Compressible and Viscous Aerodynamics

Source/Vortex Panel Method for Evaluating the Potential Flow Over an Arbitrary Airfoil Shape.

Ahmed Mohamed Hassan Abdulrahman

SEC: 1 BN:11

Under supervision of Dr. Professor Hesham El-Banna

Table of Contents

Abstract

This project will be discussing one method of evaluating the potential flow (Ideal flow) over 3 airfoils; NACA 0006, NACA 0012, NACA 0018. Computing the Coefficient of lift of the airfoil at 0 angle of attack and the surface velocities at different distances from the leading edge using the Source/Vortex Panel Method and validating it using XFOIL. Ideal flow is often the first type of fluid motion that we study about an airfoil or any aerodynamic body, because it is the simplest. Large parts of the flows past ships, submarines, cars and light aircraft are closely ideal. Analysis of the potential flow is also critical to provide the required information that would help along with other more complicated analyses, for example, the Boundary Layer Theory. One discrepancy in this project's analysis is that XFOIL is used to validate the results and an experimental approach isn't used to validate them.

Introduction

The Source/Vortex panel method models the flow past an airfoil as the summation of a uniform flow and a series of vortex panels or sheets arranged to form a closed polygon with a shape that approximates, as nearly as possible, the actual curved shape of the airfoil, as show in figure 1.

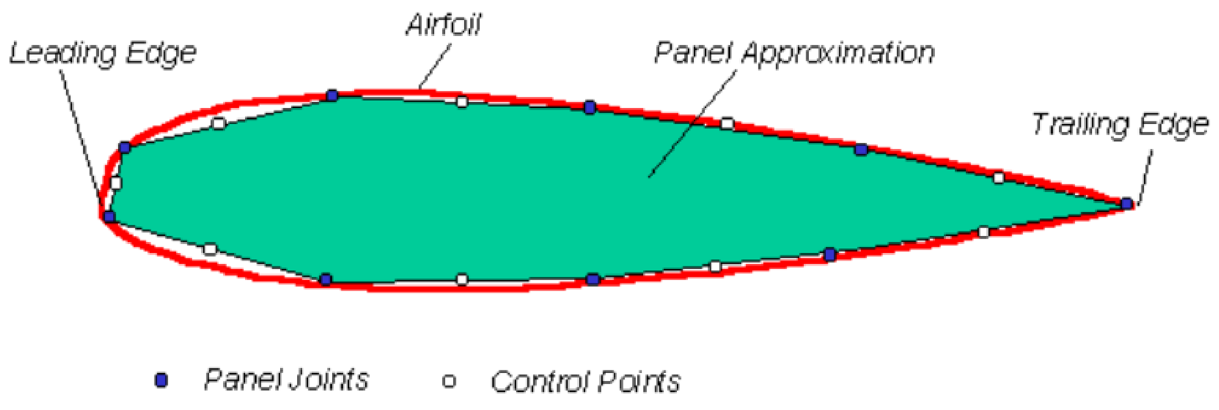


Figure 1: Vortex Panel Approximation to an airfoil

Each panel of the polygon would be made up of a line of Sources/Vortices of equal strength or a mixture of both. We assume that the distribution of strength of sources of each panel varies and the strength is continuous across the panel joints meanwhile the vortices strength distributions are constant and equal across every panel, except at the trailing edge. The problem is to choose these strengths so that the flow past the panels is a realistic representation of the flow over the airfoil. To do this we invoke the so-called 'non-penetration' condition - the condition that the flow cannot pass through the surface of the airfoil or, equivalently, that the component of velocity perpendicular to the airfoil surface is zero. We apply this condition approximately by writing equations for the velocity generated by all the panels, plus the free stream, at the central point of each panel (called the control point). We then set the component of this velocity perpendicular to the panel equal to zero. We will model N panels on an airfoil therefore we would have N control points. Evaluating the contribution of the sources of the panel on the control point using the la-place equation we would have 1 equation for each panel. Therefore, we have N control points this gives us N equations and we can get all N source strengths. Since we assumed that the vortex strength distribution is the same for all the panels (strength/unit length is constant), we would only need to solve 1 equation that describes the vorticity over all the panels. The final equation is obtained using the Kutta condition. The Kutta condition encapsulates the observation that the flow can't go around the trailing edge, but must leave the airfoil there. This is a consequence of viscous effects, which are otherwise is absent from this calculation. For the Kutta condition to be satisfied the strengths of the vortex panels must be equal and opposite where they meet at the trailing-edge joint. Solving these equations using matrix inversion we can evaluate the strengths of the sources and the vorticity.

Derivation of Source/Vortex Panel Method

This section of the project will aim at mathematically deriving the vortex panel method into existence, going by all the sub-steps that would be used in the actual derivation.

Calculating Circulation

The definition of circulation is the integration of the velocity dotted with the contour vector:

$$\Gamma \equiv - \oint_{contour} \vec{V} \cdot d\vec{s}$$

Where $\vec{V}$ is the velocity vector, and $d\vec{s}$ is the contour vector.

We then can calculate the lift generated due to that circulation and is defined as: $L' = \rho V \Gamma$

Defining the velocity and contour vectors in cartesian components:

$$\vec{V} = V_x\hat{\imath} + V_y\hat{\jmath} \qquad\qquad d\vec{s} = dx\hat{\imath} + dy\hat{\jmath}$$

We can then write the dot product as a summation of x and y.

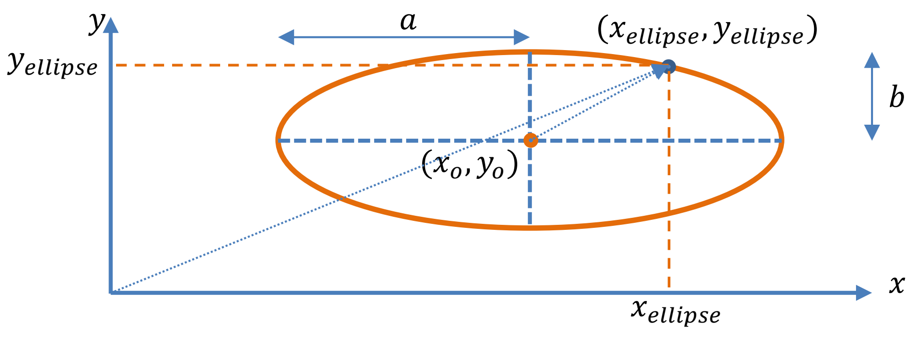
$$\Gamma = - \oint_{contour} V_x \cdot dx - \oint_{contour} V_y \cdot dy$$

Since we will be having velocity vectors at discrete grid points (i,j), we must define a closed curve around which to compute the integral.

$$V_x(i,j) \text{ } and \text{ } V_y(i,j) \quad \text{and} \quad x = [x_1, x_2, \dots \dots \dots, x_N], y = [y_1, y_2, \dots \dots \dots, y_N]$$

Defining the closed curve as an ellipse, we will parameterize using Theta. Therefore:

$$x_{ellipse} = a\cos(\theta) + x_o \qquad\qquad y_{ellipse} = b\sin(\theta) + y_o$$



And since the closed curve doesn't necessarily align with the grid points. Therefore we must interpolate the velocity to the ellipse points.

Incompressible potential flow

We would use Navier stokes equations with the assumptions that the flow is irrotational and incompressible.

Irrotational Flow

The vorticity should equal the curl of the velocity: $\boxed{\xi = \nabla \times \bar{V}}$. where $\bar{V} = [u, v, w]$

Irrotational flow means that $\nabla \times \bar{V} = 0$, therefore the vorticity would be zero. And avoiding the trivial solution of $\bar{V} = 0$.

Using the vector Identity $\nabla \times \nabla\phi = 0$. Where ϕ is the velocity potential.

$$\nabla \times \nabla\phi = \left[\frac{\partial\phi}{\partial y\partial z} - \frac{\partial\phi}{\partial z\partial y}\right]\hat{\imath} - \left[\frac{\partial\phi}{\partial x\partial z} - \frac{\partial\phi}{\partial z\partial x}\right]\hat{\jmath} + \left[\frac{\partial\phi}{\partial x\partial y} - \frac{\partial\phi}{\partial y\partial x}\right]\hat{k}=0$$

Assuming the symmetry of the 2nd derivatives $\frac{\partial\phi}{\partial x\partial y} = \frac{\partial\phi}{\partial y\partial x}$

Comparing the coefficients, we can say that $\boxed{\bar{V} = \nabla\phi}$. $\quad u = \frac{\partial\phi}{\partial x} \quad v = \frac{\partial\phi}{\partial y} \quad w = \frac{\partial\phi}{\partial z}$

To get the velocity potential we can solve the mass and momentum equations.

Incompressible Flow

The density is constant. Using only mass conservation (continuity equation): $\frac{\partial p}{\partial t} + \nabla \cdot (\rho\bar{V}) = 0$

We can prove at the end that $\boxed{\nabla \cdot \bar{V} = 0}$

Laplace Equation

$$\nabla \cdot \bar{V} = \nabla \cdot \nabla\phi = 0 \qquad \nabla \cdot \nabla\phi = \nabla^2\phi = 0 \qquad \nabla^2\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0$$

Since we have an equation that we can use to solve for ϕ and hence we can solve for $\bar{V}$

This equation is linear and 2nd order therefore we can use the principle of superposition:

$$\phi = \phi_1 + \phi_2 + \phi_3 + \cdots + \phi_n$$

Elementary Flows

The derivations of the potential functions of these flows have been clearly explained and derived in the previous Aerodynamics course.

Therefore, taking their final forms:

Uniform Flow: $\phi_u = V_\infty \operatorname{Cos}(\alpha)\, x + V_\infty \operatorname{Sin}(\alpha)\, y$

Source Flow: $\phi_s = \frac{\Lambda}{2\pi}\ln(r)$

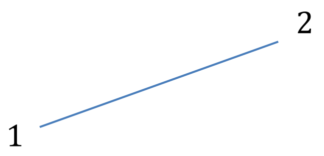
Vortex Flow: $\phi_s = \frac{-\Gamma}{2\pi}\theta$

These would be summed up together to form the Source/Vortex Panel Method that we will be familiar with

Other methods can be used to model the elementary flows, but they won't be discussed in this project.

