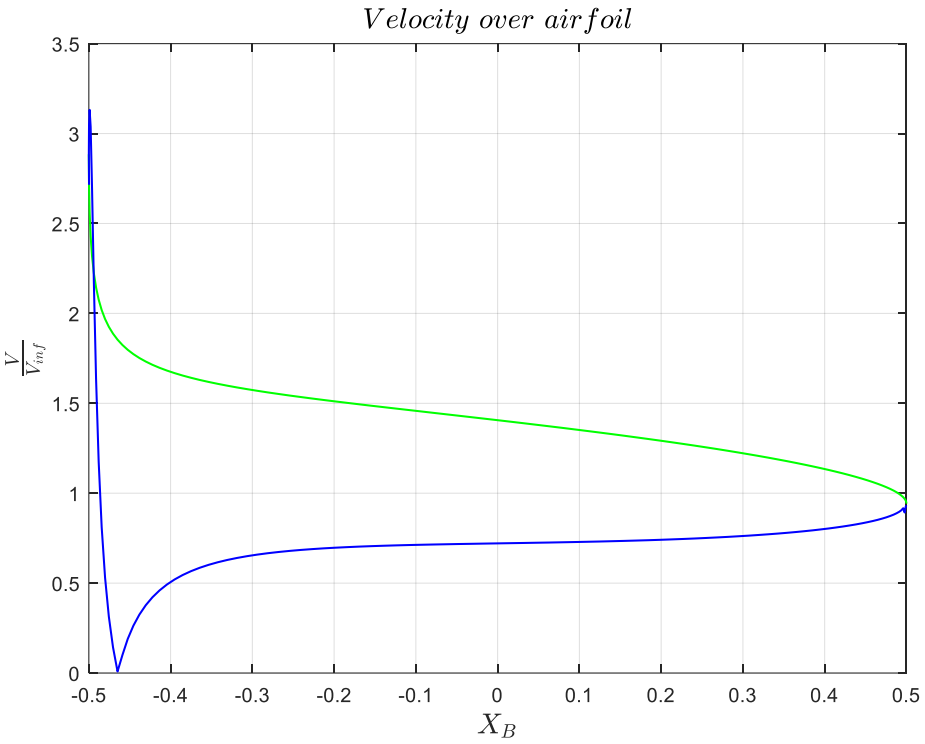


Figure 9. Exact Solution of Velocity

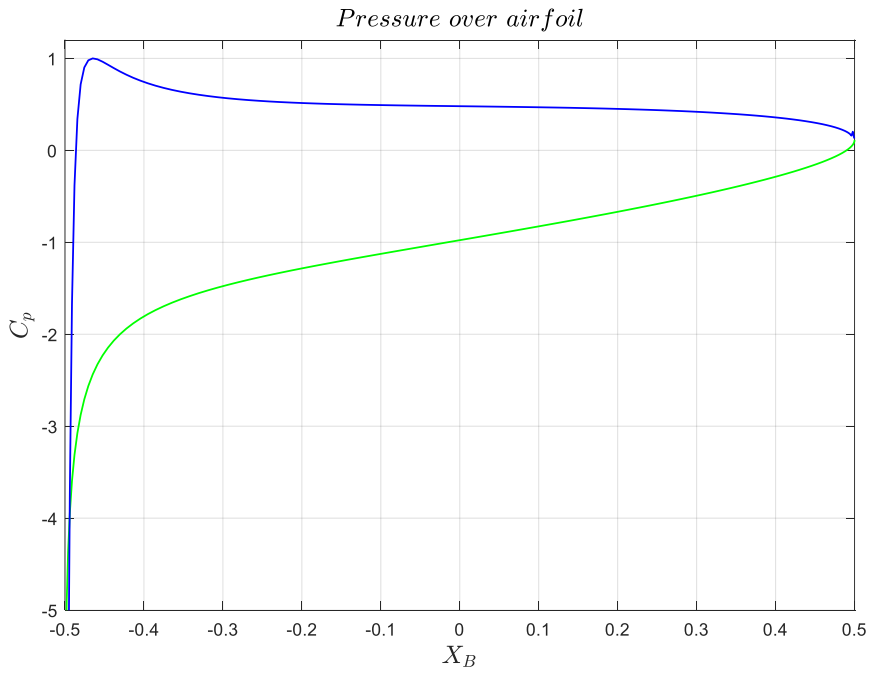


Figure 10. Exact Solution of Pressure

Appendix: Code

Main.m (script)

```
%% Clear
clear;close all;clc;

%% Initializing Inputs
inputs = struct(...
