Cairo University
Aerospace Department

Flow Solution Over a Joukowski Airfoil

Using O-grid Numerical Method and Z-Transformation Analytical Method

AER 4110

Computational Fluid Dynamics

Name	Section	Bn
Ahmed Mohamed Hassan	1	12
Shehab Osama	1	29
Mohamed Gamal Abd-Elnaser	2	10
Mohamed Trek Mohamed Amin	2	19

Submitted to:
Eng. Mina Romany

2-January 2023

Table of Contents

List of Figures

List of Tables

Problem Statement

The governing equation of an incompressible potential two dimensional flow past a Joukowski airfoil section can be written as: $\nabla^2 y = 0$ (Laplace equation), where "y" is the stream function.

a) Construct a suitable boundary fitted grid (h_1 &h_2) using (H-grid) or (O-grid) or (C-grid).

b) Write the governing equation in the proposed body fitted grid (h_1 & h_2).

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

d) Determine the values of the stream function ψ at the outer boundaries

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

f) Obtain the numerical solution until convergence.

g) Show the results of the convergence history (RMS error with the iteration number).

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

i) Show the velocity and pressure distributions over the upper and lower surfaces of the airfoil and compare with the potential flow results obtained by Joukowski transformation between the circle and the airfoil.

j) Solve the airfoil using the numerical solution of the Navier Stokes equation with a suitable turbulence model and compare the results obtained (grid, convergence history, the iso-velocity and iso-pressure lines in the entire domain, the pressure distributions over the upper and lower surfaces of the airfoil) with those obtained using the potential flow solution

Introduction

Grid generation

Mesh generation in CFD plays important Role as meshing in finite element simulations, where discretization will determine the accuracy and computation time in the simulation. Grid generation in CFD simulations involves choosing a mathematical technique to represent the arrangement and spacing between each node in the numerical grid for your system.

Why is meshing needed in CFD problems?

Meshing is needed for converting complex differential equations into simpler arithmetic problems, discretization allows a simulation to account for changes in continuous physical properties across the solution domain.

H grid or O grid or C grid for airfoil Analysis:

C-grid is preferred for viscous flow because it allows you to resolve the wake.as the grid will be aligned to the wake or slipstream at the trailing edge of the airfoil.

O-grids are good enough for inviscid flow, where they have the advantage of reduced cell count as compared to C-grids. The resolution in normal direction is high only where you really need it: right on the airfoil surface. The region behind the trailing edge is not well resolved, which makes the use of O-grids questionable in case of viscous flow. You may still get good results even for viscous flow, depending on the particular case.

H grids the only reason to use H grid is its simplicity over O grid and C grid. But it doesn't Have the same accuracy as O-Grid & C-Grid.

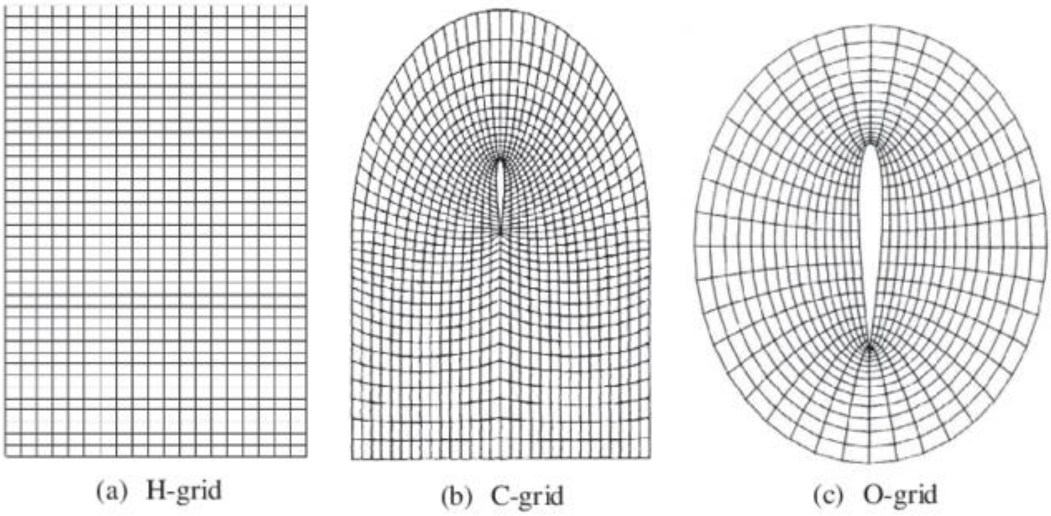


Figure 1: Different grid shapes used in CFD

We chose to use the O grid due to its simplicity and accuracy.

Parameters and constants

Joukowski Airfoil Data

Table 1: Joukowski Airfoil Data

Parameter	***Unit***	***Value***
$\boldsymbol{Chord}$	*Meter*	1
$\boldsymbol{\frac{Camber}{chord}}$	*Percentage*	4
$\boldsymbol{\frac{Thickness}{chord}}$	*Percentage*	5
$\boldsymbol{Angle\ of\ Attack}\ (\propto)$	*Degrees*	4

Grid Parameters

Table 2: Grid Parameters

Parameter	***Unit***	***Value***
$\boldsymbol{i_{max}}$	*Percentage*	4
$\boldsymbol{j_{max}}$	*Percentage*	5
$\boldsymbol{R}$ (Far-Field)	*Meter*	5
$\boldsymbol{\eta_{1_{max}}}$	*N/A*	1
$\boldsymbol{\eta_{2_{max}}}$	*N/A*	1
$\boldsymbol{\eta_{1_{min}}}$	*N/A*	0
$\boldsymbol{\eta_{2_{min}}}$	*N/A*	0

Numerical Solution Algorithm

Step 1. Determine the mesh size you will use; i_{max} and j_{max}.

Step 2. Generate the airfoil coordinates. For cord c projecting the points of a circle originated at the cord mid distance and of Diameter $= c$.

Step 3. . Transform the physical domain to the computational domain in η_1, η_2 coordinates.

Step 4. Generate the O grid with the i and j indices correspond to η_1 η_2 respectively.

Step 5. Introduce the boundary conditions.

a. at the far field boundary we assume one point that have $\psi = 0$ for example $\psi_{1,jmax}$ Calculate the values of ψ at the rest of the far field boundary

$$\Delta\psi = u\Delta y - v\Delta \mathrm{x}$$

b. Trailing edge Boundary Condition at i_1 & i_{max} will be $\psi_{1,j} = \psi_{i_{max},j}$

c. Kutta condition for $j = 1$, it states ψi,1 $= \psi k$,2 for $+ve$ camber Airfoil

Step 6. Using the PSOR scheme the transformed Laplace equation

$$\psi_{\{i,j\}}^{\{n+1\}}\left(\frac{C_{11}\left\{i+\frac{1}{2},j\right\}}{\Delta\eta_1^2}+\frac{C_{11}\left\{i-\frac{1}{2},j\right\}}{\Delta\eta_1^2}+\frac{C_{22}\left\{i,j+\frac{1}{2}\right\}}{\Delta\eta_2^2}+\frac{C_{22}\left\{i,j-\frac{1}{2}\right\}}{\Delta\eta_2^2}\right)=\frac{\psi_{\{i+1,j\}}^{\{n\}}C_{11}\left\{i+\frac{1}{2},j\right\}}{\Delta\eta_1^2}+\frac{\psi_{\{i-1,j\}}^{\{n\}}C_{11}\left\{i-\frac{1}{2},j\right\}}{\Delta\eta_1^2}$$
$$+\frac{\psi_{\{i+1,j+1\}}^{\{n\}}C_{12\{i+1,j\}}}{4\Delta\eta_1\Delta\eta_2}-\frac{\psi_{\{i+1,j-1\}}^{\{n\}}C_{12}\{i+1,j\}}{4\Delta\eta_1\Delta\eta_2}-\frac{\psi_{\{i-1,j+1\}}C_{12}\{i-1,j\}}{4\Delta\eta_1\Delta\eta_2}+\frac{\psi_{\{i-1,j-1\}}^{\{n\}}C_{12\{i-1,j\}}}{4\Delta\eta_1\Delta\eta_2}$$
$$+\frac{\psi_{\{i+1,j+1\}}^{\{n\}}C_{12}\{i,j+1\}}{4\Delta\eta_1\Delta\eta_2}-\frac{\psi_{\{i-1,j+1\}}^{\{n\}}C_{12}\{i,j+1\}}{4\Delta\eta_1\Delta\eta_2}-\frac{\psi_{\{i+1,j-1\}}^{\{n\}}C_{12}\{i,j-1\}}{4\Delta\eta_1\Delta\eta_2}+\frac{\psi_{\{i-1,j-1\}}^{\{n\}}C_{12}\{i,j-1\}}{4\Delta\eta_1\Delta\eta_2}$$
$$+\frac{\psi_{\{i,j+1\}}^{\{n\}}C_{22}\left\{i,j+\frac{1}{2}\right\}}{\Delta\eta_2^2}+\frac{\psi_{\{i,j-1\}}^{\{n\}}C_{22}\left\{i,j-\frac{1}{2}\right\}}{\Delta\eta_2^2}=0$$

Analytical Solution Algorithm

To solve the flow over the airfoil analytically the Z-Transformation method to obtain the exact solution. The following is the procedure to obtain the solution.

Step 1: From the Airfoil parameters calculate the transformation parameters using the following equations:

$$b = c/4, \qquad e = \frac{tmax/c}{1.3}, \quad \beta = 2(Cmax/c), \qquad a = \frac{\mathrm{b(1+e)}}{cos\beta},$$

$$x_0 = -b \times e, \qquad\qquad y_0 = a$$

Step 2: Generate two coordinates' vectors for a circle at the origin with $radius\ =\ a$ in Z' plane by generating an angles vector $\bar{\theta}'$ form $0\ to\ 2\pi$ with step $\Delta\theta$, Then use the following equations to get the coordinates.

$$\bar{x}' = r \times \cos(\bar{\theta}') \qquad\qquad \bar{y}' = r \times \sin(\bar{\theta}')$$

Step 3: The created circle in (Z'plane) can be described in (Z plane) as a circle shifted horizontally and vertically by the values of $x0$, $y0$.

$$\bar{x} = \bar{x}' + x_0, \quad \bar{y} = \bar{y}' + y_0, \quad \bar{r} = \sqrt{\bar{x}^2 + \bar{y}^2}, \qquad \bar{\theta} = \tan^{-1}\left(\frac{\bar{y}}{\bar{x}}\right)$$

Step 4: The shifted circle (Zplane) can be described in (Z_1 plane) as an airfoil using the following transformation:

$$\bar{x}_1 = \bar{x}\left(1 + \frac{b^2}{(\bar{x}^2+\bar{y}^2)}\right), \qquad \bar{y}_1 = \bar{y}\left(1 - \frac{b^2}{(\bar{x}^2+\bar{y}^2)}\right), \qquad \bar{r}_1 = \sqrt{\bar{x}_1^2 + \bar{y}_1^2}, \qquad \bar{\theta}_1 = \tan^{-1}\left(\frac{\bar{y}_1}{\bar{x}_1}\right)$$

Step 5: the velocity and pressure coefficient distribution can be calculated as follows.

$$\bar{V}_{r'} = V_\infty\left(1 - \frac{a^2}{r'^2}\right)\cos\left(\bar{\theta}' - \alpha\right)$$

$$\bar{V}_{\theta'} = -V_\infty\left[\sin\left(\bar{\theta}' - \alpha\right)\left(1 + \frac{a^2}{r'^2}\right) + 2\left(\frac{a}{r'}\right)\sin\left(\alpha + \beta\right)\right]$$

$$\bar{V}_1 = \sqrt{\frac{\bar{V}_{\theta'}}{1 - \frac{2b^2}{r^2}\cos\left(2\bar{\theta}\right) + \frac{b^4}{r^4}}}$$