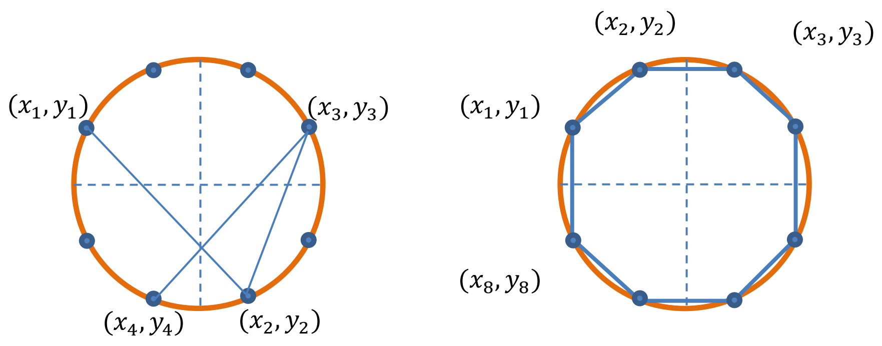
Panel Method Geometry

To define a polygon in 2D space we can use lines. A line has 2 nodes.



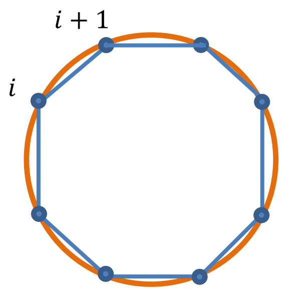
We can represent a circle using 8 nodes and 8 lines; therefore, it will look like an octagon. But this octagon doesn't fully represent the circle, therefore we would have to use more lines/nodes to represent this circle accurately. A good enough approximation can be used without affecting the accuracy of the results.

The order of the connected points matter, since if arranged in an arbitrary way wouldn't give the shape of a circle.



Taking that the number of the node is i. Then if we want to find the side length, we can calculate the side length of the 1st panel using use $S_1 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ using Pythagorean theorem.

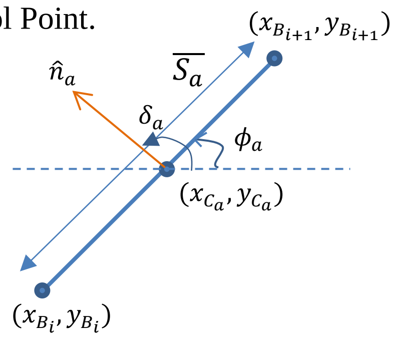
Coordinate system



Defining a coordinate system for each panel we can conclude that:

Defining a new point called the Control Point.

$$\overline{S_a} = \sqrt{\left(x_{B_{i+1}} - x_{B_i}\right)^2 + \left(y_{B_{i+1}} - y_{B_i}\right)^2}$$



Clockwise and counter-clockwise ordering is important since we will be defining normal and tangential vectors for each panel; defining normal out of polygon.

ϕ_a is the angle from positive x-axis to the inside surface of the panel.

δ_a is the angle from positive x-axis to outward normal vector of panel $\hat{n}_a$.

β_a is the angle between freestream vector $\bar{V}_\infty$ and the outward normal of panel.

The control point is the point in the middle of the panel. $x_{C_a} = \frac{x_{B_i}+x_{B_{i+1}}}{2}$ and $y_{C_a} = \frac{y_{B_i}+y_{B_{i+1}}}{2}$.

By defining $dx = X_{B_{i+1}} - x_{B_i}$ and $dy = y_{B_{i+1}} - y_{B_i}$. We can show that, $\phi_a = \tan^{-1}\left(\frac{dy}{dx}\right)$ and if $\phi_a < 0$ we can use $\phi_a = \phi_a + 360$.

Finally, $\delta_a = \phi_a + 90$.

Since β_a is also dependant on the angle of attack. $\beta_a = \phi_a + 90 - \alpha = \delta_a - \alpha$.

Rest of derivation

The previous parts were the most important, after concluding the derivation:

The

For the normal velocity:

$$V_{n,i} = V_\infty Cos(\beta_i) + \sum_{j=1}^{N} \frac{\lambda_j}{2\pi} \int_j \frac{\partial}{\partial n_i} \ln(r_{ij})\, ds_j + \sum_{j=1}^{N} \frac{-\gamma}{2\pi} \int_j \frac{\partial \theta_{ij}}{\partial n_i} ds_j = 0$$

At $j = i$

$$\sum_{j=i}^{N} \frac{\lambda_j}{2\pi} \int_j \frac{\partial}{\partial n_i} \ln(r_{ij})\, ds_j = \frac{\lambda_j}{2}$$

$$\sum_{j=i}^{N} \frac{-\gamma}{2\pi} \int_j \frac{\partial \theta_{ij}}{\partial n_i} ds_j = 0$$

Let $\int_j \frac{\partial}{\partial n_i} \ln(r_{ij})\, ds_j = I_{ij}$, $\int_j \frac{\partial \theta_{ij}}{\partial n_i} ds_j = K_{ij}$

$$\therefore V_{n,i} = V_\infty Cos(\beta_i) + \frac{\lambda_j}{2} + \sum_{\substack{j=1 \\ j\neq i}}^{N} \frac{\lambda_j}{2\pi} I_{ij} + \sum_{\substack{j=1 \\ j\neq i}}^{N} \frac{-\gamma}{2\pi} K_{ij} = 0 \text{ [multiplying by } 2\pi]$$

$$\pi\lambda_j + \sum_{\substack{j=1 \\ j\neq i}}^{N} \lambda_j I_{ij} + \sum_{\substack{j=1 \\ j\neq i}}^{N} -\gamma K_{ij} = -2\pi V_\infty Cos(\beta_i)$$

Therefore, the system of equation will be [for let's say 3 panels]:

$$\begin{bmatrix} \pi & I_{12} & I_{13} & -(K_{12}+K_{13}) \\ I_{21} & \pi & I_{23} & -(K_{21}+K_{23}) \\ I_{31} & I_{32} & \pi & -(K_{31}+K_{31}) \\ \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \gamma \end{bmatrix} = \begin{bmatrix} -2\pi V_\infty Cos(\beta_1) \\ -2\pi V_\infty Cos(\beta_2) \\ -2\pi V_\infty Cos(\beta_3) \\ \dots \end{bmatrix} \quad \begin{matrix} N\ Unkowns \\ N-1\ equations \end{matrix}$$

For the last equation we will apply the Kutta condition[1].

We will approximate the Kutta condition by setting the first panel and the last panel velocity to be equal

[1] In fluid flow around a body with a sharp corner, the Kutta condition refers to the flow pattern in which fluid approaches the corner from above and below, meets at the corner, and then flows away from the body. None of the fluid flows around the sharp corner.

$$V_{t,N} = -V_{t,1} \rightarrow V_{t,N} + V_{t,1} = 0$$

Similar to the normal velocity the tangential velocity eq is:

$$V_{t,N} = V_\infty Sin(\beta_N) + \frac{\gamma_N}{2} + \sum_{j=1}^{N} \frac{\lambda_j}{2\pi} J_{Nj} + \sum_{j=1}^{N} \frac{-\gamma}{2\pi} L_{Nj} = 0$$

Let $\int_j \frac{\partial}{\partial t_i} \ln(r_{ij})\, ds_j = J_{ij}$, $\int_j \frac{\partial \theta_{ij}}{\partial t_i} ds_j = L_{ij}$

Therefore

$$V_{t,N} + V_{t,1} = V_\infty Sin(\beta_1) + \frac{\gamma_1}{2} + \sum_{j=2}^{N} \frac{\lambda_j}{2\pi} J_{1j} + \sum_{j=2}^{N} \frac{-\gamma}{2\pi} L_{1j} + V_\infty Sin(\beta_N) + \frac{\gamma_N}{2} + \sum_{j=1}^{N} \frac{\lambda_j}{2\pi} J_{Nj} + \sum_{j=1}^{N} \frac{-\gamma}{2\pi} L_{Nj} = 0$$

$$\sum_{\substack{j=1 \\ j\neq i \\ j\neq N}}^{N} \lambda_j (J_{1j} + J_{Nj}) + \gamma \left[\sum_{\substack{j=1 \\ j\neq i \\ j\neq N}}^{N} -(L_{1j} + L_{Nj}) + 2\pi \right] = -2\pi V_\infty \left(Sin(\beta_1) + Sin(\beta_N) \right)$$

So, the system of equation matrix will be

$$\begin{bmatrix} \pi & I_{12} & I_{13} & -(K_{12}+K_{13}) \\ I_{21} & \pi & I_{23} & -(K_{21}+K_{23}) \\ I_{31} & I_{32} & \pi & -(K_{31}+K_{31}) \\ J_{31} & (J_{13}+J_{32}) & J_{31} & (L_{12}+L_{13}+L_{31}+L_{32}) + 2\pi \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \gamma \end{bmatrix} = \begin{bmatrix} -2\pi V_\infty Cos(\beta_1) \\ -2\pi V_\infty Cos(\beta_2) \\ -2\pi V_\infty Cos(\beta_3) \\ -2\pi V_\infty \left(Sin(\beta_1) + Sin(\beta_N) \right) \end{bmatrix}$$

Finally, Using Matrix inversion we can solve for the strengths. And we would be able to solve for the Tangential velocity. This will be easily done and solved for due to each elementary flow.

Application of the Source/Vortex Panel Method

Applying this method to calculate the potential flow characteristics of 3 airfoils, NACA 0006, NACA 0012, NACA 0018. Computing the Coefficient of lift of the airfoil at 0 angle of attack and the surface velocities at different distances from the leading edge using the Source/Vortex Panel Method and validating it using XFOIL. It was implemented using MATLab.

Using 400 Panels, with uniform flow velocity of 50 m/s and a chord of 1 meter.