    'chord', 1,...
    'Vinf', 100,...
    'max_thickness', 0.07,...
    'max_camber', 0.05,...
    'alpha_deg', 8,...
    'i_max', 61,...
    'j_max', 121,...
    'n', 30000,...
    'R', 5);

%% Initializing Airfoil Class
airfoil = Airfoil(inputs);

% Calculate Joukowski Airfoil
[~, outerCircle, joukowski] = airfoil.joukowskiAirfoil();

% Plot Joukowski Airfoil
figure;
plot(joukowski.x(1:end/2), joukowski.y(1:end/2),'g-','LineWidth',1); hold on
plot(joukowski.x(end/2: end), joukowski.y(end/2: end), 'b-','LineWidth',1);
xlabel('$X_{B}$ $for$ $Airfoil$','interpreter','latex','FontSize',14);
ylabel('$Y_{B}$ $for$ $Airfoil$','interpreter','latex','FontSize',14);
title('$Airfoil$ $shape$ $approximate$','interpreter','latex','FontSize',14);
axis equal
grid on

% Calculate & Plot O-Grid
figure;
hold on;
plot(outerCircle.x, outerCircle.y, '-r', 'LineWidth', 3);
plot(joukowski.x, joukowski.y, '-b', 'LineWidth', 2);
axis equal;
[xGrid, yGrid, ~] = airfoil.generatePhysicalGrid(outerCircle, joukowski);

% Calculate & Plot Computational Grid
figure;
[eta1Grid, eta2Grid] = airfoil.generateComputationalGrid([0, 1], [0, 1]);

% Calculate Metric derivatives x/eta1, y/eta1, x/eta2, y/eta2
% And Calculate Values of C11, C22, C12, J
airfoil.transformationMetrics(xGrid, yGrid, eta1Grid, eta2Grid);

% Calculate & return Boundary Conditions
psi_ = airfoil.calculateDirichletBoundary();
n = 1;
errorLog10 = ones(1, airfoil.inputs.n);
error = ones(1, airfoil.inputs.n);
while((n <= airfoil.inputs.n) && (min(error) > 1e-8))
```

```
    psi_new = airfoil.iterate(psi_);
    psi_new = psi_ + 1.1 .* (psi_new - psi_);

    error(n) = max(max(abs(psi_new-psi_)));
    if error(n) > 0; errorLog10(n) = log10(error(n)); end

    psi_ = psi_new;
    psi_(:,1) = psi_(1,2);
    n = n + 1;
end

%% Plot Error History
figure;
plot(errorLog10,'k')
grid on
xlabel('Iteration number', 'fontsize',14)
ylabel('Log_1_0 (Error)', 'fontsize',14)
title('Convergence history using Point-SOR for alpha = 8^o (O-
Grid)','fontsize',12)

%% Calculate velocity and Cp

dpsi_deta1 = airfoil.zerosImaxJmax();
dpsi_deta2 = airfoil.zerosImaxJmax();
for i=1:airfoil.inputs.i_max
     if i ==1
         dpsi_deta1(i,:) = (psi_(i+1,:)-psi_(i,:))./(eta1Grid(i+1,:)-
eta1Grid(i,:));
     elseif i==airfoil.inputs.i_max
         dpsi_deta1(i,:) = (psi_(i,:)-psi_(i-1,:))./(eta1Grid(i,:)-eta1Grid(i-
1,:));
     else
         dpsi_deta1(i,:) = (psi_(i+1,:)-psi_(i-1,:))./(eta1Grid(i+1,:)-
eta1Grid(i,:));
     end
 end
for j=1:airfoil.inputs.j_max
     if j ==1
         dpsi_deta2(:,j) = (psi_(:,j+1)-psi_(:,j))./(eta2Grid(:,j+1)-
eta2Grid(:,j));
     elseif j==airfoil.inputs.j_max
          dpsi_deta2(:,j) = (psi_(:,j)-psi_(:,j-1))./(eta2Grid(:,j)-eta2Grid(:,j-
1));
     else
         dpsi_deta2(:,j) = (psi_(:,j+1)-psi_(:,j-1))./(eta2Grid(:,j+1)-
eta2Grid(:,j-1));
     end
end
u = dpsi_deta1.*airfoil.deta1_dy + dpsi_deta2.*airfoil.deta2_dy;
v = -(dpsi_deta1.*airfoil.deta1_dx + dpsi_deta2.* airfoil.deta2_dx);

V = sqrt(u.^2+v.^2);
nonDimV = V/airfoil.inputs.Vinf;
C_p = 1-(nonDimV).^2;
```

```
%% Plotting Results
figure
