AE244
Assignment 2

Anuttar Jain
22B0003

March 14th 2024

Team Introduction & Contributions

S.No.	Name	Roll No.	Contribution Level	Contribution Details
1.	Anuttar Jain	22B0003	5	$\gamma(\theta)$ function, Vector Plot Total Circulation
2.	Wadate Vivek Kundlik	22B0012	5	$\gamma(\theta)$ function, An() function, Cl Value
3.	Ishika Biswas	22B0036	5	Camber line plot, Camber line slopes Main function

Algorithm

Plotting Camber Line

- Note:
  theta, x, y are numpy arrays of length N (N = 1001 in our case)
  theta stores the values of parametric coordinate θ
  x stores the x-coordinates of the camber line points
  y stores the y-coordinates of the camber line points
  m & p are variables used to store the maximum camber & position of maximum camber along the chord respectively
  I am using built-in python library functions in the pseudo code
  I am using my own created functions without going into their functioning as I will explain their algorithm later.
  The following equation is used to find the points on the camber line.

$$y_c = \begin{cases} \dfrac{m}{p^2}\left(2px - x^2\right), & 0 \le x \le p, \\ \dfrac{m}{(1-p)^2}\left((1-2p) + 2px - x^2\right), & p \le x \le 1, \end{cases}$$

Figure 1: camber line equation of a NACA airfoil

- Pseudo-Code for plotting:

```
theta ← numpy.linspace(0, $\pi$, N)
x ← 0.5( 1 - cos(theta))
y ← f_NACA(x,m,p)
pyplot.plot(x, y)
```

- Pseudo-Code for f_NACA(x,m,p):

```
y ← numpy.zeros(N)
i ← 0
while i < N {
    if x[i] ≤ p {
        y[i] ← (m/p$^2$)(2px[i] - x[i]$^2$)
    }
    else {
        y[i] ← (m/(1-p)$^2$)(1-2p + 2px[i] - x[i]$^2$)
    }
    i ← i + 1
}
return y
```

Function to find Slope of Camber Line

- Note:
  I created 2 functions related to this-
  One (derivative function) computes the slope at all points and stores them in a numpy array.
  Other (slopeFindingFunc function) takes a point input from user and returns the slope corresponding to that point, using the already created dy array.
  dy stores the derivatives at all the points on camber-line

- Pseudo-Code for derivative(x,y) Function:

```
dy ← numpy.zeros(N)
dy[0] = (y[1]-y[0])/(x[1]-x[0])
i ← 1
while i < N {
    dy[i] ← (y[i] - y[i-1])/(x[i] - x[i-1])
}
return dy
```

- Pseudo-Code for slopeFindingFunc(x,dy) Function:

```
px ← input()
i ← 0
while i < N {
    if x[i] ≤ px and x[i+1] > px {
        return dy[i]  }
    print ("invalid point")
}
```

Functions used to find Cl

- Note:
  Cl is calculated using the following result.

$$C_{\mathrm{L}} = \pi(2A_0 + A_1)$$

  The fourier coefficients are calculted using the following results

$$A_0 = \alpha - \frac{1}{\pi}\int_0^{\pi} \frac{dy}{dx}\, d\theta$$

$$A_{\mathrm{n}} = \frac{2}{\pi}\int_0^{\pi} \frac{dy}{dx} cos(n\theta)\, d\theta$$

  A_n values are calculated from n = 0 till n = M-1 (M = 200 in our case) since calculating till infinity is not possible
  An is a numpy array of size M used to store these calculated values
  alpha is the angle of attack

- Pseudo-Code for Cl(An) function:

```
Cl ← $\pi$( 2An[0] + An[1])
return Cl
```

- Pseudo-Code for An(dy,theta,N,M,alpha) function:

```
An ← numpy.zeros(N)
r ← 0
i ← 0
while i < N {
r ← r + dy[i]
}
An[0] ← alpha - (1/$\pi$)*r*($\pi$/N)
i ← i + 1
while i < M {
j ← 0
r ← 0
while j < N {
r ← dy[j]*cos(i*theta[j])
j ← j + 1
}
An[i] = (2/$\pi$)*r*($\pi$/N)
i ← i + 1
}
return An
```