Results of the Application

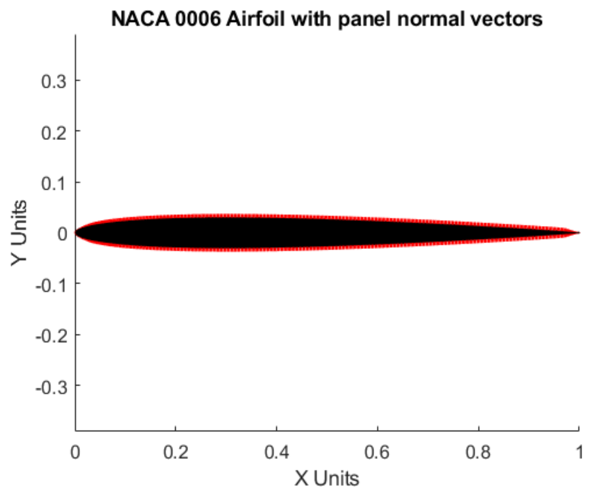
NACA 0006

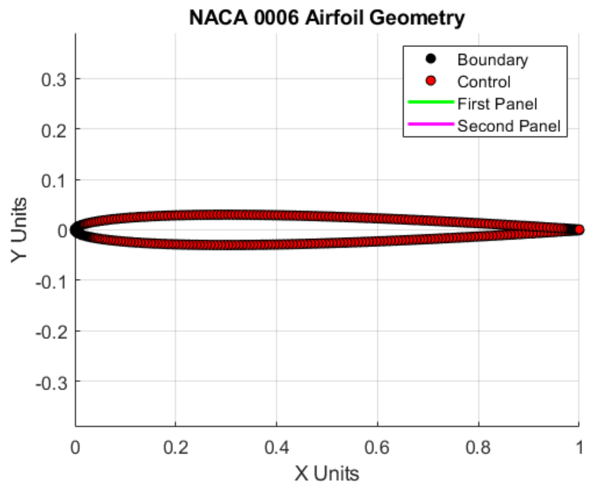
Lift and Moment Coefficient using Source/Vortex Panel Method

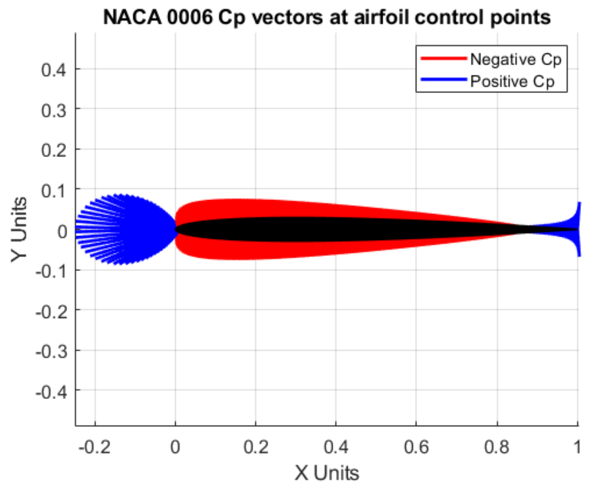
	Source/Vortex Panel Method	XFOIL
Lift Coefficient	0.0000	-0.0009
Moment Coefficient	0.0000	0.0002

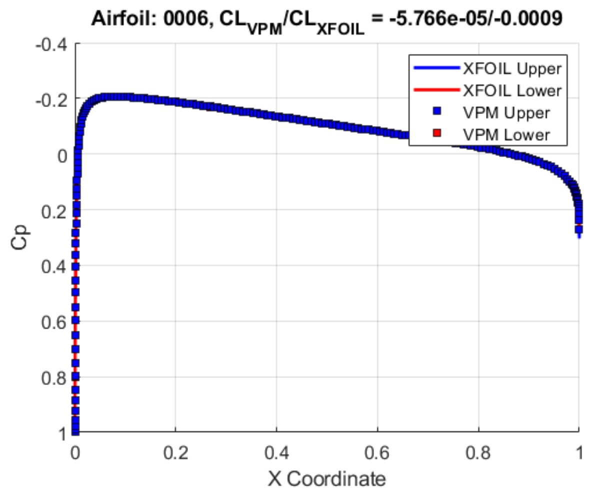
velocities using Source/Vortex Panel Method at different x-coordinates.

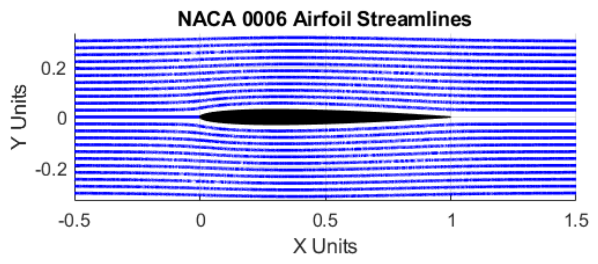
X-coordinate	0.2	0.4	0.6	0.8	1
Tangential Velocity	54.48	53.268	52.02	50.632	42.687

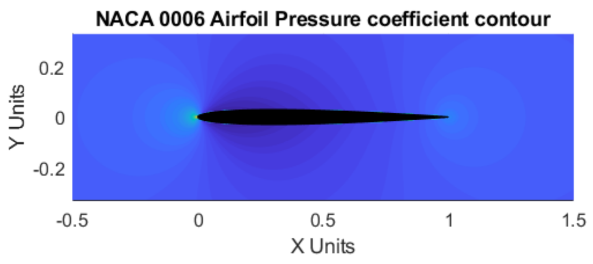












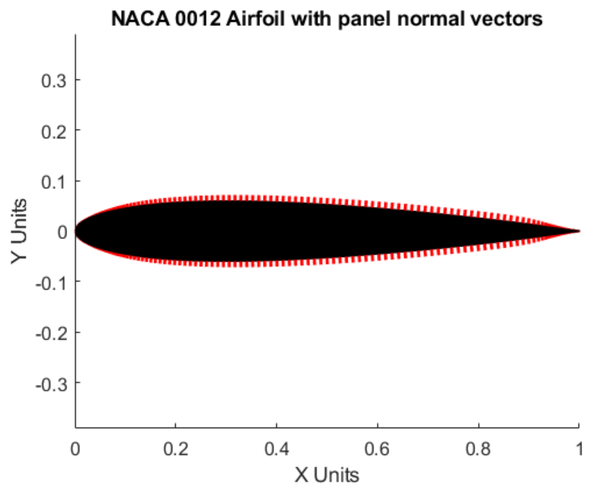
NACA 0012

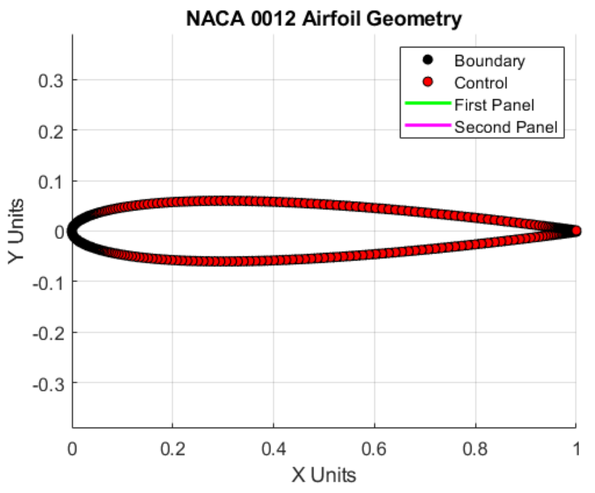
Lift and Moment Coefficient using Source/Vortex Panel Method

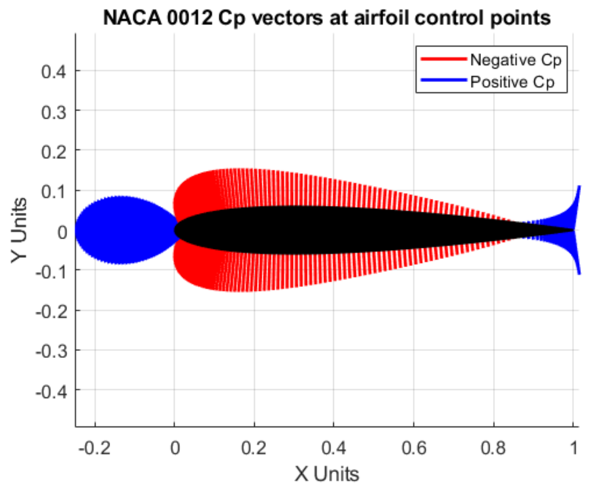
	Source/Vortex Panel Method	XFOIL
Lift Coefficient	0.0000	-0.0009
Moment Coefficient	0.0000	0.0002

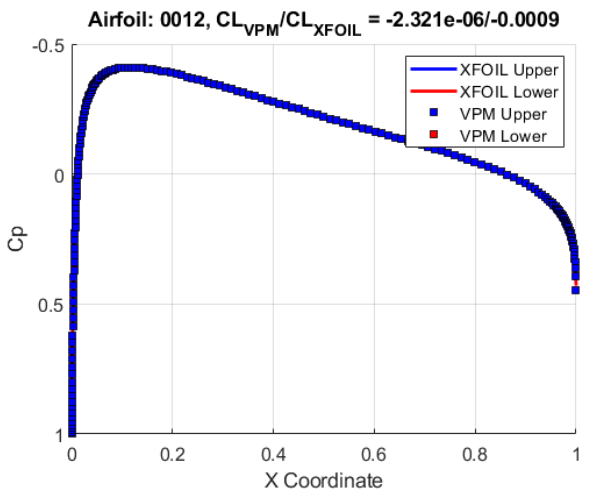
velocities using Source/Vortex Panel Method at different x-coordinates.