quiver(xGrid,yGrid,u, v, 'k-', 'LineWidth', 0.5);hold on
plot(joukowski.x, joukowski.y,'r-','LineWidth',2) ; hold on
axis equal
xlim([-(airfoil.inputs.chord+0.5)/2 (airfoil.inputs.chord+0.5)/2])
xlabel('X-axis', 'fontsize',14)
ylabel('Y-axis', 'fontsize',14)
title('Velocity vector for angle of attack =8^o (Numerical Solution - O
Grid)','fontsize',12)

figure;
contour(xGrid, yGrid, C_p, 5000);
title('Contours of Pressure coefficient (angle of attack =8^o) (Numerical Solution
- O Grid)','fontsize',12)

figure
plot(xGrid(1:end/2,1),nonDimV(1:end/2,1), 'LineWidth', 1.5);
hold on;
plot(xGrid(end/2:end,1),nonDimV(end/2:end,1), 'LineWidth', 1.5);
xlabel('$X_{B}$','interpreter','latex','FontSize',14);
ylabel('$\frac{V}{V_{inf}}$','interpreter','latex','FontSize',14);
title('$Velocity$ $over$ $airfoil$ Using Numerical Solution (O-
Grid)','interpreter','latex','FontSize',14);

figure;
plot(xGrid(1:end/2,1),C_p(1:end/2,1), 'LineWidth', 1.5);
hold on;
plot(xGrid(end/2:end,1),C_p(end/2:end,1), 'LineWidth', 1.5);
ylim([-4 1.3]);
xlabel('$X_{B}$','interpreter','latex','FontSize',14);
ylabel('$C_{p}$','interpreter','latex','FontSize',14);
title('$Pressure$ $over$ $airfoil$ Using Numerical Solution (O-
Grid)','interpreter','latex','FontSize',14);
```

Airfoil.m (Class)

```
classdef Airfoil < handle

    properties
        inputs; %Inputs to Constructor function

        b, e, beta, a, alpha, x0, y0, cosa, sina; % Calculated form Inputs

        delta_eta1, delta_eta2;

        dx_deta1, dy_deta1, dx_deta2, dy_deta2; % Metrics of Transformation
        deta1_dx, deta1_dy, deta2_dx, deta2_dy;

        Jacobian, C11, C22, C12;

        C11_plusHalf_i, C11_negHalf_i,
        C22_plusHalf_i, C22_negHalf_i,
        C12_plusHalf_i, C12_negHalf_i,
```

```
        C11_plusHalf_j, C11_negHalf_j,
        C22_plusHalf_j, C22_negHalf_j,
        C12_plusHalf_j, C12_negHalf_j,

        s_i_j
        s_in1_j
        s_ip1_j
        s_i_jn1
        s_i_jp1
        s_in1_jn1, s_ip1_jp1
        s_in1_jp1, s_ip1_jn1
    end

    methods
        function obj = Airfoil(inputs_struct)
            obj.inputs = inputs_struct;
            obj.b = inputs_struct.chord/4;
            obj.e = inputs_struct.max_thickness/1.3;
            obj.beta = 2*inputs_struct.max_camber;
            obj.a = obj.b *(1+obj.e)/cos(obj.beta);
            obj.alpha = inputs_struct.alpha_deg * pi / 180;
            obj.x0 = -obj.b * obj.e;
            obj.y0 = obj.a * sin(obj.beta);
            obj.cosa = cos(obj.alpha);
            obj.sina = sin(obj.alpha);
        end

        function [innerCircle, outerCircle, airfoil] = joukowskiAirfoil(this)
            theta = linspace(0, 2*pi, this.inputs.i_max);

            innerCircle = struct(...
                'x', this.inputs.chord/2*cos(theta),...
                'y', this.inputs.chord/2*sin(theta));

            outerCircle = struct(...
                'x', this.inputs.R*cos(theta),...
                'y', this.inputs.R*sin(theta));

            sign=(sin(theta)./abs(sin(theta)));
            sign(1)=1;

            joukowski_y = 2*this.b*this.e*(1-innerCircle.x/2/this.b)...