$$\bar{C}_p = 1 - \left(\frac{\bar{V}^2}{V_\infty^2}\right)$$

Results

O-Grid

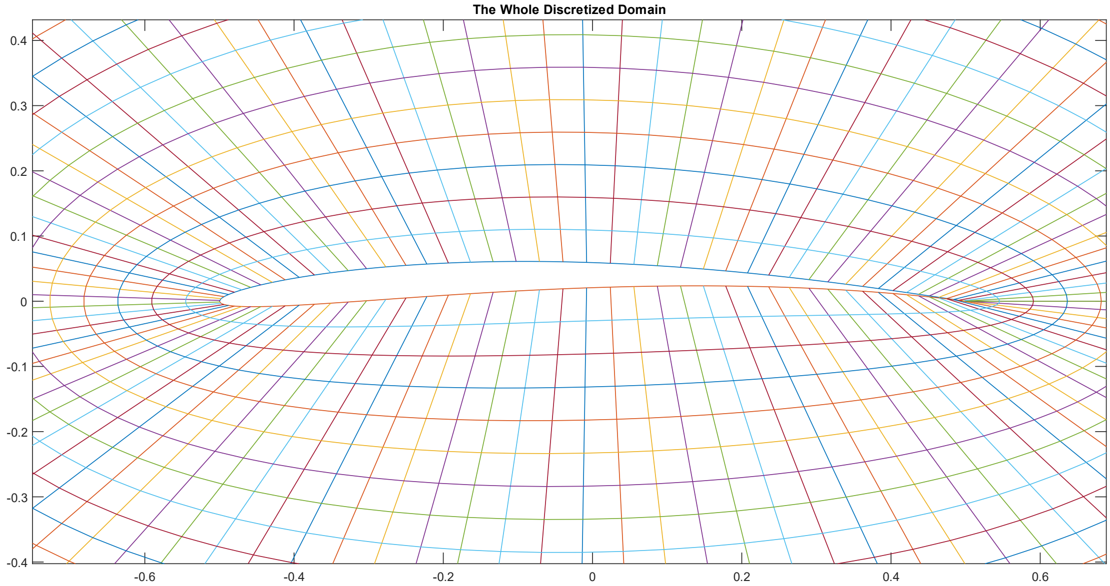


Figure 2 Zoomed in Figure for the grid around the arifoil

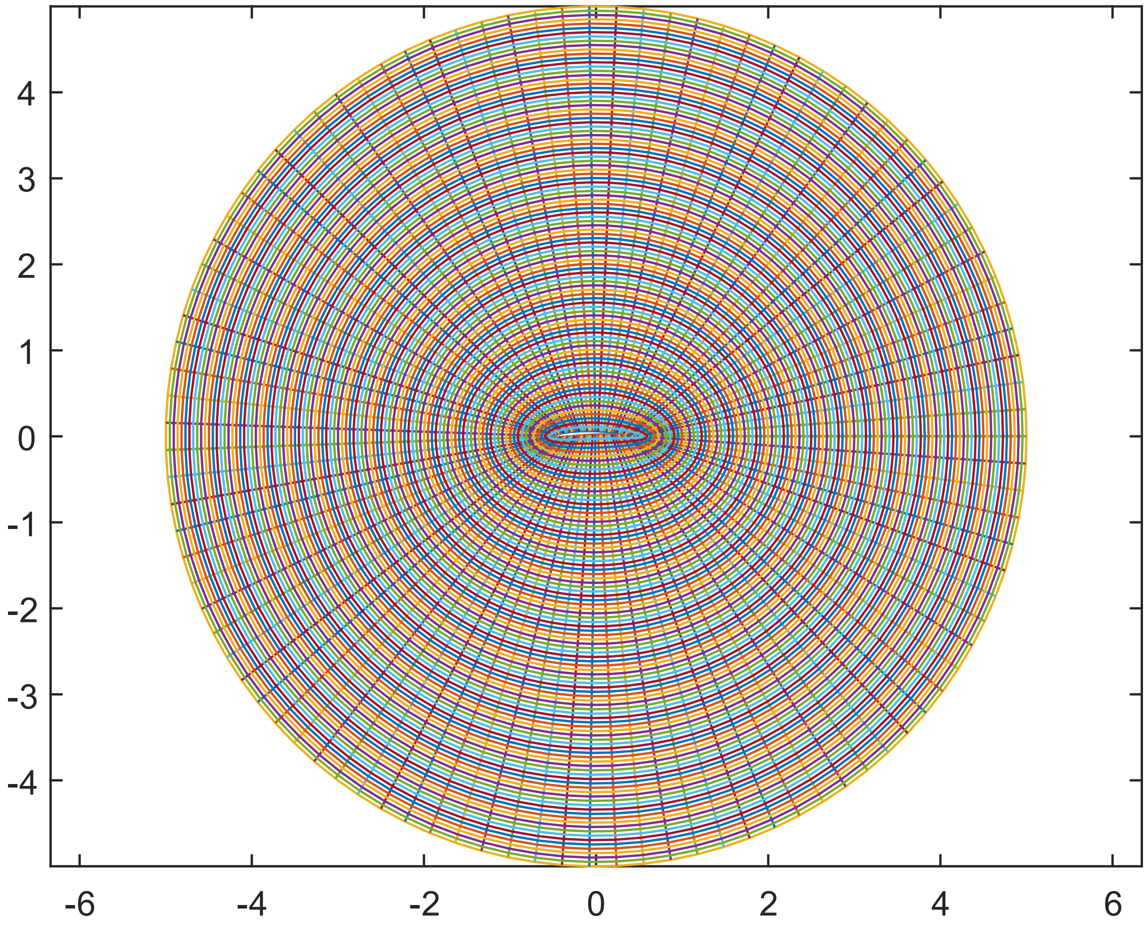
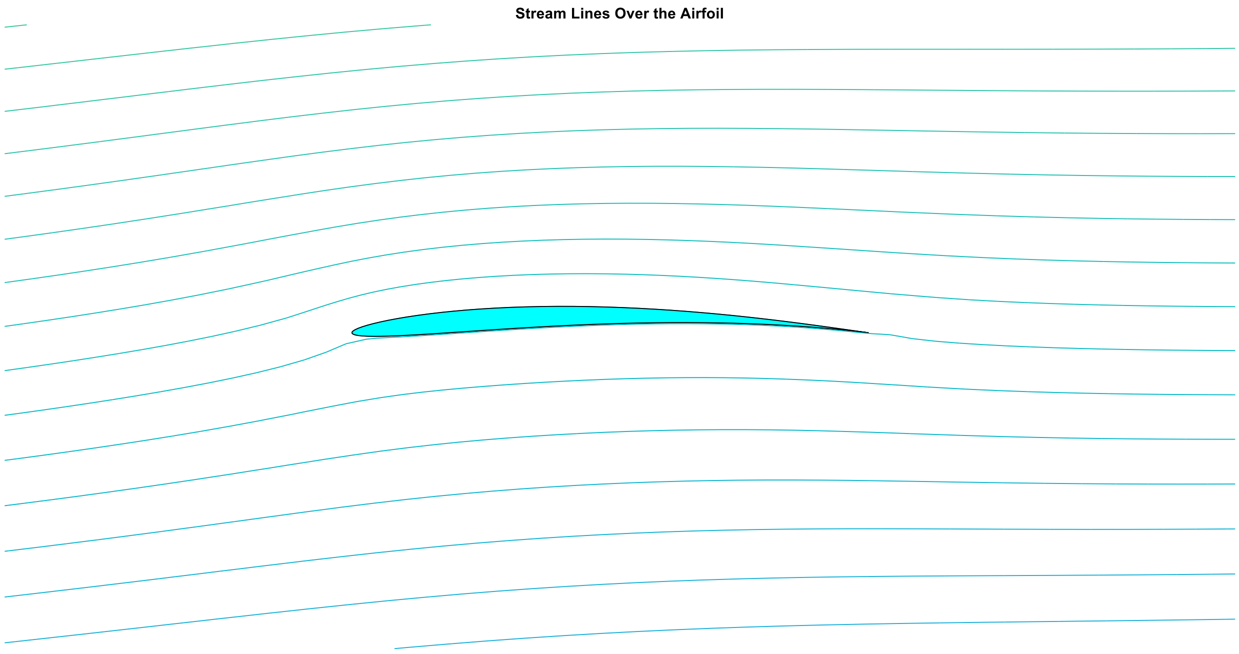