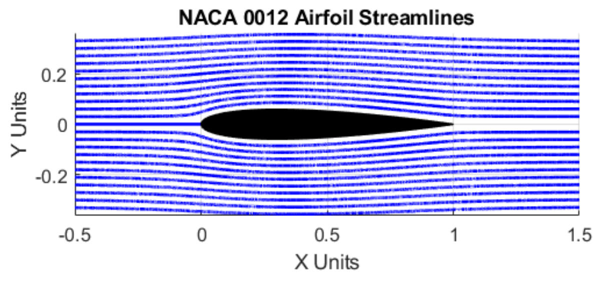
X-coordinate	0.2	0.4	0.6	0.8	1
Tangential Velocity	58.895	56.524	53.947	51.131	37.211

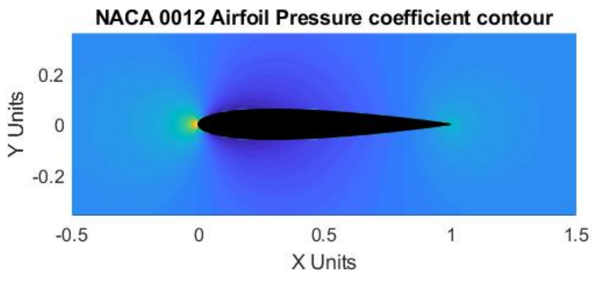












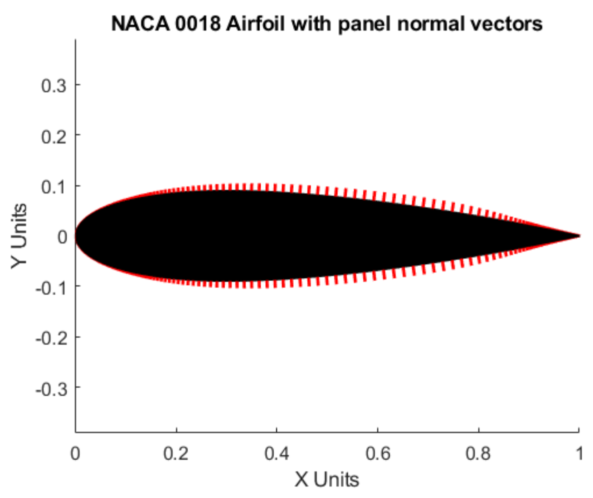
NACA 0018

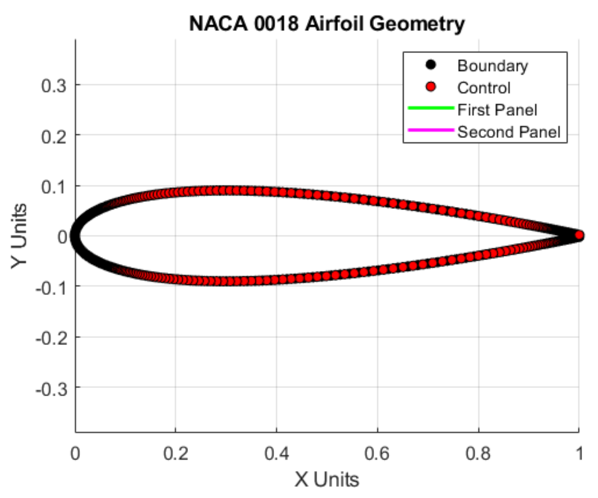
Lift and Moment Coefficient using Source/Vortex Panel Method

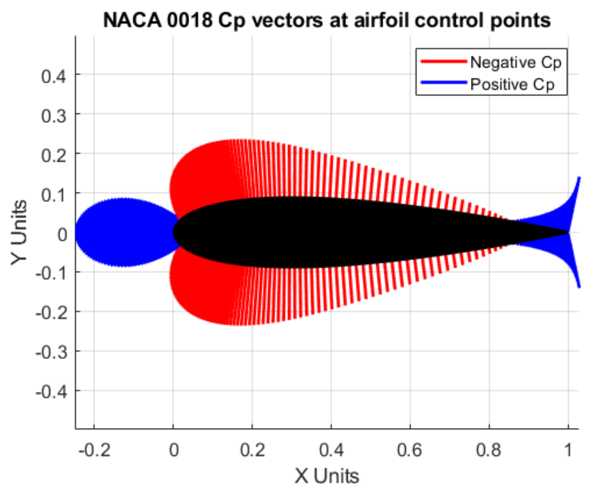
	Source/Vortex Panel Method	XFOIL
Lift Coefficient	0.0000	0.0002
Moment Coefficient	-0.0000	-0.0000

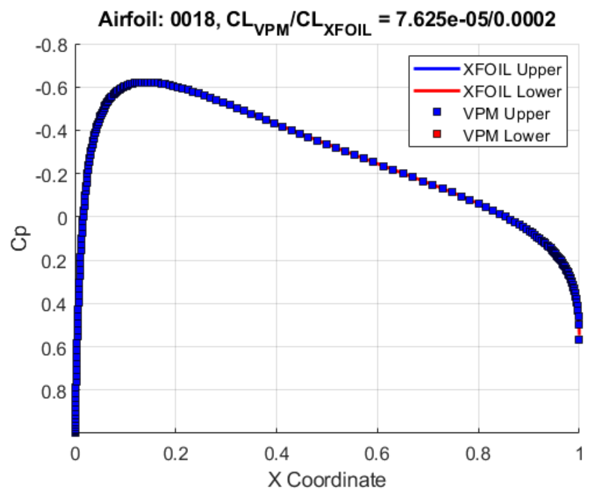
velocities using Source/Vortex Panel Method at different x-coordinates.

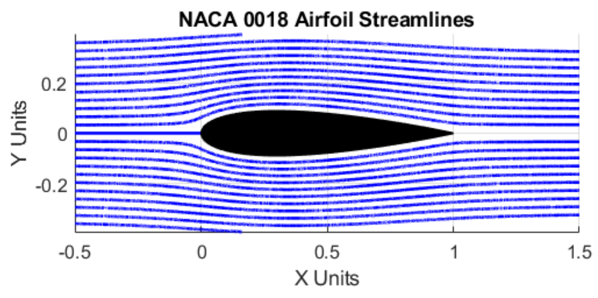
X-coordinate	0.2	0.4	0.6	0.8	1
Tangential Velocity	63.276	59.756	55.787	51.505	32.875

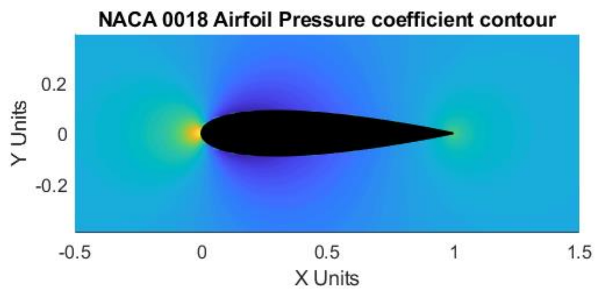












Conclusion

The Source/Vortex Panel Method Proved to be quite accurate, more accurate than XFOIL, since it computed that the Coefficient of lift is zero, which is what we should be expecting from a symmetric airfoil at zero angle of attack. The results are promising and can be greatly improved upon to accommodate for many more calculations and characteristics to be evaluated from it.

References

[1] [Online]. Available: https://youtube.com/playlist?list=PLxT-itJ3HGuUDVMuWKBxyoY8Dm9O9qstP.

[2] [Online]. Available: https://www.engapplets.vt.edu/fluids/vpm/vpminfo.html.

Appendix

Appendix A: Main Script

```
% SOURCE/VORTEX PANEL METHOD - SINGLE AIRFOIL


% Notes     : This code is not optimized, but is instead written in such a way
%             that it is easy to understand.

% Functions Needed:
% - XFOIL.m
% - COMPUTE_IJ_SPM.m
% - COMPUTE_KL_VPM.m
% - STREAMLINE_SPM.m
% - STREAMLINE_VPM.m

% Programs Needed:
% - xfoil.exe
%
% Folder Needed:
% - Airfoil_DAT_Selig: folder containing all Selig-format airfoils


clear;
clc;
close all;
```

PROVIDED GIVENS

```
% Flag to specify creating or loading airfoil
flagAirfoil.XFoilCreate = 1;                                      % Create specified NACA
airfoil in XFOIL
flagAirfoil.XFoilLoad   = 0;                                      % Load Selig-format
airfoil from directory

% User-defined knowns
Vinf = 50;                                                         % Freestream velocity []
(just leave this at 1)
AoA  = 0;                                                         % Angle of attack [deg]
NACA = '0018';                                                    % NACA airfoil to load
[####(#)]

% Plotting flags
flagPlot = [1;          % Airfoil with panel normal vectors
            1;          % Geometry boundary pts, control pts, first panel, second panel
            1;          % Cp vectors at airfoil surface panels
            1;          % Pressure coefficient comparison (XFOIL vs. SPVP)
            1;          % Airfoil streamlines
            1];         % Pressure coefficient contour
```

XFOIL - CREATE/LOAD AIRFOIL

```
% PPAR menu options
PPAR.N  = '401';                                                  % "Number of panel nodes"
PPAR.P  = '4';                                                    % "Panel bunching
parameter"
PPAR.T  = '1';                                                    % "TE/LE panel density
```

```
ratios"
PPAR.R  = '1';                                                   % "Refined area/LE panel
density ratio"
PPAR.XT = '1 1';                                                 % "Top side refined area
x/c limits"
PPAR.XB = '1 1';                                                 % "Bottom side refined
area x/c limits"

% Call XFOIL function to obtain the following:
% - Airfoil coordinates
% - Pressure coefficient along airfoil surface
% - Lift, drag, and moment coefficients
[xFoilResults,success] = XFOIL(NACA,PPAR,AoA,flagAirfoil);       % Get XFOIL results for
prescribed airfoil
if (success == 0)                                                % If user canceled
airfoil dialog box
    return;                                                      % Exit the program
end

% Separate out results from XFOIL function results
afName  = xFoilResults.afName;                                   % Airfoil name
xFoilX  = xFoilResults.X;                                        % X-coordinate for Cp
result
xFoilY  = xFoilResults.Y;                                        % Y-coordinate for Cp
result
xFoilCP = xFoilResults.CP;                                       % Pressure coefficient
XB      = xFoilResults.XB;                                       % Boundary point X-
coordinate
YB      = xFoilResults.YB;                                       % Boundary point Y-
coordinate
xFoilCL = xFoilResults.CL;                                       % Lift coefficient
xFoilCD = xFoilResults.CD;                                       % Drag coefficient
xFoilCM = xFoilResults.CM;                                       % Moment coefficient

% Number of boundary points and panels
numPts = length(XB);                                             % Number of boundary
points
numPan = numPts - 1;                                             % Number of panels
(control points)
```