                .*sign.*sqrt(1-(innerCircle.x/2/this.b).^2)...
                + 2*this.b*this.beta*(1-(innerCircle.x/2/this.b).^2);
            airfoil = struct(...
                'x', innerCircle.x,...
                'y', joukowski_y);

        end

        function [xGrid, yGrid, rGrid] = generatePhysicalGrid(this, outerCircle,
airfoil)
            xGrid = zeros(this.inputs.i_max, this.inputs.j_max);
            yGrid = zeros(this.inputs.i_max, this.inputs.j_max);
```

```
            for i=1:this.inputs.i_max

xGrid(i,:)=linspace(airfoil.x(i),outerCircle.x(i),this.inputs.j_max);

yGrid(i,:)=linspace(airfoil.y(i),outerCircle.y(i),this.inputs.j_max);
            end

            rGrid = sqrt(xGrid.^2+yGrid.^2);
            colormap([0.95 0.95 0.95]);
            g = pcolor(xGrid, yGrid, rGrid);
            set(g, 'EdgeColor', [0.7 0.7 0.7]);
        end


        function [Eta1, Eta2] = generateComputationalGrid(this, eta1_limit,
eta2_limit)
            eta1 = linspace(eta1_limit(1), eta1_limit(2), this.inputs.i_max);
            eta2 = linspace(eta2_limit(1), eta2_limit(2), this.inputs.j_max);
            [Eta2, Eta1] = meshgrid(eta2, eta1);
            plot(Eta1, Eta2, Eta1', Eta2', 'Color', 'b');
            axis equal;
            this.delta_eta1 = (eta1_limit(2)-eta1_limit(1))/(this.inputs.i_max-1);
            this.delta_eta2 = (eta2_limit(2)-eta2_limit(1))/(this.inputs.j_max-1);
        end

        function [] = transformationMetrics(this, xGrid, yGrid, eta1Grid,
eta2Grid)
            this.dx_deta1 = this.zerosImaxJmax();
            this.dy_deta1 = this.zerosImaxJmax();
            for i = 1:this.inputs.i_max
                if(i == 1)
                    this.dx_deta1(i,:) =(xGrid(i+1,:)-
xGrid(i,:))./(eta1Grid(i+1,:)-eta1Grid(i,:));
                    this.dy_deta1(i,:) =(yGrid(i+1,:)-
yGrid(i,:))./(eta1Grid(i+1,:)-eta1Grid(i,:));
                elseif (i == this.inputs.i_max)
                    this.dx_deta1(i,:) =(xGrid(i,:)-xGrid(i-1,:))./(eta1Grid(i,:)-
eta1Grid(i-1,:));
                    this.dy_deta1(i,:) =(yGrid(i,:)-yGrid(i-1,:))./(eta1Grid(i,:)-
eta1Grid(i-1,:));
                else
                    this.dx_deta1(i,:) =(xGrid(i+1,:)-xGrid(i-
1,:))./(eta1Grid(i+1,:)-eta1Grid(i-1,:));
                    this.dy_deta1(i,:) =(yGrid(i+1,:)-yGrid(i-
1,:))./(eta1Grid(i+1,:)-eta1Grid(i-1,:));
                end
            end

            this.dx_deta2 = this.zerosImaxJmax();
            this.dy_deta2 = this.zerosImaxJmax();
            for j = 1:this.inputs.j_max
                if(j == 1)
                    this.dx_deta2(:,j) =(xGrid(:,j+1)-
xGrid(:,j))./(eta2Grid(:,j+1)-eta2Grid(:,j));
                    this.dy_deta2(:,j) =(yGrid(:,j+1)-
```

```
yGrid(:,j))./(eta2Grid(:,j+1)-eta2Grid(:,j));
                elseif (j == this.inputs.j_max )
                    this.dx_deta2(:,j) =(xGrid(:,j)-xGrid(:,j-1))./(eta2Grid(:,j)-
eta2Grid(:,j-1));
                    this.dy_deta2(:,j) =(yGrid(:,j)-yGrid(:,j-1))./(eta2Grid(:,j)-
eta2Grid(:,j-1));
                else
                    this.dx_deta2(:,j) =(xGrid(:,j+1)-xGrid(:,j-
1))./(eta2Grid(:,j+1)-eta2Grid(:,j-1));
                    this.dy_deta2(:,j) =(yGrid(:,j+1)-yGrid(:,j-
1))./(eta2Grid(:,j+1)-eta2Grid(:,j-1));
                end
            end