Figure 3: O-grid

Streamlines around the Airfoil



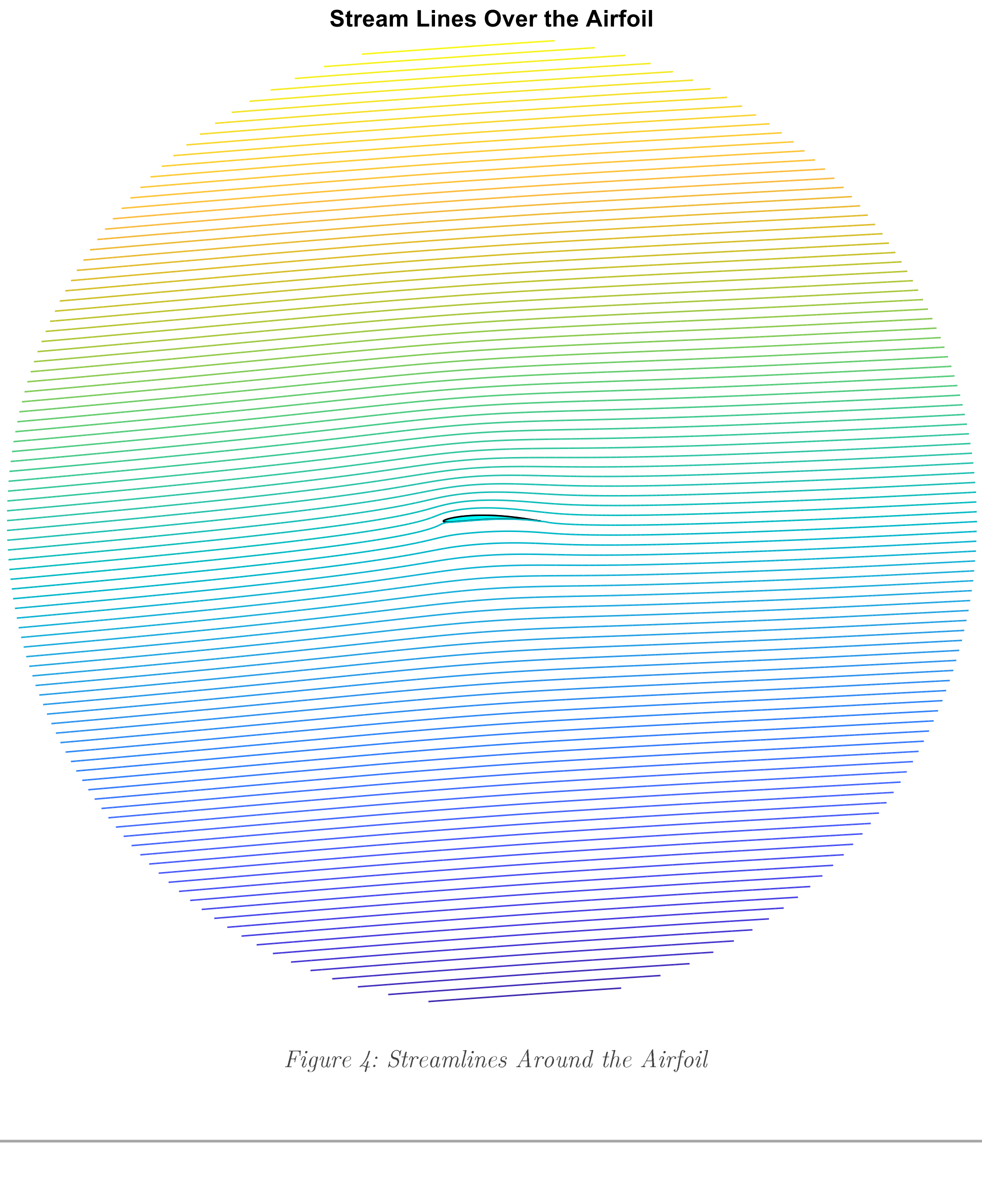


Figure 4: Streamlines Around the Airfoil

Velocity and C_p distribution over the Airfoil

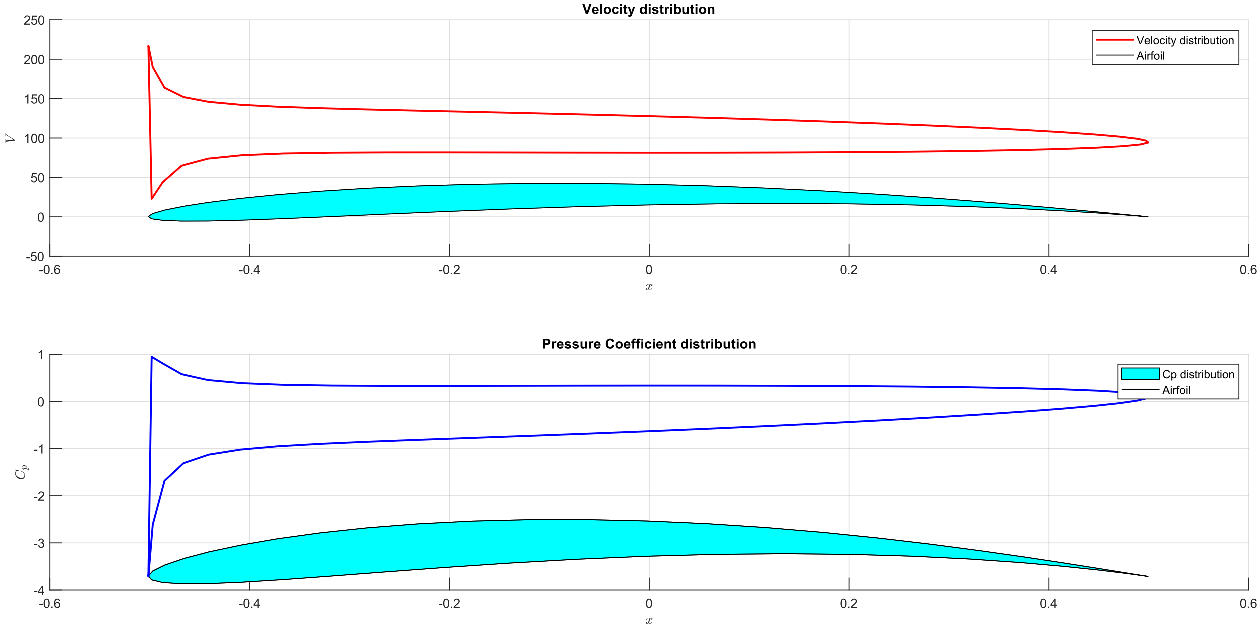


Figure 5: Analytical Solution of Velocity and C_p distribution over the airfoil

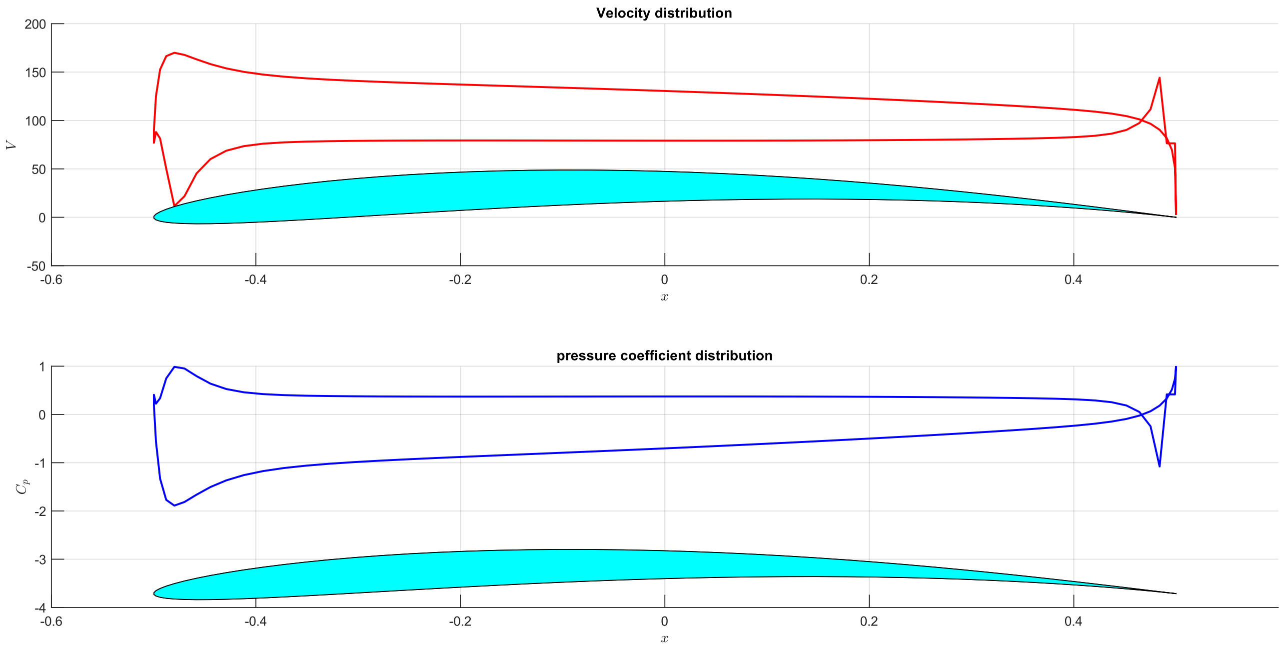


Figure 6: Numerical Solution of Velocity and C_p distribution over the airfoil

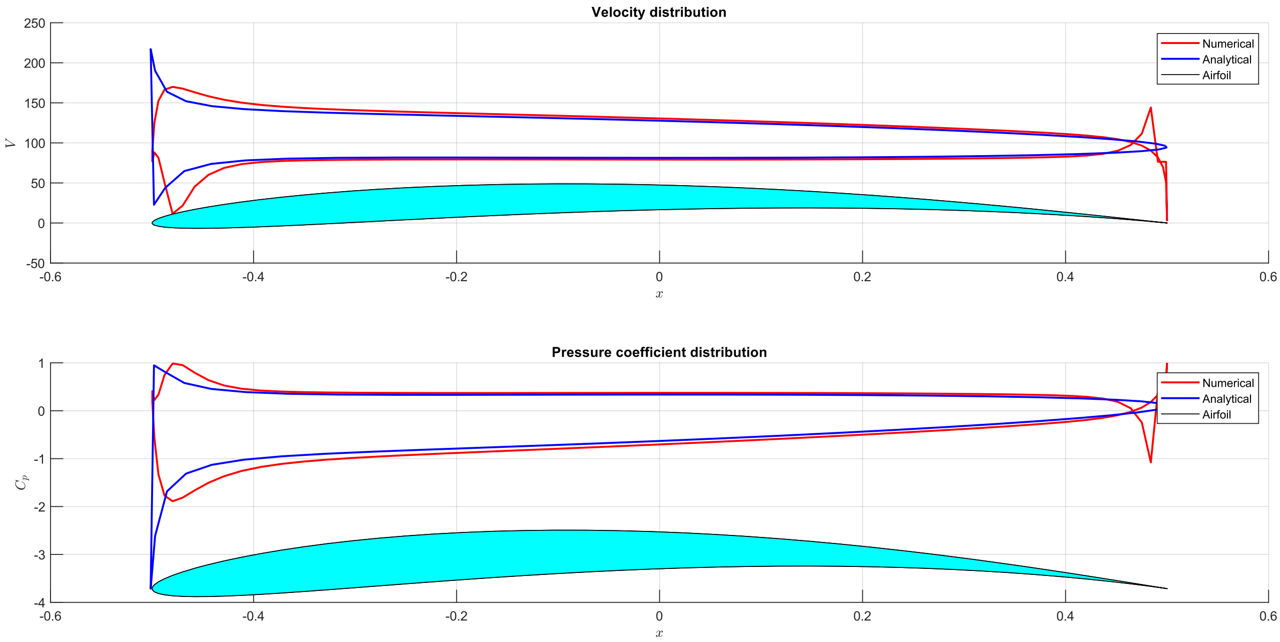


Figure 7: Comparison between the analytical and Numerical Solution of Velocity and C_p distribution over the airfoil