CHECK PANEL DIRECTIONS - FLIP IF NECESSARY

```
% Check for direction of points
edge = zeros(numPan,1);                                          % Initialize edge value
array
for i = 1:1:numPan                                               % Loop over all panels
    edge(i) = (XB(i+1)-XB(i))*(YB(i+1)+YB(i));                   % Compute edge values
end
sumEdge = sum(edge);                                             % Sum all edge values

% If panels are CCW, flip them (don't if CW)
if (sumEdge < 0)                                                 % If panels are CCW
    XB = flipud(XB);                                             % Flip the X-data array
    YB = flipud(YB);                                             % Flip the Y-data array
end
```

PANEL METHOD GEOMETRY

```
% Initialize variables
XC   = zeros(numPan,1);                                            % Initialize control
point X-coordinate array
YC   = zeros(numPan,1);                                            % Initialize control
point Y-coordinate array
S    = zeros(numPan,1);                                            % Initialize panel length
array
phiD = zeros(numPan,1);                                            % Initialize panel
orientation angle array [deg]

% Find geometric quantities of the airfoil
for i = 1:1:numPan                                                 % Loop over all panels
    XC(i)   = 0.5*(XB(i)+XB(i+1));                                 % X-value of control
point
    YC(i)   = 0.5*(YB(i)+YB(i+1));                                 % Y-value of control
point
    dx      = XB(i+1)-XB(i);                                       % Change in X between
boundary points
    dy      = YB(i+1)-YB(i);                                       % Change in Y between
boundary points
    S(i)    = (dx^2 + dy^2)^0.5;                                   % Length of the panel
    phiD(i) = atan2d(dy,dx);                                       % Angle of the panel
(positive X-axis to inside face) [deg]
    if (phiD(i) < 0)                                               % Make all panel angles
positive [deg]
        phiD(i) = phiD(i) + 360;
    end
end

% Compute angle of panel normal w.r.t horizontal and include AoA
deltaD             = phiD + 90;                                    % Angle from positive X-
axis to outward normal vector [deg]
betaD              = deltaD - AoA;                                 % Angle between
freestream vector and outward normal vector [deg]
betaD(betaD > 360) = betaD(betaD > 360) - 360;                     % Make all panel angles
between 0 and 360 [deg]

% Convert angles from [deg] to [rad]
phi  = phiD.*(pi/180);                                             % Convert from [deg] to
[rad]
beta = betaD.*(pi/180);                                            % Convert from [deg] to
[rad]
```

COMPUTE SOURCE AND VORTEX PANEL STRENGTHS

```
% Geometric integrals for SPM and VPM (normal [I,K] and tangential [J,L])
% - Refs [2], [3], [6], and [7]
[I,J] = COMPUTE_IJ_SPM(XC,YC,XB,YB,phi,S);                         % Call COMPUTE_IJ_SPM
function (Refs [2] and [3])
[K,L] = COMPUTE_KL_VPM(XC,YC,XB,YB,phi,S);                         % Call COMPUTE_KL_VPM
function (Refs [6] and [7])

% Populate A matrix
% - Simpler option: A = I + pi*eye(numPan,numPan);
A = zeros(numPan,numPan);                                          % Initialize the A matrix
for i = 1:1:numPan                                                 % Loop over all i panels
    for j = 1:1:numPan                                             % Loop over all j panels
        if (j == i)                                                % If the panels are the
```

```
same
            A(i,j) = pi;                                                  % Set A equal to pi
        else                                                              % If panels are not the
same
            A(i,j) = I(i,j);                                              % Set A equal to I
        end
    end
end

% Right column of A matrix
for i = 1:1:numPan                                                        % Loop over all i panels
(rows)
    A(i,numPan+1) = -sum(K(i,:));                                         % Add gamma term to
right-most column of A matrix
end

% Bottom row of A matrix (Kutta condition)
for j = 1:1:numPan                                                        % Loop over all j panels
(columns)
    A(numPan+1,j) = (J(1,j) + J(numPan,j));                               % Source contribution of
Kutta condition equation
end
A(numPan+1,numPan+1) = -sum(L(1,:) + L(numPan,:)) + 2*pi;                 % Vortex contribution of
Kutta condition equation

% Populate b array
% - Simpler option: b = -Vinf*2*pi*cos(beta);
b = zeros(numPan,1);                                                      % Initialize the b array
for i = 1:1:numPan                                                        % Loop over all i panels
(rows)
    b(i) = -Vinf*2*pi*cos(beta(i));                                       % Compute RHS array
end

% Last element of b array (Kutta condition)
b(numPan+1) = -Vinf*2*pi*(sin(beta(1)) + sin(beta(numPan)));              % RHS of Kutta condition
equation

% Compute result array
resArr = A\b;                                                             % Solve system of
equations for all source strengths and single vortex strength

% Separate lambda and gamma values from result array
lambda = resArr(1:end-1);                                                 % All panel source
strenths
gamma  = resArr(end);                                                     % Constant vortex
strength
```

COMPUTE PANEL VELOCITIES AND PRESSURE COEFFICIENTS

```
% Compute velocities on each panel
Vt = zeros(numPan,1);                                                     % Initialize tangential
velocity
Cp = zeros(numPan,1);                                                     % Initialize pressure
coefficient
for i = 1:1:numPan
    term1 = Vinf*sin(beta(i));                                            % Uniform flow term
    term2 = (1/(2*pi))*sum(lambda.*J(i,:)');                              % Source panel terms when
j is not equal to i
    term3 = gamma/2;                                                      % Vortex panel term when
j is equal to i
```

```
    term4 = -(gamma/(2*pi))*sum(L(i,:));                                % Vortex panel terms when
j is not equal to i

    Vt(i) = term1 + term2 + term3 + term4;                              % Compute tangential
velocity on panel i
    Cp(i) = 1-(Vt(i)/Vinf)^2;                                           % Compute pressure
coefficient on panel i
end
```

COMPUTE LIFT AND MOMENT

```
% Compute normal and axial force coefficients
CN = -Cp.*S.*sin(beta);                                                 % Normal force
coefficient []
CA = -Cp.*S.*cos(beta);                                                 % Axial force coefficient
[]

% Compute lift and moment coefficients
CL = sum(CN.*cosd(AoA)) - sum(CA.*sind(AoA));                           % Decompose axial and
normal to lift coefficient []
CM = sum(Cp.*(XC-0.25).*S.*cos(phi));                                   % Moment coefficient []

% Print the results to the Command Window
fprintf('======= RESULTS =======\n');
fprintf('Lift Coefficient (CL)\n');
fprintf('\tSPVP : %2.4f\n',CL);                                         % From this SPVP code
fprintf('\tK-J  : %g\n',2*sum(gamma.*S));                               % From Kutta-Joukowski
lift equation
fprintf('\tXFOIL: %2.4f\n',xFoilCL);                                    % From XFOIL program
fprintf('Moment Coefficient (CM)\n');
fprintf('\tSPVP : %2.4f\n',CM);                                         % From this SPVP code
fprintf('\tXFOIL: %2.4f\n',xFoilCM);                                    % From XFOIL program
```

```
======= RESULTS =======
Lift Coefficient (CL)
       SPVP : 0.0000
       K-J  : 7.62499e-05
       XFOIL: 0.0002
Moment Coefficient (CM)
       SPVP : -0.0000
       XFOIL: -0.0000
```

COMPUTE STREAMLINES

```
if (flagPlot(5) == 1 || flagPlot(6) == 1)
    % Grid parameters
    nGridX = 100;                                                       % X-grid for streamlines
and contours
    nGridY = 100;                                                       % Y-grid for streamlines
and contours
    xVals  = [min(XB)-0.5 max(XB)+0.5];                                 % X-grid extents [min,
max]
    yVals  = [min(YB)-0.3 max(YB)+0.3];                                 % Y-grid extents [min,
max]

    % Streamline parameters
    stepsize = 0.01;                                                    % Step size for
streamline propagation
    maxVert  = nGridX*nGridY*100;                                       % Maximum vertices
    slPct    = 25;                                                      % Percentage of
streamlines of the grid
    Ysl      = linspace(yVals(1),yVals(2),floor((slPct/100)*nGridY))';      % Create array of Y
streamline starting points

    % Generate the grid points
    Xgrid   = linspace(xVals(1),xVals(2),nGridX)';                          % X-values in evenly
spaced grid
    Ygrid   = linspace(yVals(1),yVals(2),nGridY)';                          % Y-values in evenly
spaced grid
    [XX,YY] = meshgrid(Xgrid,Ygrid);                                        % Create meshgrid from X
and Y grid arrays

    % Initialize velocities
    Vx = zeros(nGridX,nGridY);                                              % Initialize X velocity
matrix
    Vy = zeros(nGridX,nGridY);                                              % Initialize Y velocity
matrix

    % Solve for grid point X and Y velocities
    for m = 1:1:nGridX
        for n = 1:1:nGridY
            XP      = XX(m,n);                                              % Current iteration's X
grid point
            YP      = YY(m,n);                                              % Current iteration's Y
grid point
            [Mx,My] = STREAMLINE_SPM(XP,YP,XB,YB,phi,S);                    % Compute Mx and My
geometric integrals (Ref [4])
            [Nx,Ny] = STREAMLINE_VPM(XP,YP,XB,YB,phi,S);                    % Compute Nx and Ny
geometric integrals (Ref [8])

            [in,on] = inpolygon(XP,YP,XB,YB);
            if (in == 1 || on == 1)                                         % If the grid point is in
or on the airfoil
                Vx(m,n) = 0;                                                % Set X-velocity equal to
zero
                Vy(m,n) = 0;                                                % Set Y-velocity equal to
zero
            else                                                            % If the grid point is
outside the airfoil
                Vx(m,n) = Vinf*cosd(AoA) + sum(lambda.*Mx./(2*pi)) + ...    % Compute X-velocity
                            sum(-gamma.*Nx./(2*pi));
                Vy(m,n) = Vinf*sind(AoA) + sum(lambda.*My./(2*pi)) + ...    % Compute Y-velocity
                            sum(-gamma.*Ny./(2*pi));
            end
        end
    end

    % Compute grid point velocity magnitude and pressure coefficient
    Vxy  = sqrt(Vx.^2 + Vy.^2);                                             % Compute magnitude of
velocity vector []
    CpXY = 1-(Vxy./Vinf).^2;                                                % Pressure coefficient []
end
```