            this.Jacobian = this.dx_deta1.*this.dy_deta2-
this.dx_deta2.*this.dy_deta1;
            this.deta1_dx = this.dy_deta2./this.Jacobian;
            this.deta1_dy = -this.dx_deta2./this.Jacobian;
            this.deta2_dx = -this.dy_deta1./this.Jacobian;
            this.deta2_dy = this.dx_deta1./this.Jacobian;



            this.C11 = (this.dx_deta2.^2+this.dy_deta2.^2)./this.Jacobian;
            this.C22 = (this.dx_deta1.^2+this.dy_deta1.^2)./this.Jacobian;
            this.C12 = -
(this.dx_deta1.*this.dx_deta2+this.dy_deta1.*this.dy_deta2)./this.Jacobian;

            this.C11_plusHalf_i = this.zerosImaxJmax();
            this.C11_negHalf_i = this.zerosImaxJmax();
            this.C22_plusHalf_i = this.zerosImaxJmax();
            this.C22_negHalf_i = this.zerosImaxJmax();
            this.C12_plusHalf_i = this.zerosImaxJmax();
            this.C12_negHalf_i = this.zerosImaxJmax();

            for i=1:this.inputs.i_max
               if(i ~= this.inputs.i_max)
                    this.C11_plusHalf_i(i, :) = (this.C11(i, :)+this.C11(i+1,
:))/2;
                    this.C22_plusHalf_i(i, :) = (this.C22(i, :)+this.C22(i+1,
:))/2;
                    this.C12_plusHalf_i(i, :) = (this.C12(i, :)+this.C12(i+1,
:))/2;
               end
               if(i ~= 1)
                    this.C11_negHalf_i(i, :) = (this.C11(i, :)+this.C11(i-1, :))/2;
                    this.C22_negHalf_i(i, :) = (this.C22(i, :)+this.C22(i-1, :))/2;
                    this.C12_negHalf_i(i, :) = (this.C12(i, :)+this.C12(i-1, :))/2;
               end
            end
            this.C11_plusHalf_i(this.inputs.i_max, :) = this.C11_plusHalf_i(1, :);
            this.C22_plusHalf_i(this.inputs.i_max, :) = this.C22_plusHalf_i(1, :);
            this.C12_plusHalf_i(this.inputs.i_max, :) = this.C12_plusHalf_i(1, :);

            this.C11_negHalf_i(1, :) = this.C11_negHalf_i(this.inputs.i_max, :);
            this.C22_negHalf_i(1, :) = this.C22_negHalf_i(this.inputs.i_max, :);
            this.C12_negHalf_i(1, :) = this.C12_negHalf_i(this.inputs.i_max, :);
```

```
            this.C11_plusHalf_j = this.zerosImaxJmax();
            this.C11_negHalf_j = this.zerosImaxJmax();
            this.C22_plusHalf_j = this.zerosImaxJmax();
            this.C22_negHalf_j = this.zerosImaxJmax();
            this.C12_plusHalf_j = this.zerosImaxJmax();
            this.C12_negHalf_j = this.zerosImaxJmax();

            for j=2:this.inputs.j_max-1
                this.C11_plusHalf_j(:,j) = (this.C11(:,j)+this.C11(:,j+1))/2;
                this.C11_negHalf_j(:,j) = (this.C11(:,j)+this.C11(:,j-1))/2;
                this.C22_plusHalf_j(:,j) = (this.C22(:,j)+this.C22(:,j+1))/2;
                this.C22_negHalf_j(:,j) = (this.C22(:,j)+this.C22(:,j-1))/2;
                this.C12_plusHalf_j(:,j) = (this.C12(:,j)+this.C12(:,j+1))/2;
                this.C12_negHalf_j(:,j) = (this.C12(:,j)+this.C12(:,j-1))/2;
            end

            deta1_deta2 = this.delta_eta1/this.delta_eta2;

            this.s_i_j = this.C11_plusHalf_i + this.C11_negHalf_i +
(this.C22_plusHalf_j + this.C22_negHalf_j)*(deta1_deta2)^2;

            this.s_in1_j = this.C11_negHalf_i - (this.C12_plusHalf_j -
this.C12_negHalf_j)*(deta1_deta2/2)^2;
            this.s_ip1_j = this.C11_plusHalf_i + (this.C12_plusHalf_j -