Velocity and Pressure Contours

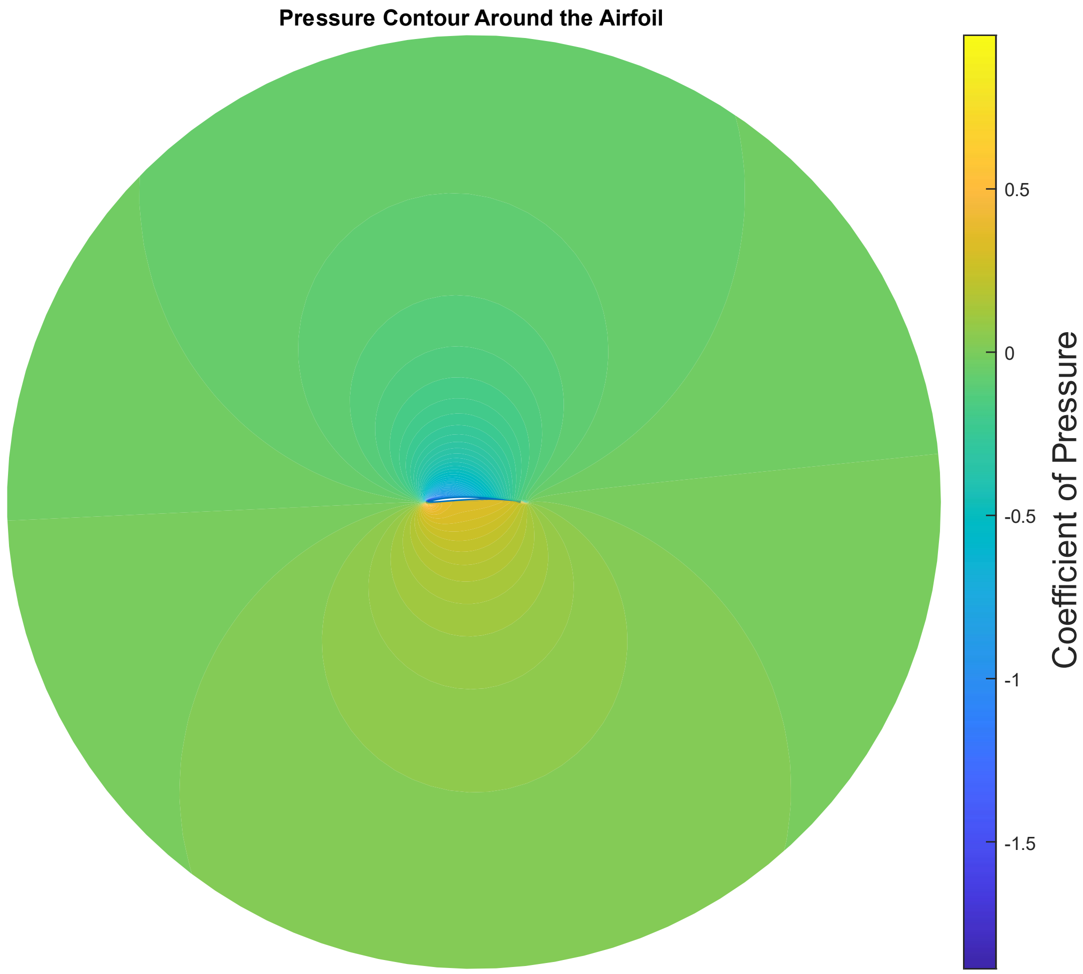


Figure 8: Pressure Contour around the Airfoil

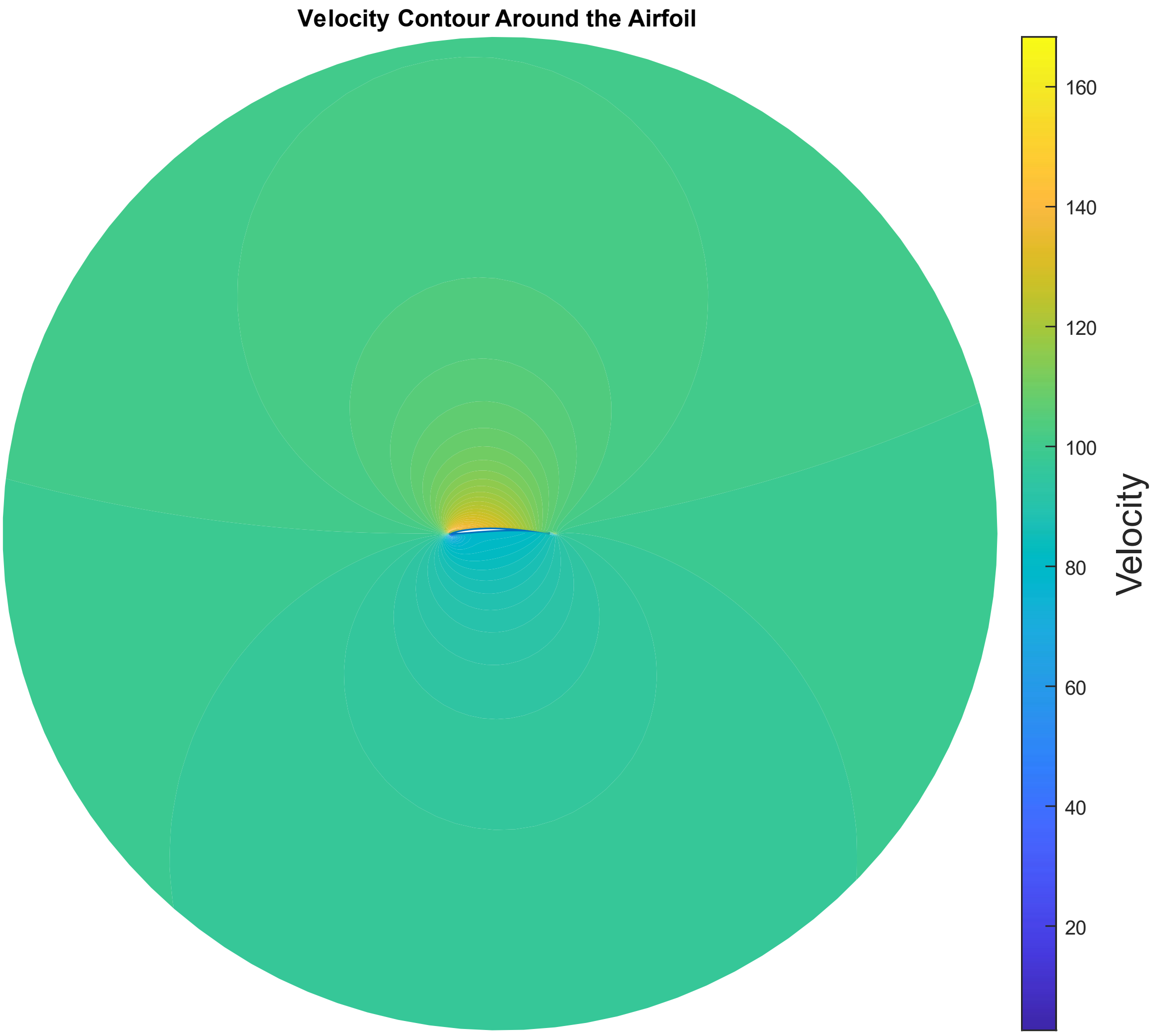


Figure 9: Velocity Contour around the airfoil

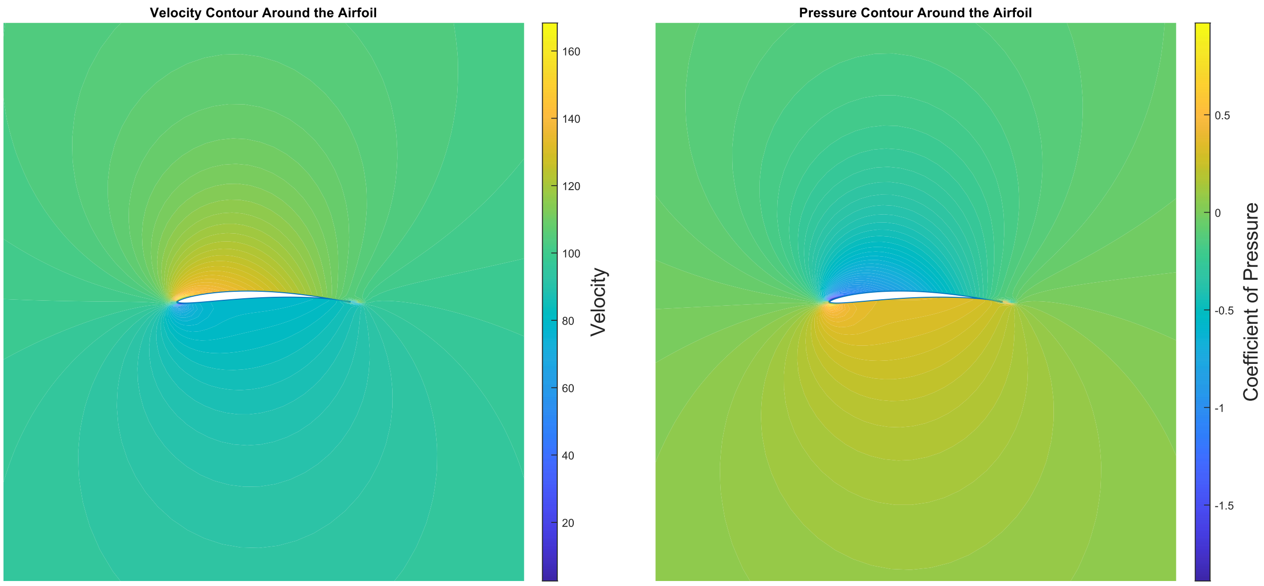


Figure 10: Closeup figure of pressure and velocity contours around the Airfoil

Conclusion

The numerical results show promising results, not as accurate as the analytical potential flow method, but provides decent results when using a finite difference approach.

Appendix A: Main script

```
% O grid of airfoil using transformations
% Project made by Ahmed Mohamed Hassan Abdulrahman

clc
clearvars
close all
```

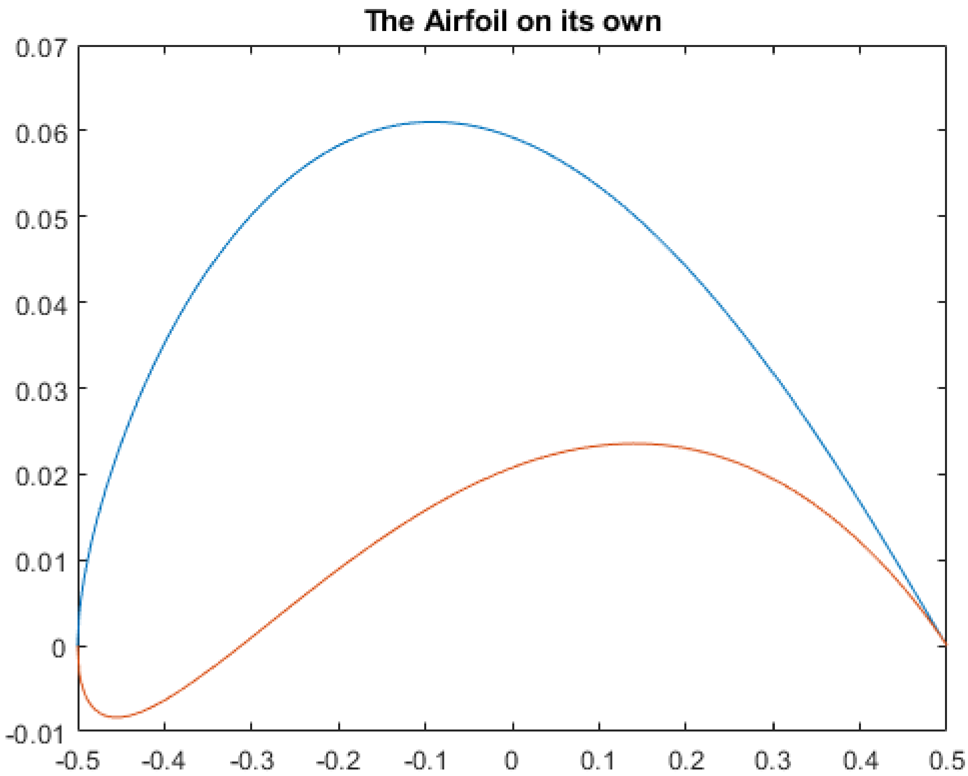
Initialization (Inputs)

```
% The Free Stream Velocity

Vinf=100;

% Choose the maximum mesh size:

i_max=100; j_max=100;

% Enter the Joukowski Airfoil Parameters

c=1;                    % Chord
C_max_c=0.04;           % Maximum Camber/Chord Percentage
t_max_c=0.05;            % Maximum Thickness/Chord Percentage
AoA=4*pi/180;           % Angle of Attack of flow Percentage

% Drawing Parameters

airfoil_points=500;     % Number of points on the airfoil; More points = Better accuracy
R=5*c;                  % Far field radius assumption
Contour_Detail=100;     % How fine the contour plots would be

% The Transformation Parameters

eta1_max=1;     eta1_min=0;
eta2_max=1;     eta2_min=0;

% Solution Limits

RMS_limit=1e-5;
```

Generating Airfoil Coordinates

```
% Joukowski airfoil parameters calculations

b=c/4;
r=2*b;                  % Radius of the circle
e=t_max_c/1.3;          % eccentricity of the circle
beta=2*C_max_c;
a=b*(1+e)/cos(beta);

% x coordinates of the points on the upper and lower surfaces of the airfoil:
x_airfoil_coord=linspace(-r,r,airfoil_points);
```

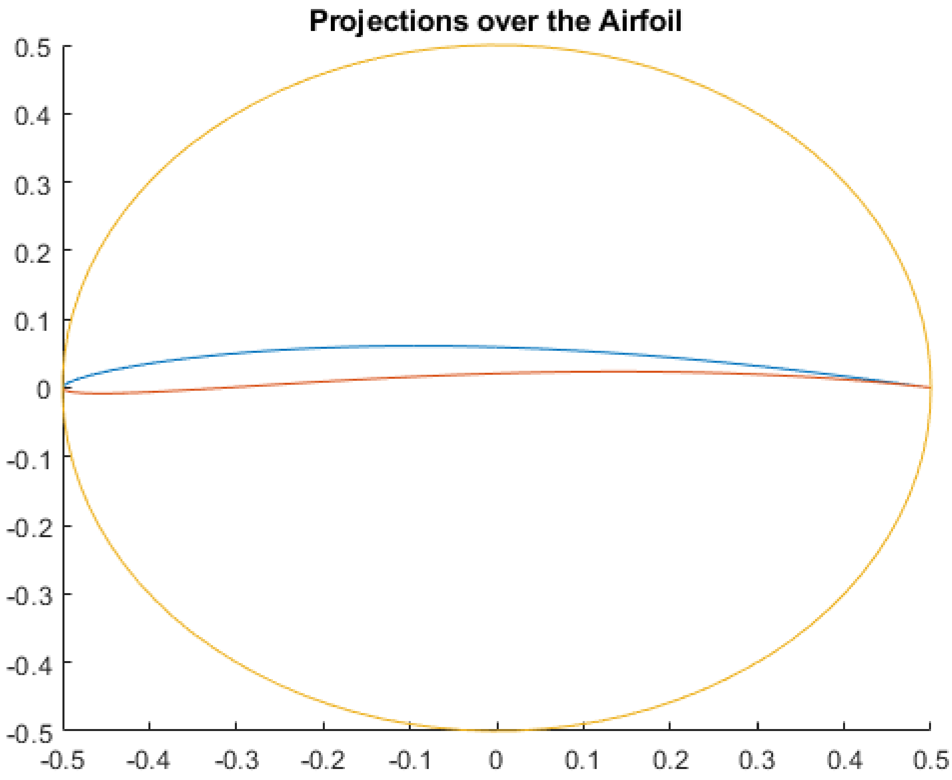
```
% y coordinates equation on the upper surface of the airfoil:
y_upper= @(x) 2*b*e*(1-x/2/b).*(sqrt(1-(x/2/b).^2))+2*b*beta*(1-(x/2/b).^2);
% y coordinates equation on the lower surface of the airfoil:
y_lower= @(x) 2*b*e*(1-x/2/b).*(-sqrt(1-(x/2/b).^2))+2*b*beta*(1-(x/2/b).^2);

% Plotting the Airfoil on its own

figure('Name','The Airfoil on its own')
plot(x_airfoil_coord,y_upper(x_airfoil_coord),x_airfoil_coord,y_lower(x_airfoil_coord))
title('The Airfoil on its own')

% plotting the Airfoil along with the discretizing circles
figure('Name','Projections over the Airfoil')
hold on
plot(x_airfoil_coord,y_upper(x_airfoil_coord),x_airfoil_coord,y_lower(x_airfoil_coord))
title('Projections over the Airfoil')
```



Generating Circle around airfoil

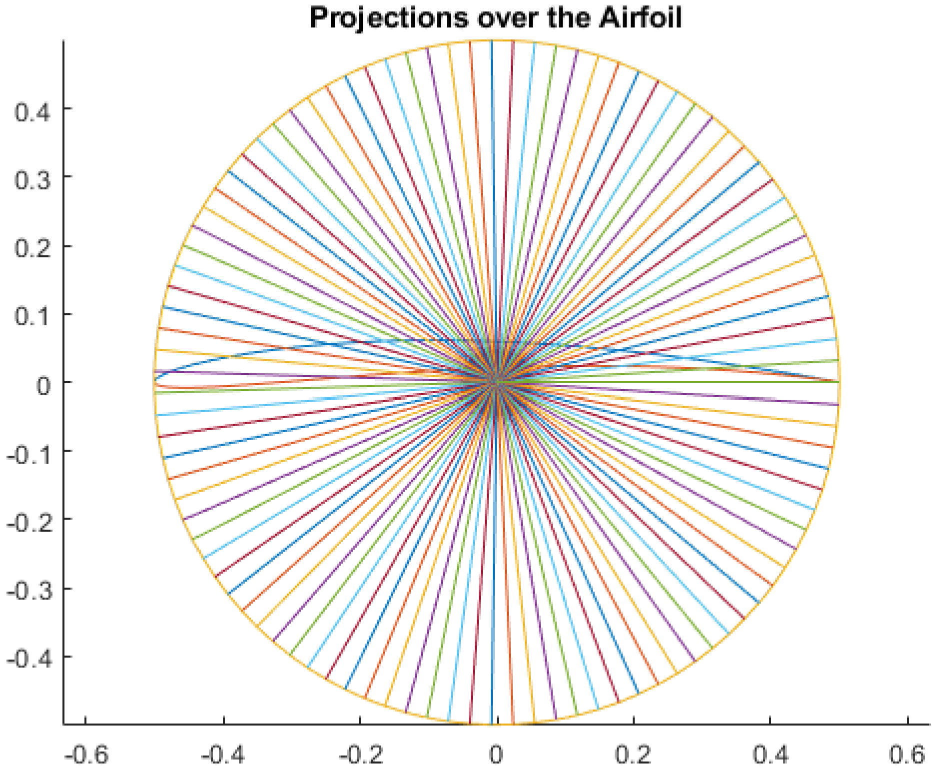
```
 % We must divide the circumference of the circle by the number of
 % solving points (in i)
Delta_theta=2*pi/(i_max-1);
 % to plot a circle we need points in x and y, this can be
 % achieved by using x=r*cos(theta) and y=r*sin(theta)

x_circle_plot=zeros(1,i_max);
y_circle_plot=zeros(1,i_max);
```

```
for i=1:i_max
    x_circle_plot(i)=r*cos(Delta_theta*(i-1));  % x coordinates of circle
    y_circle_plot(i)=r*sin(Delta_theta*(i-1));  % y coordinates of circle
end

 % Plotting the plain circle
plot(x_circle_plot,y_circle_plot)

hold on
```