PLOTTING

```
% FIGURE: Airfoil with panel normal vectors
if (flagPlot(1) == 1)
    figure(1);                                                              % Create figure
    cla; hold on; grid off;                                                 % Get ready for plotting
    set(gcf,'Color','White');                                     % Set color to white
    set(gca,'FontSize',12);                                       % Set font size
    fill(XB,YB,'k');                                              % Plot airfoil
    for i = 1:1:numPan                                            % Loop over all panels
        X(1) = XC(i);                                             % Set X start of panel
orientation vector
        X(2) = XC(i) + S(i)*cosd(betaD(i)+AoA);                   % Set X end of panel
orientation vector
        Y(1) = YC(i);                                             % Set Y start of panel
orientation vector
        Y(2) = YC(i) + S(i)*sind(betaD(i)+AoA);                   % Set Y end of panel
orientation vector
        plot(X,Y,'r-','LineWidth',2);                             % Plot panel normal
vector
    end
    xlabel('X Units');                                            % Set X-label
    ylabel('Y Units');                                            % Set Y-label
    xlim('auto');                                                 % Set X-axis limits to
auto
    ylim('auto');                                                 % Set Y-axis limits to
auto
    title(['NACA ' xFoilResults.afName  ' Airfoil' ...            % Title
           ' with panel normal vectors'])
    axis equal;                                                   % Set axes equal
    zoom reset;                                                   % Reset zoom
end

% FIGURE: Geometry with the following indicated:
% - Boundary pts, control pts, first panel, second panel
if (flagPlot(2) == 1)
    figure(2);                                                    % Create figure
    cla; hold on; grid on;                                        % Get ready for plotting
    set(gcf,'Color','White');                                     % Set color to white
    set(gca,'FontSize',12);                                       % Set font size
    plot(XB,YB,'k-','LineWidth',3);                               % Plot airfoil panels
    p1 = plot([XB(1) XB(2)],[YB(1) YB(2)],'g-','LineWidth',2);    % Plot first panel
    p2 = plot([XB(2) XB(3)],[YB(2) YB(3)],'m-','LineWidth',2);    % Plot second panel
    pB = plot(XB,YB,'ko','MarkerFaceColor','k');                  % Plot boundary points
(black circles)
    pC = plot(XC,YC,'ko','MarkerFaceColor','r');                  % Plot control points
(red circles)
    legend([pB,pC,p1,p2],...                                      % Show legend
           {'Boundary','Control','First Panel','Second Panel'});
    xlabel('X Units');                                            % Set X-label
    ylabel('Y Units');                                            % Set Y-label
    xlim('auto');                                                 % Set X-axis limits to
auto
    ylim('auto');                                                 % Set Y-axis limits to
auto
    title(['NACA ' xFoilResults.afName  ' Airfoil Geometry' ])    % Title
    axis equal;                                                   % Set axes equal
    zoom reset;                                                   % Reset zoom
end

% FIGURE: Cp vectors at airfoil control points
if (flagPlot(3) == 1)
    figure(3);                                                    % Create figure
    cla; hold on; grid on;                                        % Get ready for plotting
    set(gcf,'Color','White');                                     % Set color to white
    set(gca,'FontSize',12);                                       % Set font size
    Cps = abs(Cp*0.25);                                           % Scale and make positive
```

```
all Cp values
    for i = 1:1:length(Cps)                                             % Loop over all panels
        X(1) = XC(i);                                                   % Control point X-
coordinate
        X(2) = XC(i) + Cps(i)*cosd(betaD(i)+AoA);                       % Ending X-value based on
Cp magnitude
        Y(1) = YC(i);                                                   % Control point Y-
coordinate
        Y(2) = YC(i) + Cps(i)*sind(betaD(i)+AoA);                       % Ending Y-value based on
Cp magnitude

        if (Cp(i) < 0)                                                  % If pressure coefficient
is negative
            p{1} = plot(X,Y,'r-','LineWidth',2);                               % Plot as a red
line
        elseif (Cp(i) >= 0)                                             % If pressure coefficient
is zero or positive
            p{2} = plot(X,Y,'b-','LineWidth',2);                               % Plot as a blue
line
        end
    end
    fill(XB,YB,'k');                                                    % Plot the airfoil as
black polygon
    legend([p{1},p{2}],{'Negative Cp','Positive Cp'});                  % Show legend
    xlabel('X Units');                                                  % Set X-label
    ylabel('Y Units');                                                  % Set Y-label
    xlim('auto');                                                       % Set X-axis limits to
auto
    ylim('auto');                                                       % Set Y-axis limits to
auto
    title(['NACA ' xFoilResults.afName  ...                             % Title
        ' Cp vectors at airfoil control points' ])
    axis equal;                                                         % Set axes equal
    zoom reset;                                                         % Reset zoom
end

% FIGURE: Pressure coefficient comparison (XFOIL vs. VPM)
if (flagPlot(4) == 1)
    figure(4);                                                          % Create figure
    cla; hold on; grid on;                                              % Get ready for plotting
    set(gcf,'Color','White');                                           % Set color to white
    set(gca,'FontSize',12);                                             % Set font size
    midIndX = floor(length(xFoilCP)/2);                                 % Airfoil middle index
for XFOIL data
    midIndS = floor(length(Cp)/2);                                      % Airfoil middle index
for SPM data
    pXu = plot(xFoilX(1:midIndX),xFoilCP(1:midIndX),'b-','LineWidth',2);    % Plot Cp for upper
surface of airfoil from XFOIL
    pXl = plot(xFoilX(midIndX+1:end),xFoilCP(midIndX+1:end),'r-',...        % Plot Cp for lower
surface of airfoil from XFOIL
                    'LineWidth',2);
    pVl = plot(XC(1:midIndS),Cp(1:midIndS),'ks','MarkerFaceColor','r');    % Plot Cp for upper
surface of airfoil from SPM
    pVu = plot(XC(midIndS+1:end),Cp(midIndS+1:end),'ks',...                 % Plot Cp for lower
surface of airfoil from SPM
                    'MarkerFaceColor','b');
    legend([pXu,pXl,pVu,pVl],...                                        % Show legend
           {'XFOIL Upper','XFOIL Lower','VPM Upper','VPM Lower'});
    xlabel('X Coordinate');                                             % Set X-label
    ylabel('Cp');                                                       % Set Y-label
    xlim([0 1]);                                                        % Set X-axis limits
    ylim('auto');                                                        % Set Y-axis limits to
auto
    set(gca,'Ydir','reverse')                                            % Reverse direction of Y-
axis
    title(['Airfoil: ' xFoilResults.afName ...                           % Title
           ', CL_{VPM}/CL_{XFOIL} = ' ...
           num2str((2*sum(gamma.*S)),4) '/' num2str(xFoilCL,4)]);
    zoom reset;                                                          % Reset zoom
end

% FIGURE: Airfoil streamlines
if (flagPlot(5) == 1)
    figure(5);                                                           % Create figure
    cla; hold on; grid on;                                               % Get ready for plotting
    set(gcf,'Color','White');                                            % Set color to white
    set(gca,'FontSize',12);                                              % Set font size
    for i = 1:1:length(Ysl)                                              % Loop over all Y
streamline starting points
        sl = streamline(XX,YY,Vx,Vy,xVals(1),Ysl(i),[stepsize,maxVert]);  % Plot the streamline
        set(sl,'LineWidth',2);                                           % Set streamline line
width
    end
    fill(XB,YB,'k');                                                     % Plot airfoil as black
polygon
    xlabel('X Units');                                                   % Set X-label
    ylabel('Y Units');                                                   % Set Y-label
    xlim(xVals);                                                         % Set X-axis limits
    axis equal;                                                          % Set axes equal
    ylim(yVals);                                                         % Set Y-axis limits
    title(['NACA ' xFoilResults.afName  ' Airfoil Streamlines' ])        % Title
    zoom reset;                                                          % Reset zoom
end

% FIGURE: Pressure coefficient contour
if (flagPlot(6) == 1)
    figure(6);                                                           % Create figure
    cla; hold on; grid on;                                               % Get ready for plotting
    set(gcf,'Color','White');                                            % Set color to white
    set(gca,'FontSize',12);                                              % Set font size
    contourf(XX,YY,CpXY,100,'EdgeColor','none');                         % Plot Cp contour
    fill(XB,YB,'k');                                                     % Plot airfoil as black
polygon
    xlabel('X Units');                                                   % Set X-label
    ylabel('Y Units');                                                   % Set Y-label
    xlim(xVals);                                                         % Set X-axis limits
    axis equal;                                                          % Set axes equal
    ylim(yVals);                                                         % Set Y-axis limits
    title(['NACA ' xFoilResults.afName  ...
        ' Airfoil Pressure coefficient contour' ])                       % Title
    zoom reset;                                                          % Reset zoom
end

X_Coordinate=[0.2; 0.4; 0.6; 0.8; XC(1)];
v_index=zeros(length(X_Coordinate),1);
v_tangential=zeros(length(X_Coordinate),1);
for i = 1:1:length(X_Coordinate)
v_index(i)=find(XC>=X_Coordinate(i) & XC<X_Coordinate(i)+0.02,1,'last' );
v_tangential(i)=Vt(v_index(i)-1)+((X_Coordinate(i)-XC(v_index(i)-1))/(XC(v_index(i))-XC(v_index(i)-
1)))*(Vt(v_index(i))-Vt(v_index(i)-1));
end
fprintf('~~~~~~ Tangential velocities at specified X coordinates ~~~~~~\n');
```

```
disp(table(X_Coordinate,v_tangential))
```

```
~~~~~~ Tangential velocities at specified X coordinates ~~~~~~
    X_Coordinate    v_tangential
    ____________    ____________