Velocity Field around the airfoil

- Note:
  The function first creates a grid of points where the velocity vectors are to be plotted. This grid is created using the meshgrid function of numpy library. After this, for each element in the grid, the induced velocity is calculated using the Biot-Savart Law.

$$dv_{\text{induced}} = \frac{\gamma(\theta)}{4\pi}\frac{\vec{dl}\times\vec{r}}{r^3}$$

  We consider the span of the airfoil to be infinite. In such case, there will be a induced velocity component from each $\vec{dl}$ component along the span. The contribution from each such components leads to an integral. This is computed mathematically which leads to the following result of net induced velocity from one point on chord.

$$v_{\text{induced}} = \frac{\gamma(\theta)}{2\pi r}$$

  Now, the total induced velocity will be the integral of the velocities induced from all the different points on the chord. We assumed that on a small length $d\zeta$ along the chord, $\gamma(\theta)$ remains constant. So, We get the following integral.

$$\vec{v}_{\text{induced}} = \int_0^{\pi} \frac{\gamma(\theta)}{2\pi r}(-\vec{k}\times\vec{r})\frac{sin(\theta)}{2}\,d\theta$$

  Finally, the net velocity at a point is found by summing the freestream and induced velocities.

$$v_{\text{x}} = Ucos(\alpha) + \int_0^{\pi} \frac{\gamma(\theta)}{2\pi r}cos(\delta)\frac{sin(\theta)}{2}\,d\theta$$

$$v_{\text{y}} = Ucos(\alpha) - \int_0^{\pi} \frac{\gamma(\theta)}{2\pi r}sin(\delta)\frac{sin(\theta)}{2}\,d\theta$$

  This integral is evaluated for every point in the grid numerically. We computed the x & y components directly using $cos(\delta)$ & $\sin(\delta)$ where δ is the slope angle of the the point in grid to the point on the chord considered. After integration, the freestream velocity is added to the net induced velocity to get the effective velocity at each point. After computing velocities for all points in the grid, we plot the resulting vector field using the quiver function of the pyplot library.
  In the following pseudo code, We've assumed that X[i][j] & Y[i][j] to give the x-component & y-component of the point at i^{th} row, and j^{th} column of the grid respectively. The size of grid is m×n. The u, v arrays store the x and y component of total velocity respectively at all the points on the grid.

- Pseudo-Code of vectorPlot(theta,x,y,gamma,v0,alpha) Function:

```
X, Y ← numpy.meshgrid( numpy.linspace(x1, x2, n), numpy.linspace(y1, y2, m) )
u = numpy.zeros_like(X)
v = numpy.zeros_like(X)
i ← 0
```

```
while i < m {
    j ← 0
    while j < n {
        k ← 0
        while k < N {
            r ← $\sqrt{((X[i][j] - x[k])^2 + (Y[i][j] - y[k])^2}$
            dzeta ← 0.5sin(theta[k])dtheta
            K ← gamma[k]/(2πr)
            cos_delta ← (X[i][j] - x[k])/r
            sin_delta ← (Y[i][j] - y[k])/r
            u[i][j] ← u[i][j] + K*sin_delta*dzeta
            v[i][j] ← v[i][j] - K*cos_delta*dzeta
            k ← k + 1
        }
        u[i][j] ← u[i][j] + v0*cos(alpha)
        v[i][j] ← v[i][j] + v0*sin(alpha)
        j ← j + 1
    }
    i ← i + 1
}
pyplot.quiver(X,Y,u,v)
```

Total Circulation Γ

- Note:
  Total Circulation is found via 2 methods-
  1. Via Velocity line Integral: Velocity Line integral over a closed curve (circle in our case) containing the camber line is computed to give the total circulation.

$$\Gamma = \oint_C \vec{V}.\vec{ds}$$

  2. Via Integrating vorticity $\gamma(\theta)$ at all points on camber line.

$$\Gamma = \int_0^{\pi} \frac{1}{2}\gamma(\theta)sin(\theta)\,d\theta$$

  $\gamma(\theta)$ is the vorticity strength at θ. It is found using its fourier expansion as per the following result.

$$\gamma(\theta) = 2U_\infty\left(A_0\frac{1+\cos\theta}{\sin\theta} + \sum_{n=1}^{\infty} A_n \sin n\theta\right)$$

Figure 2: Formula to calculate $\gamma(\theta)$

  We defined $\gamma(0) = \gamma(\pi) = 0$ as division by $\sin(0) = \sin(\pi) = 0$ occurs which is invalid.