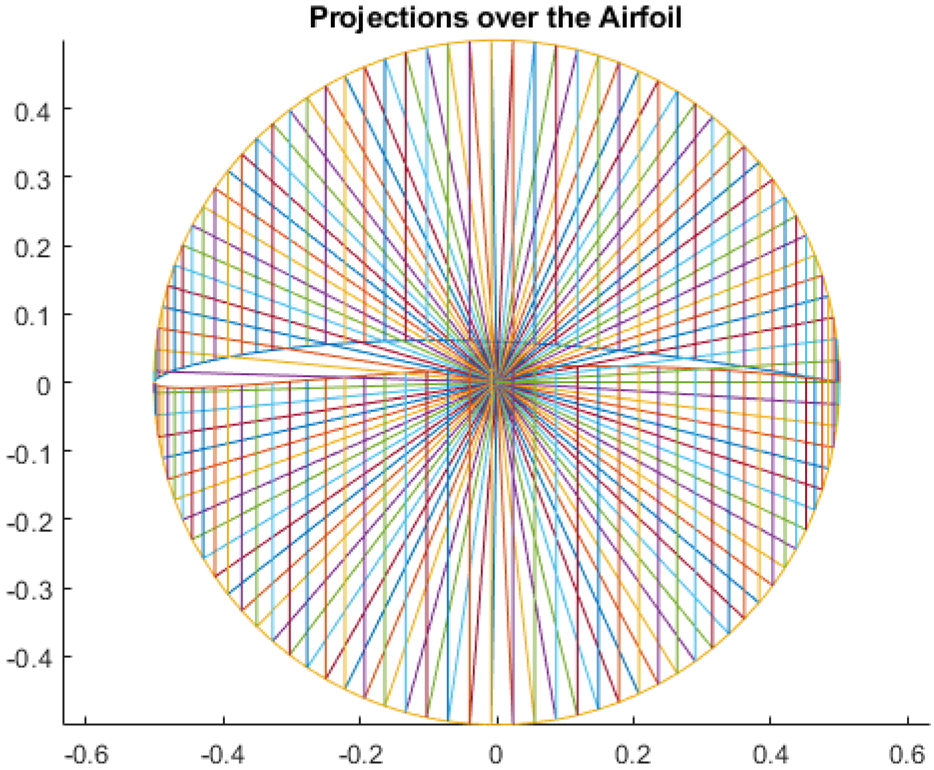
Plotting $\boldsymbol{i_{max}}$ lines from the center to the circle of the airfoil

```
for i=1:i_max
    plot([0 x_circle_plot(i)],[0 y_circle_plot(i)])
end
hold on
axis equal
```



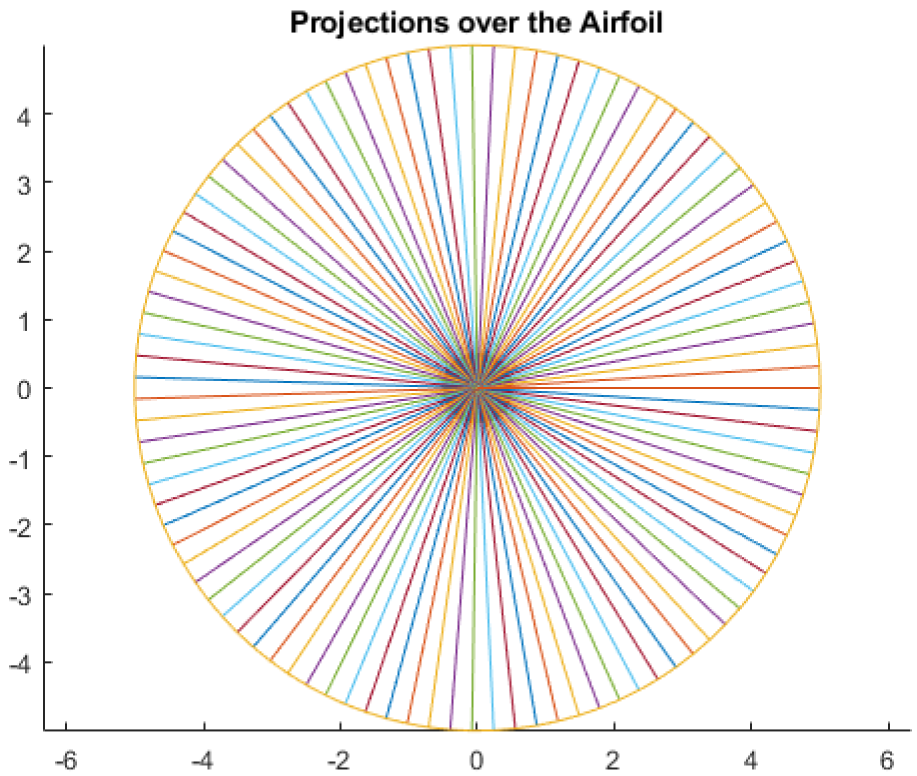
Projecting the intersection points

```
airfoil_proj=zeros(1,i_max);

for i=1:i_max
    if i<=i_max/2
        airfoil_proj(i)=y_upper(r*cos(Delta_theta*(i-1)));  % Projected coordinate
        plot([x_circle_plot(i) x_circle_plot(i)],[y_circle_plot(i) airfoil_proj(i)])
    elseif i>=i_max/2
        airfoil_proj(i)=y_lower(r*cos(Delta_theta*(i-1)));  % Projected coordinate
        plot([x_circle_plot(i) x_circle_plot(i)],[y_circle_plot(i) airfoil_proj(i)])
    end
end
```



Plotting Far field and its lines

```
x_circleR_plot=zeros(1,i_max);
y_circleR_plot=zeros(1,i_max);

for i=1:i_max
    x_circleR_plot(i)=R*cos(Delta_theta*(i-1));  % x coordinates of farfield circle
    y_circleR_plot(i)=R*sin(Delta_theta*(i-1));  % y coordinates of farfield circle

    plot([0 x_circleR_plot(i)],[0 y_circleR_plot(i)])
end

 % Plotting the far field circle
plot(x_circleR_plot,y_circleR_plot)
hold on
```



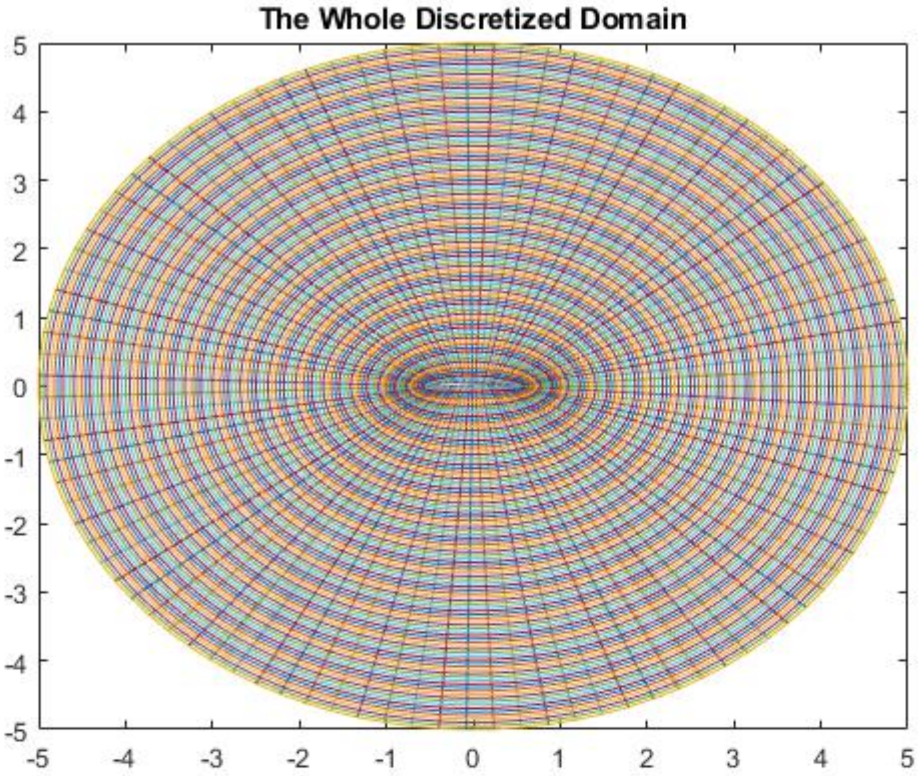
Discretizing the Domain

```
% Discretizing the domain in delta x and delta y
Delta_x=(x_circleR_plot-x_circle_plot)/(j_max-1);   % The discretized change in the x direction
Delta_y=(y_circleR_plot-airfoil_proj)/(j_max-1);    % The discretized change in the y direction
```

Plotting the whole discretized domain

```
figure('Name','The Whole Discretized Domain')
plot(x_airfoil_coord,y_upper(x_airfoil_coord),x_airfoil_coord,y_lower(x_airfoil_coord))
title('The Whole Discretized Domain')
hold on

plot(x_circleR_plot,y_circleR_plot)
for i=1:i_max
    plot([x_circle_plot(i) x_circleR_plot(i)],[airfoil_proj(i) y_circleR_plot(i)])
end

% initialiazing variables used in plotting
x_coords=zeros(j_max,i_max);
y_coords=zeros(j_max,i_max);

% Setting the coordinates of the airfoil and the farfield into the
% variables

x_coords(1,:)=x_circle_plot;
y_coords(1,:)=airfoil_proj;
x_coords(j_max,:)=x_circleR_plot;
y_coords(j_max,:)=y_circleR_plot;
```

```
% plotting the lines

for j=2:j_max-1
    for i=1:i_max
        x_coords(j,i)=x_circle_plot(i)+Delta_x(i)*(j-1);
        y_coords(j,i)=airfoil_proj(i)+Delta_y(i)*(j-1);
    end
    plot(x_coords(j,:),y_coords(j,:))
end
```



Transforming into the computational domain

```
Delta_eta1=(eta1_max-eta1_min)/(i_max-1);
Delta_eta2=(eta2_max-eta2_min)/(j_max-1);

% initializing the eta domain

eta1_coords=zeros(j_max,i_max);
eta2_coords=zeros(j_max,i_max);

% Getting the eta1 and eta2 coordinates

for j=1:j_max
    for i=1:i_max
        eta1_coords(j,i)=Delta_eta1*(i-1);
        eta2_coords(j,i)=Delta_eta2*(j-1);
    end
end
```

```matlab
% Calculating computational domain derivatives

% Initialization
x_eta1=zeros(j_max,i_max); y_eta1=zeros(j_max,i_max);
x_eta2=zeros(j_max,i_max); y_eta2=zeros(j_max,i_max);

x_eta1(:,1)=(-1*x_coords(:,i_max-1)+x_coords(:,2))./(2*Delta_eta1);
x_eta2(1,:)=(-3*x_coords(1,:)+4*x_coords(2,:)-x_coords(3,:))./(2*Delta_eta2);
y_eta1(:,1)=(-1*y_coords(:,i_max-1)+y_coords(:,2))./(2*Delta_eta1);
y_eta2(1,:)=(-3*y_coords(1,:)+4*y_coords(2,:)-y_coords(3,:))./(2*Delta_eta2);

x_eta1(:,i_max)=(-1*x_coords(:,i_max-1)+x_coords(:,2))./(2*Delta_eta1);
x_eta2(j_max,:)=(3*x_coords(j_max,:)-4*x_coords(j_max-1,:)+x_coords(j_max-2,:))./(2*Delta_eta2);
y_eta1(:,i_max)=(-1*y_coords(:,i_max-1)+y_coords(:,2))./(2*Delta_eta1);
y_eta2(j_max,:)=(3*y_coords(j_max,:)-4*y_coords(j_max-1,:)+y_coords(j_max-2,:))./(2*Delta_eta2);

for i=2:i_max-1
    x_eta1(:,i)=(-1*x_coords(:,i-1)+x_coords(:,i+1))/(2*Delta_eta1);
    y_eta1(:,i)=(-1*y_coords(:,i-1)+y_coords(:,i+1))/(2*Delta_eta1);
end

for j=2:j_max-1
    x_eta2(j,:)=(-1*x_coords(j-1,:)+x_coords(j+1,:))/(2*Delta_eta2);
    y_eta2(j,:)=(-1*y_coords(j-1,:)+y_coords(j+1,:))/(2*Delta_eta2);
end

J=x_eta1.*y_eta2-x_eta2.*y_eta1;
c11=(x_eta2.^2+y_eta2.^2)./J;
c12=-1*(x_eta1.*x_eta2+y_eta1.*y_eta2)./J;
c22=(x_eta1.^2+y_eta1.^2)./J;
```

Calculating psi at the zero condition

```matlab
% Calculating velocity components

uinf=Vinf*cos(AoA);
vinf=Vinf*sin(AoA);

% Initialization

psi=zeros(j_max,i_max);
Delta_x_farfield=zeros(1,i_max-1);
Delta_y_farfield=zeros(1,i_max-1);
Delta_psi=zeros(1,i_max-1);

% Calculation of the zero iteration and boundary condition iteration

% assuming that psi is zero on the airfoil
for i=2:i_max
    Delta_x_farfield(i-1)=x_circleR_plot(i)-x_circleR_plot(i-1);
    Delta_y_farfield(i-1)=y_circleR_plot(i)-y_circleR_plot(i-1);
    Delta_psi(i-1)=uinf*Delta_y_farfield(i-1)-vinf*Delta_x_farfield(i-1);
    psi(j_max,i)=psi(j_max,i-1)+Delta_psi(i-1);
    psi(:,i)=linspace(psi(1,i),psi(j_max,i),j_max);
end
```

```
psi(:,1)=psi(:,i_max);
if C_max_c==0
    psi(1,:)=psi(2,1);
elseif C_max_c>0
    psi(1,:)=psi(2,i_max-ceil(i_max*C_max_c/2));
elseif C_max_c<0
    psi(1,:)=psi(2,1+floor(i_max*C_max_c/2));
end
```