          0.2          63.276
          0.4          59.756
          0.6          55.787
          0.8          51.505
      0.99968          32.875
```

Appendix B: XFOIL.m

```
function [xFoilResults,success] = XFOIL(NACA,PPAR,AoA,flagAirfoil)
```

```
% PURPOSE
% - Create or load airfoil based on flagAirfoil
% - Save and read airfoil coordinates
% - Save and read airfoil pressure coefficient
% - Save and read airfoil lift, drag, and moment coefficients
%
% INPUTS
% - NACA        : Four-digit NACA airfoil designation
% - PPAR        : Paneling variables used in XFOIL PPAR menu
% - AoA         : Angle of attack [deg]
% - flagAirfoil  : Flag for loading/creating airfoil
%
% OUTPUTS
% - xFoilResults : Structure containing all results
% - success      : Flag indicating whether solution was a success
```

CALL XFOIL FROM MATLAB

```
xFoilResults = [];                                                          % Initialize results
structure

if (flagAirfoil.XFoilCreate == 1)                                           % If the user wants XFOIL
to create a NACA airfoil
    airfoilName          = NACA;                                            % Set the airfoilName to
the input NACA airfoil
    xFoilResults.afName = airfoilName;                                      % Send the airfoil name
back from this function
    success             = 1;                                                % This will be successful
elseif (flagAirfoil.XFoilLoad == 1)                                         % If the user wnats to
load a DAT file airfoil
    [flnm,~,success]     = uigetfile('./Airfoil_DAT_Selig/*.dat',...        % User input of airfoil
file to load
                                    'Select Airfoil File');
    airfoilName          = flnm(1:end-4);                                   % Set the airfoilName
based on loaded file
    xFoilResults.afName = airfoilName;                                      % Send the airfoil name
back from this function
    if (success == 0)                                                       % If the user exited
dialog box without selecting airfoil
        return;                                                             % Exit the function
    else                                                                    % If user selected an
```

```
airfoil
        success = 1;                                                    % This will be successful
    end
end

% Save-to file names
saveFlnm    = ['Save_' airfoilName '.txt'];                             % Airfoil coordinates
save-to file
saveFlnmCp  = ['Save_' airfoilName '_Cp.txt'];                          % Airfoil Cp save-to file
saveFlnmPol = ['Save_' airfoilName '_Pol.txt'];                         % Airfoil polar save-to
file

% Delete files if they exist
if (exist(saveFlnm,'file'))                                             % If airfoil coordinate
file exists
    delete(saveFlnm);                                                   % Delete it
end
if (exist(saveFlnmCp,'file'))                                           % If airfoil Cp file
exists
    delete(saveFlnmCp);                                                 % Delete it
end
if (exist(saveFlnmPol,'file'))                                          % If airfoil polar file
exists
    delete(saveFlnmPol);                                                % Delete it
end

% Create the airfoil
fid = fopen('xfoil_input.inp','w');                                     % Create an XFoil input
file, and make it write-able
if (flagAirfoil.XFoilLoad == 1)                                         % If user wants to load
DAT airfoil file
    fprintf(fid,['LOAD ' './Airfoil_DAT_Selig/' flnm '\n']);            % Load selected airfoil
elseif (flagAirfoil.XFoilCreate == 1)                                   % If user wants to
specify a 4-digit airfoil
    fprintf(fid,['NACA ' NACA '\n']);                                   % Specify NACA airfoil
end
fprintf(fid,'PPAR\n');                                                  % Enter the PPAR
(paneling) menu
fprintf(fid,['N ' PPAR.N '\n']);                                        % Define "Number of panel
nodes"
fprintf(fid,['P ' PPAR.P '\n']);                                        % Define "Panel bunching
paramter"
fprintf(fid,['T ' PPAR.T '\n']);                                        % Define "TE/LE panel
density ratios"
fprintf(fid,['R ' PPAR.R '\n']);                                        % Define "Refined area/LE
panel density ratio"
fprintf(fid,['XT ' PPAR.XT '\n']);                                      % Define "Top side
refined area x/c limits"
fprintf(fid,['XB ' PPAR.XB '\n']);                                      % Define "Bottom side
refined area x/c limits"
fprintf(fid,'\n');                                                      % Apply all changes
fprintf(fid,'\n');                                                      % Back out to XFOIL menu

% Save the airfoil data points
fprintf(fid,['PSAV ' saveFlnm '\n']);                                   % Save the airfoil
coordinate file

% Get Cp and polar data
fprintf(fid,'OPER\n');                                                  % Enter OPER menu
fprintf(fid,'Pacc 1 \n');                                               % Begin polar
accumulation
```

```
fprintf(fid,'\n\n');                                            % Don't enter save or
dump file names
fprintf(fid,['Alfa ' num2str(AoA) '\n']);                       % Set angle of attack
fprintf(fid,['CPWR ' saveFlnmCp '\n']);                         % Write the Cp file
fprintf(fid,'PWRT\n');                                          % Save the polar data
fprintf(fid,[saveFlnmPol '\n']);                                % Save polar data to this
file
if (exist(saveFlnmPol) ~= 0)                                    % If saveFlnmPol already
exists
    fprintf(fid,'y \n');                                        % Overwrite existing file
end

fclose(fid);                                                    % Close the input file

cmd = 'xfoil.exe < xfoil_input.inp';                            % Write command to run
XFoil

[~,~] = system(cmd);                                            % Run XFoil with the
input file

fclose all;                                                     % Close all files

% Delete the XFoil input file
if (exist('xfoil_input.inp','file'))                            % If the input file
exists
    delete('xfoil_input.inp');                                  % Delete the file
end
```

READ CP DATA

```
fidCP = fopen(saveFlnmCp);                                      % Open the airfoil Cp
file
dataBuffer = textscan(fidCP,'%f %f %f','HeaderLines',3,...      % Read in X, Y, and Cp
data
                                     'CollectOutput',1,...
                                     'Delimiter','');
fclose(fidCP);                                                  % Close the file
delete(saveFlnmCp);                                             % Delete the file

% Save airfoil Cp data to function solution variable
xFoilResults.X  = dataBuffer{1,1}(:,1);                         % X-data points
xFoilResults.Y  = dataBuffer{1,1}(:,2);                         % Y-data points
xFoilResults.CP = dataBuffer{1,1}(:,3);                         % Cp data
```

READ AIRFOIL COORDINATES

```
fidAirfoil = fopen(saveFlnm);                                   % Open the airfoil file

dataBuffer = textscan(fidAirfoil,'%f %f','CollectOutput',1,...  % Read the XB and YB data
from the data file
                                  'Delimiter','','HeaderLines',0);

XB = dataBuffer{1}(:,1);                                        % Boundary point X-
coordinate
YB = dataBuffer{1}(:,2);                                        % Boundary point Y-
coordinate
fclose(fidAirfoil);                                             % Close the airfoil file
delete(saveFlnm);                                               % Delete the airfoil file
```

```
% Save airfoil boundary points to function solution variable
xFoilResults.XB = XB;                                                   % Airfoil boundary X-
points
xFoilResults.YB = YB;                                                   % Airfoil boundary Y-
points
```

READ POLAR DATA

```
% Load and read polar file (Save_Polar.txt)
fidPolar = fopen(saveFlnmPol);                                          % Open polar data file
for reading

dataBuffer = textscan(fidPolar,'%f %f %f %f %f %f %f','CollectOutput',1,... % Read data from file
                                 'Delimiter','','HeaderLines',12);
fclose(fidPolar);                                                       % Close the file
delete(saveFlnmPol);                                                    % Delete the file

% Extract polar data from buffer into function solution variable
xFoilResults.CL = dataBuffer{1,1}(2);                                   % Lift coefficient
xFoilResults.CD = dataBuffer{1,1}(3);                                   % Drag coefficient
xFoilResults.CM = dataBuffer{1,1}(5);                                   % Moment coefficient
```

Appendix C: COMPUTE_IJ_SPM.m

```
function [I,J] = COMPUTE_IJ_SPM(XC,YC,XB,YB,phi,S)

% PURPOSE
% - Compute the integral expression for constant strength source panels
% - Source panel strengths are constant, but can change from panel to panel
% - Geometric integral for panel-normal    : I(ij)
% - Geometric integral for panel-tangential: J(ij)

% INPUTS
% - XC  : X-coordinate of control points
% - YC  : Y-coordinate of control points
% - XB  : X-coordinate of boundary points
% - YB  : Y-coordinate of boundary points
% - phi : Angle between positive X-axis and interior of panel
% - S   : Length of panel
%
% OUTPUTS
% - I   : Value of panel-normal integral (Eq. 3.163 in Anderson or Ref [1])
% - J   : Value of panel-tangential integral (Eq. 3.165 in Anderson or Ref [2])

% Number of panels
numPan = length(XC);                                                    % Number of
panels/control points

% Initialize arrays
I = zeros(numPan,numPan);                                               % Initialize I integral
matrix
J = zeros(numPan,numPan);                                               % Initialize J integral
matrix

% Compute integral
for i = 1:1:numPan                                                      % Loop over i panels
    for j = 1:1:numPan                                                  % Loop over j panels
        if (j ~= i)                                                     % If the i and j panels
```

```
are not the same
            % Compute intermediate values
            A  = -(XC(i)-XB(j))*cos(phi(j))-(YC(i)-YB(j))*sin(phi(j));      % A term
            B  = (XC(i)-XB(j))^2+(YC(i)-YB(j))^2;                           % B term
            Cn = sin(phi(i)-phi(j));                                        % C term (normal)
            Dn = -(XC(i)-XB(j))*sin(phi(i))+(YC(i)-YB(j))*cos(phi(i));      % D term (normal)
            Ct = -cos(phi(i)-phi(j));                                       % C term (tangential)
            Dt = (XC(i)-XB(j))*cos(phi(i))+(YC(i)-YB(j))*sin(phi(i));       % D term (tangential)
            E  = sqrt(B-A^2);                                               % E term
            if (~isreal(E))
                E = 0;
            end