this.C12_negHalf_j)*(deta1_deta2/2)^2;

            this.s_i_jn1 = (this.C22_negHalf_j)*(deta1_deta2)^2 -
(this.C12_plusHalf_i - this.C12_negHalf_i)*(deta1_deta2/4);
            this.s_i_jp1 = (this.C22_plusHalf_j)*(deta1_deta2)^2 +
(this.C12_plusHalf_i - this.C12_negHalf_i)*(deta1_deta2/4);

            this.s_in1_jn1 = (this.C12_negHalf_i +
this.C12_negHalf_j)*(deta1_deta2/4);
            this.s_ip1_jp1 = (this.C12_plusHalf_i +
this.C12_plusHalf_j)*(deta1_deta2/4);

            this.s_in1_jp1 = -(this.C12_negHalf_i +
this.C12_plusHalf_j)*(deta1_deta2/4);
            this.s_ip1_jn1 = -(this.C12_plusHalf_i +
this.C12_negHalf_j)*(deta1_deta2/4);


        end

        function [psi] = calculateDirichletBoundary(this)
            theta = linspace(0, 2*pi, this.inputs.i_max);
            u = this.inputs.Vinf * this.cosa;
            v = this.inputs.Vinf * this.sina;

            x = this.inputs.R * cos(theta);
            y = this.inputs.R * sin(theta);

            psiBoundary = zeros(this.inputs.i_max, 1);
```

```
            for i=2:this.inputs.i_max
                psiBoundary(i,1) = psiBoundary(i-1, 1) - v*(x(i) - x(i-1)) +
u*(y(i) - y(i-1));
            end

            psi = this.zerosImaxJmax();
            psi(:, end) = psiBoundary;

            for i=1:this.inputs.i_max
                psi(i, 1:end) = linspace(psi(i,1),psi(i,end),this.inputs.j_max);
            end
        end

        function [psi] = iterate(this, psi_pre)

            psi = this.zerosImaxJmax();
            psi(:, 1) = psi_pre(:, 1);
            psi(:,end) = psi_pre(:, end);

            for i=1:this.inputs.i_max
                for j=2:this.inputs.j_max-1
                    if i==1
                        psi(i,j)=(this.s_in1_j(i,j) * psi_pre(this.inputs.i_max-
1,j) + this.s_ip1_j(i,j) * psi_pre(i+1,j)...
                            + this.s_i_jp1(i,j) * psi_pre(i,j+1)
+this.s_i_jn1(i,j)*psi(i,j-1)...
                            + this.s_in1_jn1(i,j)*psi_pre(this.inputs.i_max-1,j-
1)+this.s_in1_jp1(i,j)*psi_pre(this.inputs.i_max-
1,j+1)+this.s_ip1_jn1(i,j)*psi_pre(i+1,j-
1)+this.s_ip1_jp1(i,j)*psi_pre(i+1,j+1))...
                            /this.s_i_j(i,j);
                    elseif i== this.inputs.i_max
                        psi(i,j)=(this.s_in1_j(i,j) * psi(i-1,j) +
this.s_ip1_j(i,j) * psi_pre(1+1,j)...
                            + this.s_i_jp1(i,j) * psi_pre(i,j+1)
+this.s_i_jn1(i,j)*psi(i,j-1)...
                            + this.s_in1_jn1(i,j)*psi(i-1,j-
1)+this.s_in1_jp1(i,j)*psi(i-1,j+1)+this.s_ip1_jn1(i,j)*psi(1+1,j-
1)+this.s_ip1_jp1(i,j)*psi_pre(1+1,j+1))...
                            /this.s_i_j(i,j);
                    else
                        psi(i,j)=(this.s_in1_j(i,j) * psi(i-1,j) +
this.s_ip1_j(i,j) * psi_pre(i+1,j)...
                            + this.s_i_jp1(i,j) * psi_pre(i,j+1)
+this.s_i_jn1(i,j)*psi(i,j-1)...
                            + this.s_in1_jn1(i,j)*psi(i-1,j-
1)+this.s_in1_jp1(i,j)*psi(i-1,j+1)+this.s_ip1_jn1(i,j)*psi_pre(i+1,j-
1)+this.s_ip1_jp1(i,j)*psi_pre(i+1,j+1))...
                            /this.s_i_j(i,j);
                    end
                end
            end
        end

        function [m] = zerosImaxJmax(this)
            m = zeros(this.inputs.i_max, this.inputs.j_max);
```

```
        end

    end
end
```