Calculating the half coefficients used in iterations

```
%iph= i+0.5     %jph= j+0.5
%imh= i-0.5     %jmh= j-0.5
c11_ih=zeros(j_max,i_max);
c22_jh=zeros(j_max,i_max);

for i=1:i_max-1
    c11_ih(:,i+1)=(c11(:,i)+c11(:,i+1))/2; % from i=1.5 to i=i_max-0.5
end
c11_ih(:,1)=c11_ih(:,i_max); % value at i=0.5 is the same at i=i_max-0.5

% since our iterations will start from j=2
for j=1:j_max-1
    c22_jh(j,:)=(c22(j,:)+c22(j+1,:))/2; % from j=1.5 to j=j_max-0.5
end
```

Coefficients used in iterations

```
for j=2:j_max-1
    for i=1:i_max-1
        S_i_j(j-1,i)=(c11_ih(j,i+1)+c11_ih(j,i))/Delta_eta1^2 + ...
            (c22_jh(j,i)+c22_jh(j-1,i))/Delta_eta2^2;

        S_ip1_j(j-1,i)=c11_ih(j,i+1)/Delta_eta1^2;
        S_im1_j(j-1,i)=c11_ih(j,i)/Delta_eta1^2;

        S_i_jp1(j-1,i)=c22_jh(j,i)/Delta_eta2^2;
        S_i_jm1(j-1,i)=c22_jh(j-1,i)/Delta_eta2^2;

        S_ip1_jp1(j-1,i)=(c12(j,i+1)+c12(j+1,i))/(4*Delta_eta1*Delta_eta2);
        S_ip1_jm1(j-1,i)=(-c12(j,i+1)-c12(j-1,i))/(4*Delta_eta1*Delta_eta2);
        if i==1
            S_im1_jp1(j-1,i)=(-c12(j,i_max-1)-c12(j+1,i))/(4*Delta_eta1*Delta_eta2);
            S_im1_jm1(j-1,i)=(c12(j,i_max-1)+c12(j-1,i))/(4*Delta_eta1*Delta_eta2);
        else
        S_im1_jp1(j-1,i)=(-c12(j,i-1)-c12(j+1,i))/(4*Delta_eta1*Delta_eta2);
        S_im1_jm1(j-1,i)=(c12(j,i-1)+c12(j-1,i))/(4*Delta_eta1*Delta_eta2);
        end
    end
end
```

Iterations

```
iteration_No=0;
psi_intitial=psi;
psi_iteration=psi_intitial;

RMS=RMS_limit*1e5;

while RMS>RMS_limit
    for j=2:j_max-1
        for i=1:i_max-1
            if i==1
                psi(j,i)=(psi_iteration(j,i+1)*S_ip1_j(j-1,i)+psi_iteration(j,i_max-1)*S_im1_j(j-
1,i)+...
                psi_iteration(j+1,i+1)*S_ip1_jp1(j-1,i)+psi_iteration(j-1,i+1)*S_ip1_jm1(j-1,i)+...
                psi_iteration(j+1,i_max-1)*S_im1_jp1(j-1,i)+psi_iteration(j-1,i_max-1)*S_im1_jm1(j-
1,i)+...
                psi_iteration(j+1,i)*S_i_jp1(j-1,i)+psi_iteration(j-1,i)*S_i_jm1(j-1,i))/S_i_j(j-
1,i);
            else
                psi(j,i)=(psi_iteration(j,i+1)*S_ip1_j(j-1,i)+psi_iteration(j,i-1)*S_im1_j(j-1,i)+...
                psi_iteration(j+1,i+1)*S_ip1_jp1(j-1,i)+psi_iteration(j-1,i+1)*S_ip1_jm1(j-1,i)+...
                psi_iteration(j+1,i-1)*S_im1_jp1(j-1,i)+psi_iteration(j-1,i-1)*S_im1_jm1(j-1,i)+...
                psi_iteration(j+1,i)*S_i_jp1(j-1,i)+psi_iteration(j-1,i)*S_i_jm1(j-1,i))./S_i_j(j-
1,i);
            end
        end
    end
    psi(:,i_max)=psi(:,1);

    if C_max_c==0
        psi(1,:)=psi(2,1);
    elseif C_max_c>0
        psi(1,:)=psi(2,i_max-1);
    elseif C_max_c<0
        psi(1,:)=psi(2,1+floor(i_max*C_max_c/2));
    end

    RMS=sqrt(sum(sum((psi-psi_iteration).^2))/((i_max-1)*(j_max-1)));

    psi_iteration=psi;

    iteration_No=iteration_No+1;
end
```

Velocity Calculation

```
% since at i=1 and i=i_max is the same point
% we calculate psi using central difference

psi_eta1(:,1)=(psi(:,2)-psi(:,i_max-1))/(2*Delta_eta1);
psi_eta1(:,i_max)=psi_eta1(:,1);

% calculating at j=1 using forward difference
psi_eta2(1,:)=(-3*psi(1,:)+4*psi(2,:)-1*psi(3,:))/(2*Delta_eta2);
```

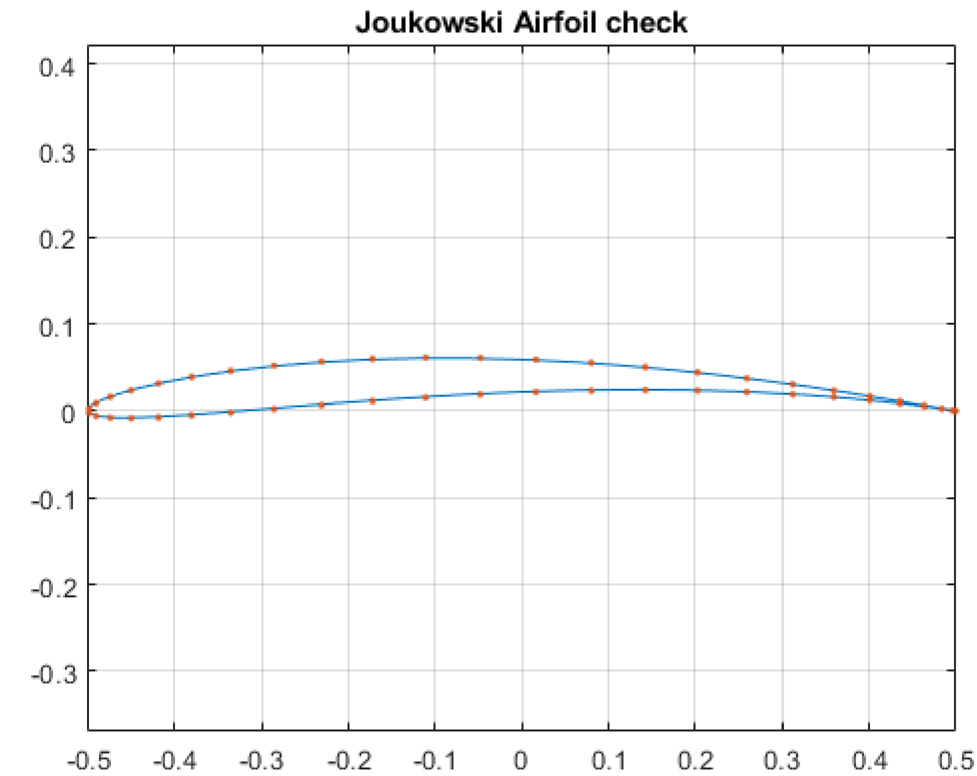
```
% calculating at j=j_max using backward difference
psi_eta2(j_max,:)=(3*psi(j_max,:)-4*psi(j_max-1,:)+psi(j_max-2,:))/(2*Delta_eta2);

for j=2:j_max-1
    for i=1:i_max
        psi_eta2(j,i)=(psi(j+1,i)-psi(j-1,i))/(2*Delta_eta2);
    end
end
for j=2:j_max
    for i=2:i_max-1
        psi_eta1(j,i)=(psi(j,i+1)-psi(j,i-1))/(2*Delta_eta1);
    end
end

eta1_x=y_eta2./J;
eta1_y=-x_eta2./J;
eta2_x=-y_eta1./J;
eta2_y=x_eta1./J;

u=psi_eta1.*eta1_y+psi_eta2.*eta2_y;
v=-psi_eta1.*eta1_x-psi_eta2.*eta2_x;
```

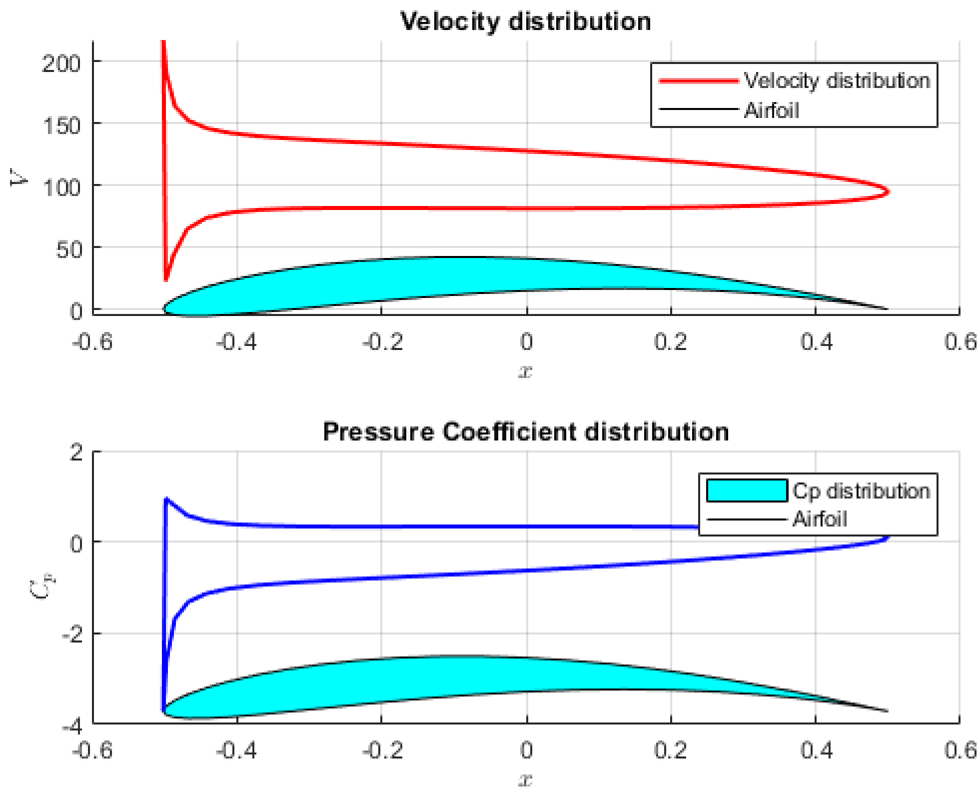
Cp Calculation

```
V=sqrt(u.^2+v.^2);
[j_ind,i_ind]=find(V>=3*mean(mean(V)));
V(j_ind,i_ind)=(V(j_ind,i_ind-1)+V(j_ind,i_ind+1))/2;
V(j_ind:j_ind,i_ind-1:i_ind+1)=linspace(V(j_ind,i_ind-1),V(j_ind,i_ind+1),length(i_ind));

Cp=1-(V/Vinf).^2;
```

Analytical solution using Joukowski Airfoil

```
[V_analytical, Cp_analytical, x_coords_analytical]=Joukowski(Vinf,AoA,c,C_max_c,t_max_c,i_max/2);
```