            % Compute I (needed for normal velocity), Ref [1]
            term1  = 0.5*Cn*log((S(j)^2+2*A*S(j)+B)/B);                     % First term in I
equation
            term2  = ((Dn-A*Cn)/E)*(atan2((S(j)+A),E) - atan2(A,E));        % Second term in I
equation
            I(i,j) = term1 + term2;                                         % Compute I integral

            % Compute J (needed for tangential velocity), Ref [2]
            term1  = 0.5*Ct*log((S(j)^2+2*A*S(j)+B)/B);                     % First term in J
equation
            term2  = ((Dt-A*Ct)/E)*(atan2((S(j)+A),E) - atan2(A,E));        % Second term in J
equation
            J(i,j) = term1 + term2;                                         % Compute J integral
        end

        % Zero out any NANS, INFs, or imaginary numbers
        if (isnan(I(i,j)) || isinf(I(i,j)) || ~isreal(I(i,j)))
            I(i,j) = 0;
        end
        if (isnan(J(i,j)) || isinf(J(i,j)) || ~isreal(J(i,j)))
            J(i,j) = 0;
        end
    end
end
```

Appendix D: COMPUTE_KL_VPM.m

```
function [K,L] = COMPUTE_KL_VPM(XC,YC,XB,YB,phi,S)

% FUNCTION - COMPUTE K AND L GEOMETRIC INTEGRALS FOR VORTEX PANEL METHOD

% PURPOSE
% - Compute the integral expression for constant strength vortex panels
% - Vortex panel strengths are constant, but can change from panel to panel
% - Geometric integral for panel-normal    : K(ij)
% - Geometric integral for panel-tangential: L(ij)

% INPUTS
% - XC  : X-coordinate of control points
% - YC  : Y-coordinate of control points
% - XB  : X-coordinate of boundary points
% - YB  : Y-coordinate of boundary points
% - phi : Angle between positive X-axis and interior of panel
% - S   : Length of panel
%
% OUTPUTS
% - K   : Value of panel-normal integral (Ref [1])
% - L   : Value of panel-tangential integral (Ref [2])

% Number of panels
numPan = length(XC);                                                  % Number of panels

% Initialize arrays
K = zeros(numPan,numPan);                                             % Initialize K integral
matrix
L = zeros(numPan,numPan);                                             % Initialize L integral
matrix

% Compute integral
for i = 1:1:numPan                                                    % Loop over i panels
    for j = 1:1:numPan                                                % Loop over j panels
        if (j ~= i)                                                   % If panel j is not the
same as panel i
            A  = -(XC(i)-XB(j))*cos(phi(j))-(YC(i)-YB(j))*sin(phi(j));      % A term
            B  = (XC(i)-XB(j))^2+(YC(i)-YB(j))^2;                     % B term
            Cn = -cos(phi(i)-phi(j));                                 % C term (normal)
            Dn = (XC(i)-XB(j))*cos(phi(i))+(YC(i)-YB(j))*sin(phi(i));       % D term (normal)
            Ct = sin(phi(j)-phi(i));                                  % C term (tangential)
            Dt = (XC(i)-XB(j))*sin(phi(i))-(YC(i)-YB(j))*cos(phi(i));       % D term (tangential)
            E  = sqrt(B-A^2);                                         % E term
            if (~isreal(E))
                E = 0;
            end

            % Compute K
            term1  = 0.5*Cn*log((S(j)^2+2*A*S(j)+B)/B);               % First term in K
equation
            term2  = ((Dn-A*Cn)/E)*(atan2((S(j)+A),E)-atan2(A,E));    % Second term in K
equation
            K(i,j) = term1 + term2;                                   % Compute K integral

            % Compute L
            term1  = 0.5*Ct*log((S(j)^2+2*A*S(j)+B)/B);               % First term in L
equation
            term2  = ((Dt-A*Ct)/E)*(atan2((S(j)+A),E)-atan2(A,E));    % Second term in L
equation
            L(i,j) = term1 + term2;                                   % Compute L integral
        end

        % Zero out any NANs, INFs, or imaginary numbers
        if (isnan(K(i,j)) || isinf(K(i,j)) || ~isreal(K(i,j)))
            K(i,j) = 0;
        end
        if (isnan(L(i,j)) || isinf(L(i,j)) || ~isreal(L(i,j)))
            L(i,j) = 0;
        end
    end
end
```

Appendix E: STREAMLINE_SPM.m

```
function [Mx,My] = STREAMLINE_SPM(XP,YP,XB,YB,phi,S)

% FUNCTION - COMPUTE Mx AND My GEOMETRIC INTEGRALS FOR SOURCE PANEL METHOD
% PURPOSE
% - Compute the geometric integral at point P due to source panels
% - Source panel strengths are constant, but can change from panel to panel
% - Geometric integral for X-direction: Mx(pj)
% - Geometric integral for Y-direction: My(pj)

% INPUTS
% - XP     : X-coordinate of computation point, P
% - YP     : Y-coordinate of computation point, P
% - XB     : X-coordinate of boundary points
% - YB     : Y-coordinate of boundary points
% - phi    : Angle between positive X-axis and interior of panel
% - S      : Length of panel
%
% OUTPUTS
% - Mx     : Value of X-direction geometric integral (Ref [1])
% - My     : Value of Y-direction geometric integral (Ref [1])

% Number of panels
numPan = length(XB)-1;                                              % Number of
panels/control points

% Initialize arrays
Mx = zeros(numPan,1);                                               % Initialize Mx integral
array
My = zeros(numPan,1);                                               % Initialize My integral
array

% Compute Mx and My
for j = 1:1:numPan                                                  % Loop over the j panels
    % Compute intermediate values
    A  = -(XP-XB(j))*cos(phi(j))-(YP-YB(j))*sin(phi(j));           % A term
    B  = (XP-XB(j))^2+(YP-YB(j))^2;                                 % B term
    Cx = -cos(phi(j));                                              % C term (X-direction)
    Dx = XP - XB(j);                                                % D term (X-direction)
    Cy = -sin(phi(j));                                              % C term (Y-direction)
    Dy = YP - YB(j);                                                % D term (Y-direction)
    E  = sqrt(B-A^2);                                               % E term
    if (~isreal(E))
        E = 0;
    end

    % Compute Mx, Ref [1]
    term1 = 0.5*Cx*log((S(j)^2+2*A*S(j)+B)/B);                      % First term in Mx
equation
    term2 = ((Dx-A*Cx)/E)*(atan2((S(j)+A),E) - atan2(A,E));         % Second term in Mx
equation
    Mx(j) = term1 + term2;                                          % X-direction geometric
integral

    % Compute My, Ref [1]
    term1 = 0.5*Cy*log((S(j)^2+2*A*S(j)+B)/B);                      % First term in My
equation
    term2 = ((Dy-A*Cy)/E)*(atan2((S(j)+A),E) - atan2(A,E));         % Second term in My
equation
    My(j) = term1 + term2;                                          % Y-direction geometric
integral

    % Zero out any NANs, INFs, or imaginary numbers
    if (isnan(Mx(j)) || isinf(Mx(j)) || ~isreal(Mx(j)))
        Mx(j) = 0;
    end
    if (isnan(My(j)) || isinf(My(j)) || ~isreal(My(j)))
        My(j) = 0;
    end
end
```

Appendix F: STREAMLINE_VPM.m

```
function [Nx,Ny] = STREAMLINE_VPM(XP,YP,XB,YB,phi,S)

% FUNCTION - COMPUTE Nx AND Ny GEOMETRIC INTEGRALS FOR VORTEX PANEL METHOD

% PURPOSE
% - Compute the integral expression for constant strength vortex panels
% - Vortex panel strengths are constant, but can change from panel to panel
% - Geometric integral for X-velocity: Nx(pj)
% - Geometric integral for Y-velocity: Ny(pj)

% INPUTS
% - XP     : X-coordinate of computation point, P
% - YP     : Y-coordinate of computation point, P
% - XB     : X-coordinate of boundary points
% - YB     : Y-coordinate of boundary points
% - phi    : Angle between positive X-axis and interior of panel
% - S      : Length of panel
%
% OUTPUTS
% - Nx     : Value of X-direction geometric integral
% - Ny     : Value of Y-direction geometric integral

% Number of panels
numPan = length(XB)-1;                                              % Number of panels
(control points)

% Initialize arrays
Nx = zeros(numPan,1);                                               % Initialize Nx integral
array
Ny = zeros(numPan,1);                                               % Initialize Ny integral
array

% Compute Nx and Ny
for j = 1:1:numPan                                                  % Loop over all panels
    % Compute intermediate values
    A  = -(XP-XB(j))*cos(phi(j))-(YP-YB(j))*sin(phi(j));           % A term
    B  = (XP-XB(j))^2+(YP-YB(j))^2;                                 % B term
    Cx = sin(phi(j));                                               % Cx term (X-direction)
    Dx = -(YP-YB(j));                                               % Dx term (X-direction)
    Cy = -cos(phi(j));                                              % Cy term (Y-direction)
    Dy = XP-XB(j);                                                  % Dy term (Y-direction)
    E  = sqrt(B-A^2);                                               % E term
    if (~isreal(E))
        E = 0;
    end

    % Compute Nx
    term1 = 0.5*Cx*log((S(j)^2+2*A*S(j)+B)/B);                      % First term in Nx
equation
    term2 = ((Dx-A*Cx)/E)*(atan2((S(j)+A),E) - atan2(A,E));         % Second term in Nx
equation
```

```
    Nx(j) = term1 + term2;                                                      % Compute Nx integral

    % Compute Ny
    term1 = 0.5*Cy*log((S(j)^2+2*A*S(j)+B)/B);                                  % First term in Ny
equation
    term2 = ((Dy-A*Cy)/E)*(atan2((S(j)+A),E) - atan2(A,E));                     % Second term in Ny
equation
    Ny(j) = term1 + term2;                                                      % Compute Ny integral

       % Zero out any NANs, INFs, or imaginary numbers
    if (isnan(Nx(j)) || isinf(Nx(j)) || ~isreal(Nx(j)))
        Nx(j) = 0;
    end
    if (isnan(Ny(j)) || isinf(Ny(j)) || ~isreal(Ny(j)))
        Ny(j) = 0;
    end
end
```