  The Velocity Line integral is calculated over a circle centred at (0.5c, 0) and is of radius

1c. The coordinates of a point on circle is given in terms of a parametric coordinate β shown below.

$$(x, y) = (0.5c + ccos\beta, csin\beta)$$

The total velocity is found at each point on the curve, as discussed above, and the dot product is taken with a infinitely small element $\vec{ds}$ along the curve. This product is integrated over the entire cure to get the desired Total Circulation.

$$\Gamma = \int_0^{2\pi} -v_{\mathrm{x}} sin\beta + v_{\mathrm{y}} cos\beta \, d\beta$$

v0 is the freestream velocity
gamma is a numpy array that stores all the values of $\gamma(\theta)$

- Pseudo-Code for circulation_gammaIntegral(gamma, theta):

```
circ ← 0
i ← 0
while i < N {
    circ ← circ + 0.5*gamma[i]*sin(theta[i])*(π/N)
    i ← i + 1
}
return circ
```

- Pseudo-Code for circulation_velocityLineIntegral(x,y,gamma,theta,v0,alpha):

```
beta ← numpy.linspace(0, 2π, n)
x1 ← 0.5 + cos(beta)
y1 ← sin(beta)
u ← numpy.zeros(n) v ← numpy.zeros(n) circ ← 0 i ← 0 while i < n {
    j ← 0
    while j < N {
        r ← $\sqrt{((x1[i] - x[j])^2 + (y1[i] - y[j])^2}$
        dzeta ← 0.5sin(theta[j])dtheta
        K ← gamma[j]/(2πr)
        cos_delta ← (x1[i] - x[j])/r
        sin_delta ← (y1[i] - y[j])/r
        u[i] ← u[i] + K*sin_delta*dzeta
        v[i] ← v[i] - K*cos_delta*dzeta
        j ← j + 1
    }
    u[i] ← u[i] + v0*cos(alpha)
    v[i] ← v[i] + v0*sin(alpha)
    dsx ← (-1)*(2*π/N)*sin(beta[i])
    dsy ← (2*π/N)*cos(beta[i])
    circ ← circ + u[i]*dsx + v[i]*dsy
    i ← i + 1
}
return -circ
```

- Pseudo-Code for gamma_func(theta, An, N, M, v0):

```
gamma ← numpy.zeros(N)
gamma[0] ← 0
gamma[N-1]← 0
i ← 1
while i < N-1 {
    gamma[i] ← An[0]*(1+cos(theta[i])/sin(theta[i])
    j ← 1
    while j < M {
        gamma[i] ← gamma[i] + An[j]*sin(j*theta[i])
        j ← j + 1
    }
    gamma[i] = 2*v0*gamma[i]
    i ← i + 1
}
return gamma
```