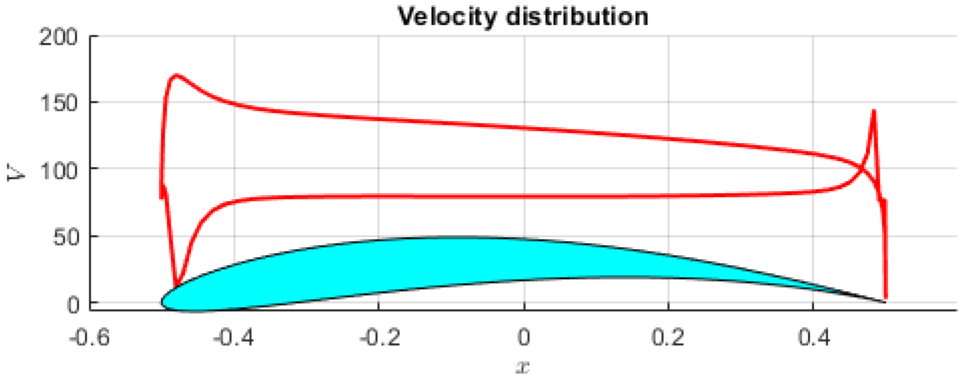
Results Graphs and plots

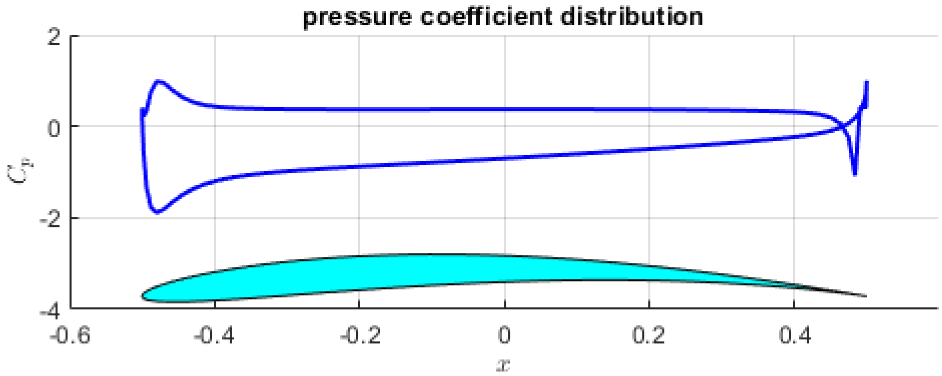
```
% Numerical pressure and velocity distribution over the Airfoil
```

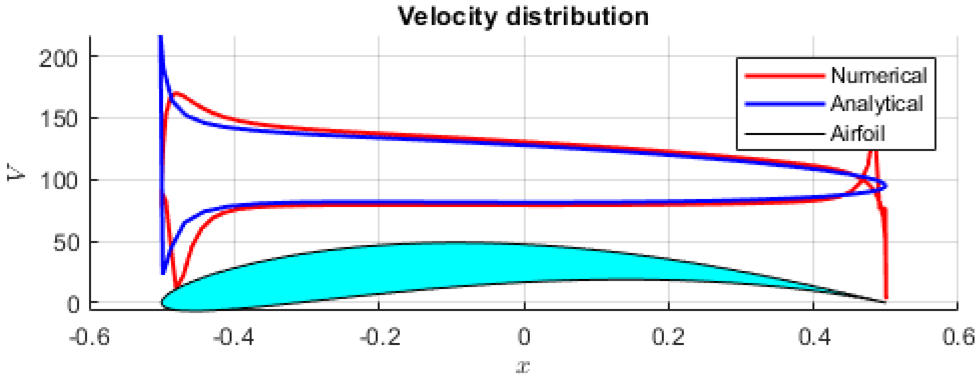
```matlab
figure('Name', 'Numerical Pressure and Velocity Distribution over the Airfoil')
     tiledlayout(2,1);
        nexttile
        hold on
            plot(x_circle_plot,V(1,:),'-' ,'LineWidth',1.5,'color','red')
            plot(x_circle_plot,800*airfoil_proj,'LineWidth',0.5,'color','black')
            fill(x_circle_plot,800*airfoil_proj,'cyan')
            grid on
            xlabel('$x$', 'interpreter', 'latex')
            ylabel('$V$', 'interpreter', 'latex')
            title('Velocity distribution', 'FontName','lm roman 9')
            xlim([-0.6 0.6])
        nexttile
        hold on
            plot(x_circle_plot,Cp(1,:),'-' ,'LineWidth',1.5,'color','blue')
            plot(x_circle_plot,15*airfoil_proj+min(Cp_analytical),'LineWidth',0.5,'color','black')
            fill(x_circle_plot,15*airfoil_proj+min(Cp_analytical),'cyan')
            grid on
            xlabel('$x$', 'interpreter', 'latex')
            ylabel('$C_p$', 'interpreter', 'latex')
            title('pressure coefficient distribution', 'FontName','lm roman 9')
            xlim([-0.6 0.6])

% Comparison Between the Joukowski Analytical and Numerical Solution

figure('Name', 'Comparison Between the Joukowski Analytical and Numerical Solution')
    tiledlayout(2,1);
        nexttile
            hold on
            % velocity results
            plot(x_circle_plot,V(1,:),'-' ,'LineWidth',1.5,'color','red')
            plot(x_coords_analytical,V_analytical,'-' ,'LineWidth',1.5,'color','blue')
            % the Airfoil
            plot(x_circle_plot,800*airfoil_proj,'LineWidth',0.5,'color','black')
            fill(x_circle_plot,800*airfoil_proj,'cyan')
            % figure settings
            grid on
            xlabel('$x$', 'interpreter', 'latex')
            ylabel('$V$', 'interpreter', 'latex')
            legend('Numerical','Analytical','Airfoil')
            title('Velocity distribution', 'FontName','lm roman 9')
        nexttile
            hold on
            % velocity results
            plot(x_circle_plot,Cp(1,:),'-' ,'LineWidth',1.5,'color','red')
            plot(x_coords_analytical,Cp_analytical,'-' ,'LineWidth',1.5,'color','blue')
            % the Airfoil
            plot(x_circle_plot,20*airfoil_proj+min(Cp_analytical),'LineWidth',0.5,'color','black')
            fill(x_circle_plot,20*airfoil_proj+min(Cp_analytical),'cyan')
            % figure settings
            grid on
            xlabel('$x$', 'interpreter', 'latex')
            ylabel('$C_p$', 'interpreter', 'latex')
            legend('Numerical','Analytical','Airfoil')
            title('Pressure coefficient distribution', 'FontName','lm roman 9')

% Stream Lines Over the Airfoil
%Zoomed out
```

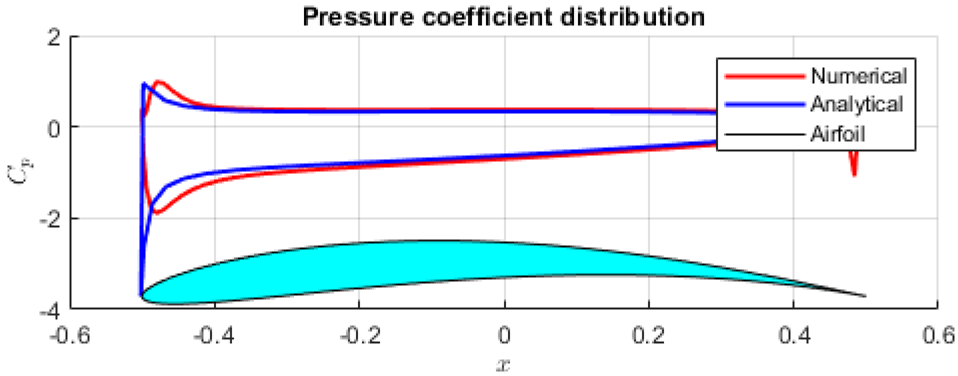
```
figure('Name', 'Streamlines Over the Airfoil')
        hold on
            plot(x_circle_plot,airfoil_proj)
            fill(x_circle_plot,airfoil_proj,'cyan')
            contour(x_coords,y_coords,psi,linspace(min(min(psi)),max(max(psi)),Contour_Detail));
            axis equal
            axis off
            title('Stream Lines Over the Airfoil', 'FontName','lm roman 12')
%Zoomed in
figure('Name', 'Streamlines Over the Airfoil')
        hold on
            plot(x_circle_plot,airfoil_proj)
            fill(x_circle_plot,airfoil_proj,'cyan')
            contour(x_coords,y_coords,psi,linspace(min(min(psi)),max(max(psi)),Contour_Detail));
            axis([-1.5 1.5 -1 1 ])
            axis off
            title('Stream Lines Over the Airfoil', 'FontName','lm roman 12')

% Velocity and pressure Distribution Contour Around the Airfoil
% zoomed out
figure('Name', 'Velocity contours')
        hold on
            plot(x_circle_plot,airfoil_proj,'LineWidth',1.5)
            title('Velocity Contour Around the Airfoil')
            contourf(x_coords,y_coords,V,Contour_Detail,'edgecolor','none')
            axis off
            xlabel('x','FontSize',16)
            ylabel('y','FontSize',16)
            c = colorbar;
            c.Label.String = 'Velocity';
            c.Label.FontSize = 16;
            axis equal

figure('Name', 'Pressure Coefficient contours')
            hold on
            plot(x_circle_plot,airfoil_proj,'LineWidth',1.5)
            title('Pressure Contour Around the Airfoil')
            contourf(x_coords,y_coords,Cp,Contour_Detail,'edgecolor','none')
            axis off
            xlabel('x','FontSize',16)
            ylabel('y','FontSize',16)
            c = colorbar;
            c.Label.String = 'Coefficient of Pressure';
            c.Label.FontSize = 16;
            axis equal

% zoomed in
figure('Name', 'Velocity and Pressure Coefficient contours')
    tiledlayout(1,2)
        nexttile
        hold on
            plot(x_circle_plot,airfoil_proj,'LineWidth',1.5)
            title('Velocity Contour Around the Airfoil')
            contourf(x_coords,y_coords,V,Contour_Detail,'edgecolor','none')
            axis([-1.5 1.5 -1.5 1.5 ])
            axis off
            xlabel('x','FontSize',16)
            ylabel('y','FontSize',16)
```

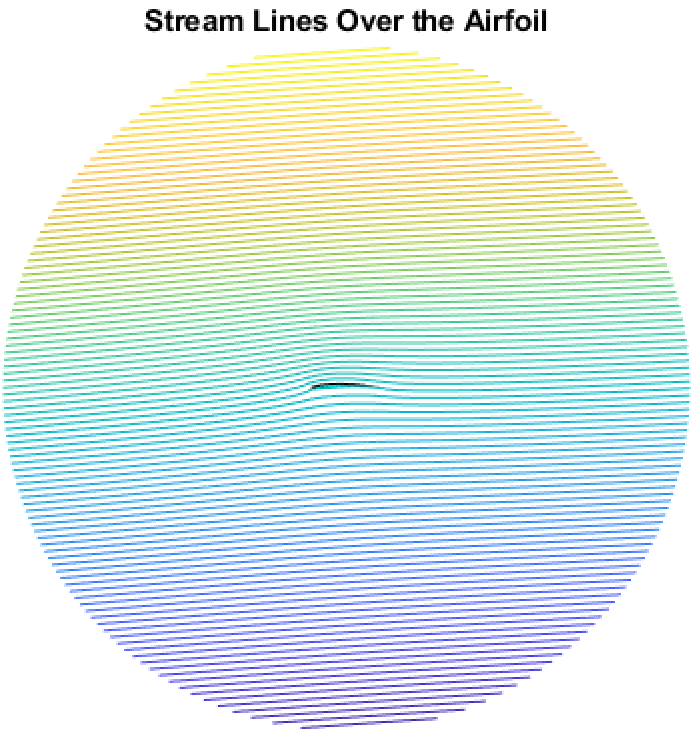
```
        c = colorbar;
        c.Label.String = 'Velocity';
        c.Label.FontSize = 16;
    nexttile
    hold on
        plot(x_circle_plot,airfoil_proj,'LineWidth',1.5)
        title('Pressure Contour Around the Airfoil')
        contourf(x_coords,y_coords,Cp,Contour_Detail,'edgecolor','none')
        axis([-1.5 1.5 -1.5 1.5 ])
        axis off
        xlabel('x','FontSize',16)
        ylabel('y','FontSize',16)
        c = colorbar;
        c.Label.String = 'Coefficient of Pressure';
        c.Label.FontSize = 16;
```



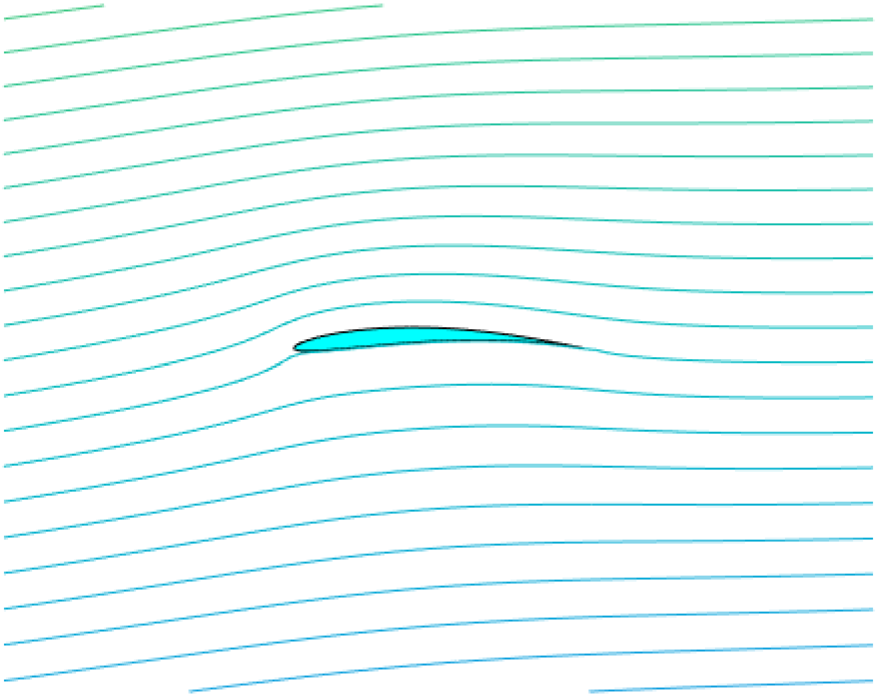



Velocity distribution
Numerical
Analytical
Airfoil
V
x
200
150
100
50
0
-0.6
-0.4
-0.2
0
0.2
0.4
0.6

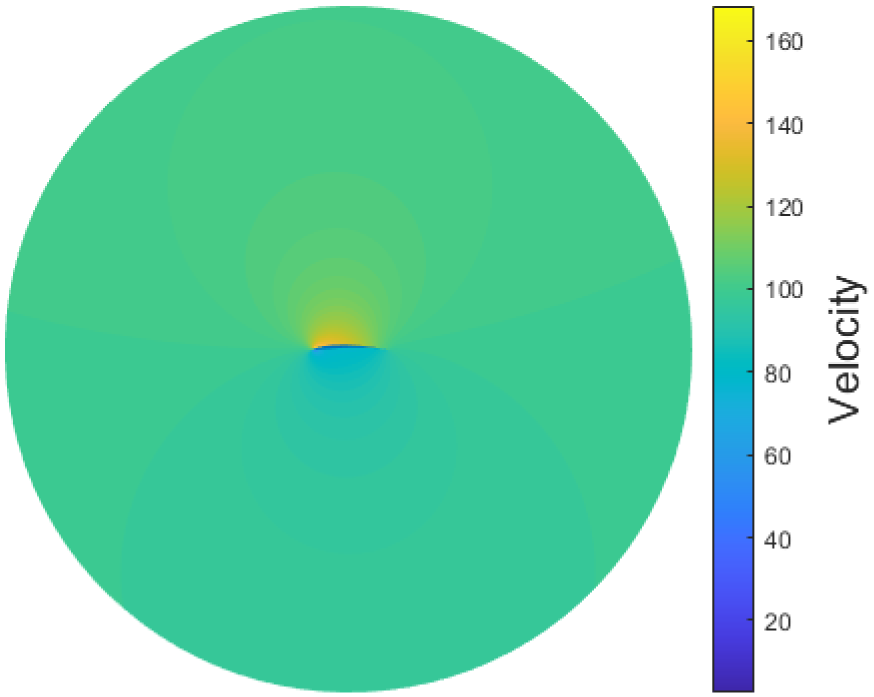

Pressure coefficient distribution
Numerical
Analytical
Airfoil
C_p
x
2
0
-2
-4
-0.6
-0.4
-0.2
0
0.2
0.4
0.6

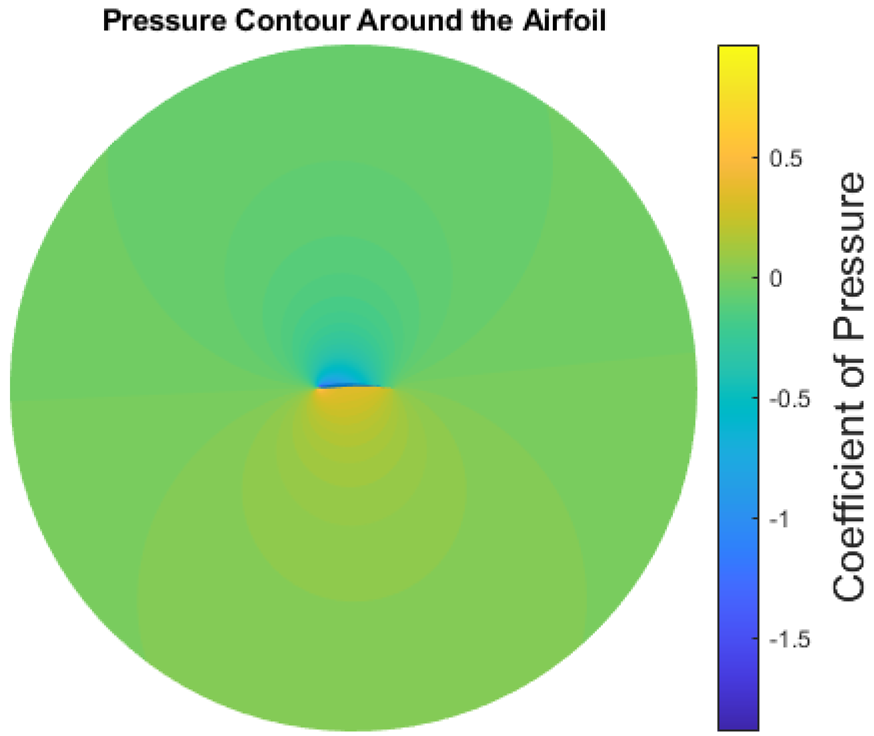

Stream Lines Over the Airfoil

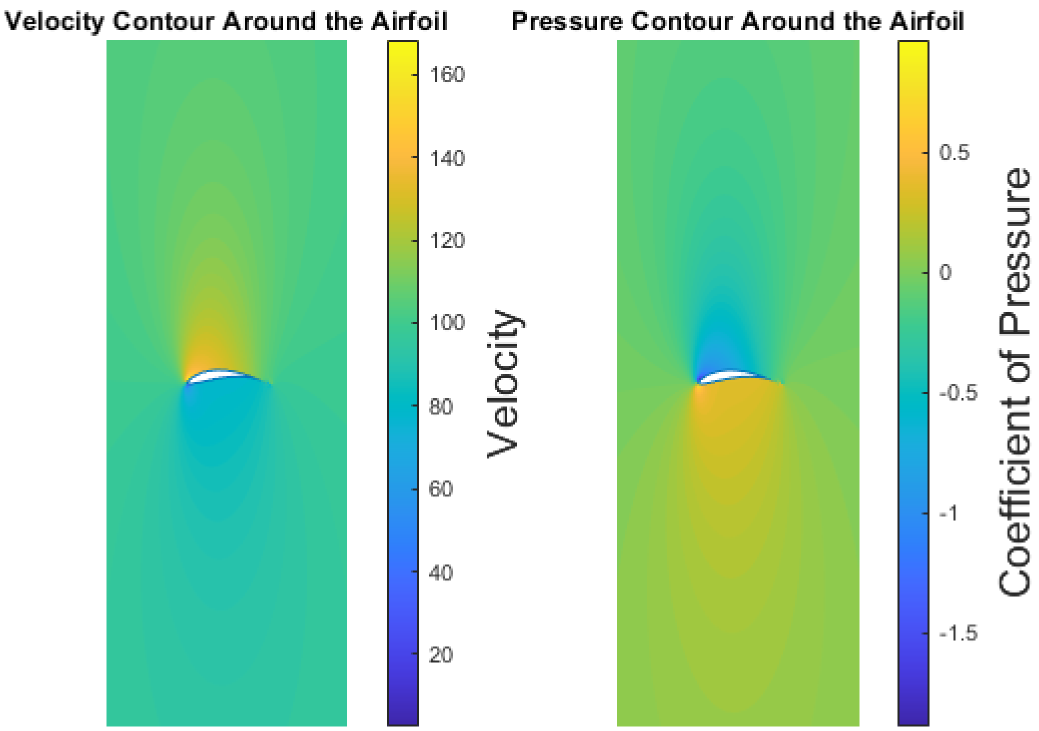
Stream Lines Over the Airfoil



Velocity Contour Around the Airfoil




Pressure Contour Around the Airfoil
0.5
0
-0.5
-1
-1.5
Coefficient of Pressure


Velocity Contour Around the Airfoil
160
140
120
100
80
60
40
20
Velocity
Pressure Contour Around the Airfoil
0.5
0
-0.5
-1
-1.5
Coefficient of Pressure

Appendix B: Joukowski Analytical Solution

Flow Over Joukowski Airfoil Function

This Fucion Calculates the Velocity and pressure Distribution over an Airforil by using the analysical joukowski transformaiotn. Funcion Argumentd : (joukowski(Vinf, AoA, c, C_max_c, t_max_c))

```
V_inf: the free streem Velocity.
  AoA: angle of attack
    c: cord line length
C_max_c: max camber (% of cord)
t_max_c: max thickness (% of cord)
i_max: number of points
```

Published by: Mohamed Tarek Mohamed Amien publishing year: 1st January /2023

```
function [V1, C_p, x1] = Joukowski(Vinf, AoA, c, C_max_c, t_max_c,  i_max )
```

Initialization (Inputs)

```
% The Free Stream Velocity

% Vinf=100;

% Enter the Joukowski Airfoil Parameters

% c=1;              % Chord
% C_max_c=0.04;     % Maximum Camber/Chord Percentage
% t_max_c=0.05;     % Maximum Thickness/Chord Percentage

% Grid parameters

%  i_max = 10;       % eta_1 grid size
% j_max = 100;      % eta_2 grid size

%  Joukowski Transformation Parameters.
```

Joukowski circle Parameters

```
b=c/4;
e=t_max_c/1.3;
beta=2*C_max_c;
a=b*(1+e)/cos(beta);

% Circle Shift in Joukowski Z Plane
x0 = -b*e;
y0 = a*beta;
```

Z' Plane

```
D_theta = 2*pi()/(i_max-1);           % angle step size in the domain  Delta theta = 2*pi/(imax-1)
Theta_dash_vec = 0:D_theta:2*pi();    % theta vector in z_dahs Plane
r_dash = a*ones(1,i_max);               % Radius of the circle
```

```
% x'-y' coords in z_dahs plane

x_dash = r_dash.*cos(Theta_dash_vec);
y_dash = r_dash.*sin(Theta_dash_vec);
```

Z Plane

```
% x-y coords in z_dahs plane

x = x_dash + x0;
y = y_dash + y0;

r_ = sqrt(x.^2+ y.^2);                % the radious of the circle in z plane

theta_vec = atan2(y,x);
```

Z1 Plane , the Airfoil Plane

```
% x1-y1 coords in z_dahs plane

x1 = x.*(1+(b^2)./(x.^2+y.^2));
y1 = y.*(1-(b^2)./(x.^2+y.^2));

r1 = sqrt(x1.^2+ y1.^2);              % the radious of the circle in Z1 plane

theta1_vec = atan2(y1,x1);
```

Joukowski Analytical solution

```
V_rDash = Vinf*(1-(a./r_dash).^2).*cos(Theta_dash_vec-AoA);
V_thetaDash = -Vinf *(sin(Theta_dash_vec-AoA).*(1+(a./r_dash).^2)+2*(a./r_dash)*sin(AoA+beta));

% Velocity Magnitude over the Airfoil in Z1 Plane
V1 = sqrt(V_thetaDash.^2./(1-2*(b./r_).^2.*cos(2*theta_vec)+(b./r_).^4));

V1_x(1:1:100,:) = V1.*cos(theta1_vec).*ones(100,1);
V1_y(1:1:100,:) = V1.*sin(theta1_vec).*ones(100,1);
% pressure Coefficient
C_p = 1-(V1/Vinf).^2;
```

Airfoil Coordinates with Formula

```
 X = 2*b*cos(Theta_dash_vec);
 Y = 2*b*e*(1-cos(Theta_dash_vec)).*sin(Theta_dash_vec)+2*b*beta*sin(Theta_dash_vec).^2;
```

Plot commands

```
% plot the Z Circle and the Circle shift

% figure('Name', 'Joukowski Circles' )
%         plot(x_dash, y_dash)
```

```
%         grid on
%         axis([-0.5 0.5 -0.5 0.5])
%         axis('equal')
%         xlabel('$x$', 'interpreter', 'latex')
%         ylabel('$y$', 'interpreter', 'latex')
% hold on
%         plot(x, y)
%         grid on
%         axis([-0.5 0.5 -0.5 0.5])
%         axis('equal')
%         xlabel('$x''$', 'interpreter', 'latex')
%         ylabel('$y''$', 'interpreter', 'latex')


figure('Name', 'Joukowski Airfoil check' )

      plot(x1, y1)
      grid on
hold on
      plot(X, Y, '.')
      grid on
      axis('equal')
      title('Joukowski Airfoil check')


figure('Name', 'Analytical Pressure and Velocity distribution over the Airfoil' )
tiledlayout(2,1);
nexttile
      hold on
      plot(x1, V1, '-' ,'LineWidth',1.5,'color','red')
      plot(x1, 700*y1,'LineWidth',0.5,'color','black')
      fill(x1, 700*y1,'cyan')
      grid on
      xlabel('$x$', 'interpreter', 'latex')
      ylabel('$V$', 'interpreter', 'latex')
      legend('Velocity distribution','Airfoil')
      title('Velocity distribution', 'FontName','lm roman 9')
nexttile
      hold on
      fill(x1, 20*y1+min(C_p),'cyan')
      plot(x1, 20*y1+min(C_p),'LineWidth',0.5,'color','black')
      plot(x1, C_p, '-','LineWidth',1.5,'color','blue')
      grid on
      xlabel('$x$', 'interpreter', 'latex')
      ylabel('$C_p$', 'interpreter', 'latex')
      legend('Cp distribution','Airfoil')
      title('Pressure Coefficient distribution', 'FontName','lm roman 9')

  end
```