Simulation on NACA Airfoil

Parameters

- maximum camber (m): 0.3%
- position of max camber (p): 12.5%

Plot of Camber Line

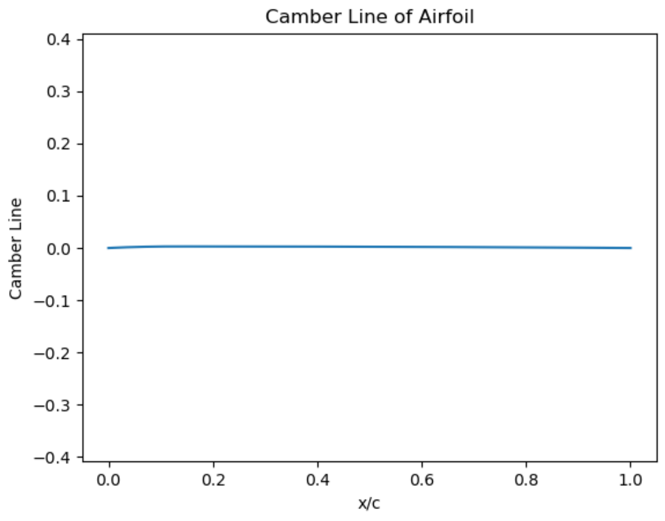

Figure 3: Equal Axes

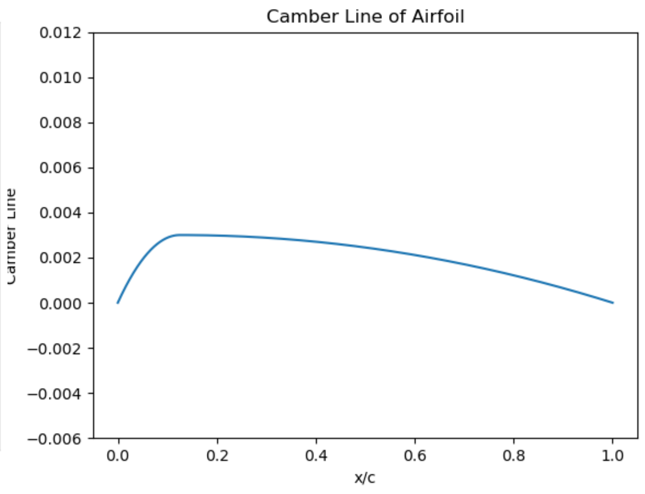

Figure 4: Unequal Axes

Plot of slope of the points in Camber Line

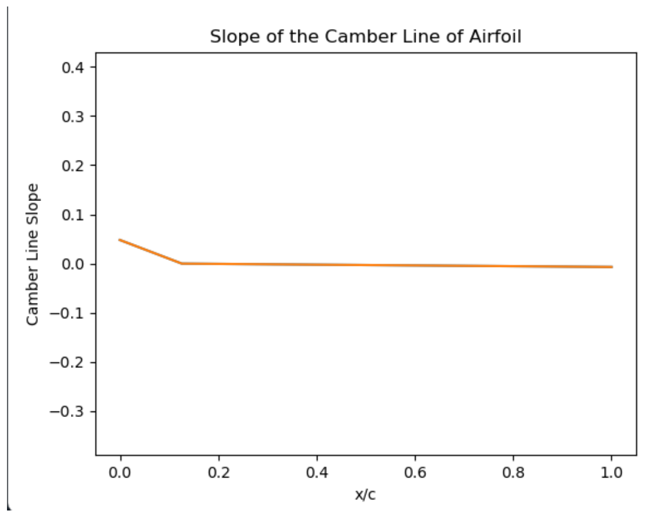

Figure 5: Equal Axes

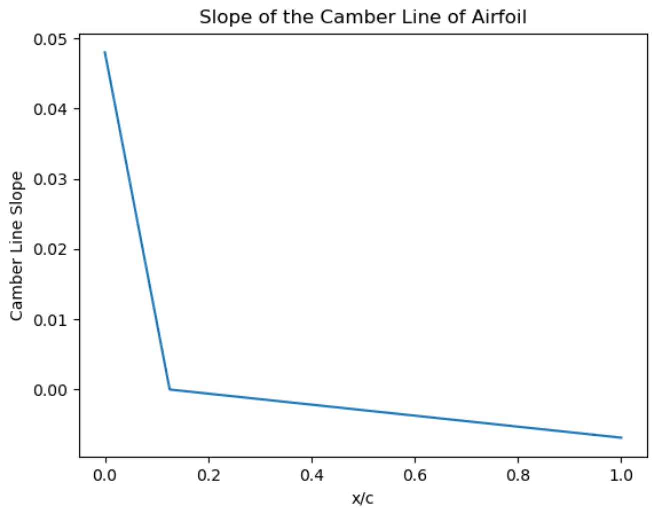

Figure 6: Unequal Axes

Plot of Cl vs Angle of Attack

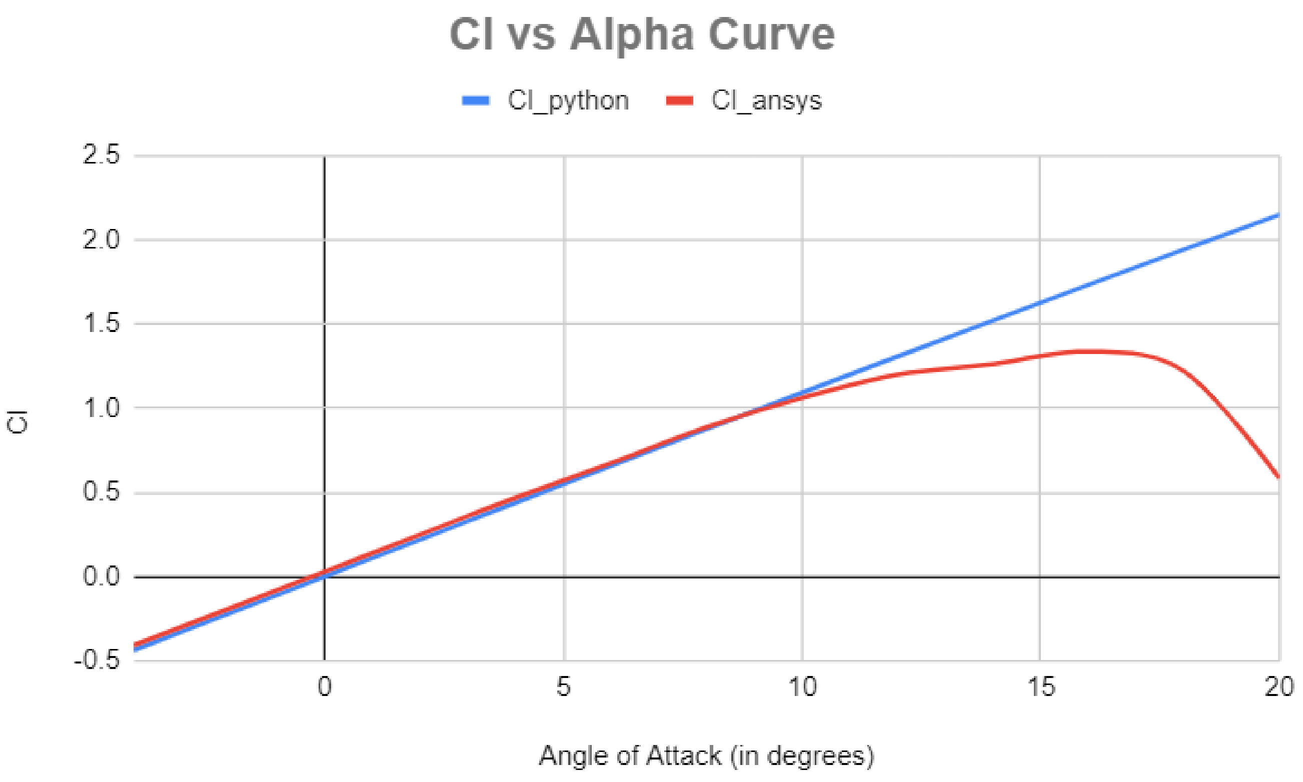

Figure 7: Cl vs Alpha Plot for both Ansys and Python simulation

Stalling occurs only when we consider the effects of viscous forces. Viscous forces lead to flow separation due to which stall happens. We allowed the effect of viscosity in our ansys simulation, which led to stalling condition at some high angle of attack, as observed from the plot. We neglected all viscous effects in the python simulation. Due to this, flow separation does not occur and hence there is no stall. Hence, we get a linear function between Cl and angle of attack.
Apart from stall, both the simulations yield same results! For the same angle of attack, both python and ansys simulation yield same Cl value. An advantage of using python simulation is clearly demonstrated as computation time for python simulation is in seconds while that of ansys simulation is in hours.

Velocity Field Plot

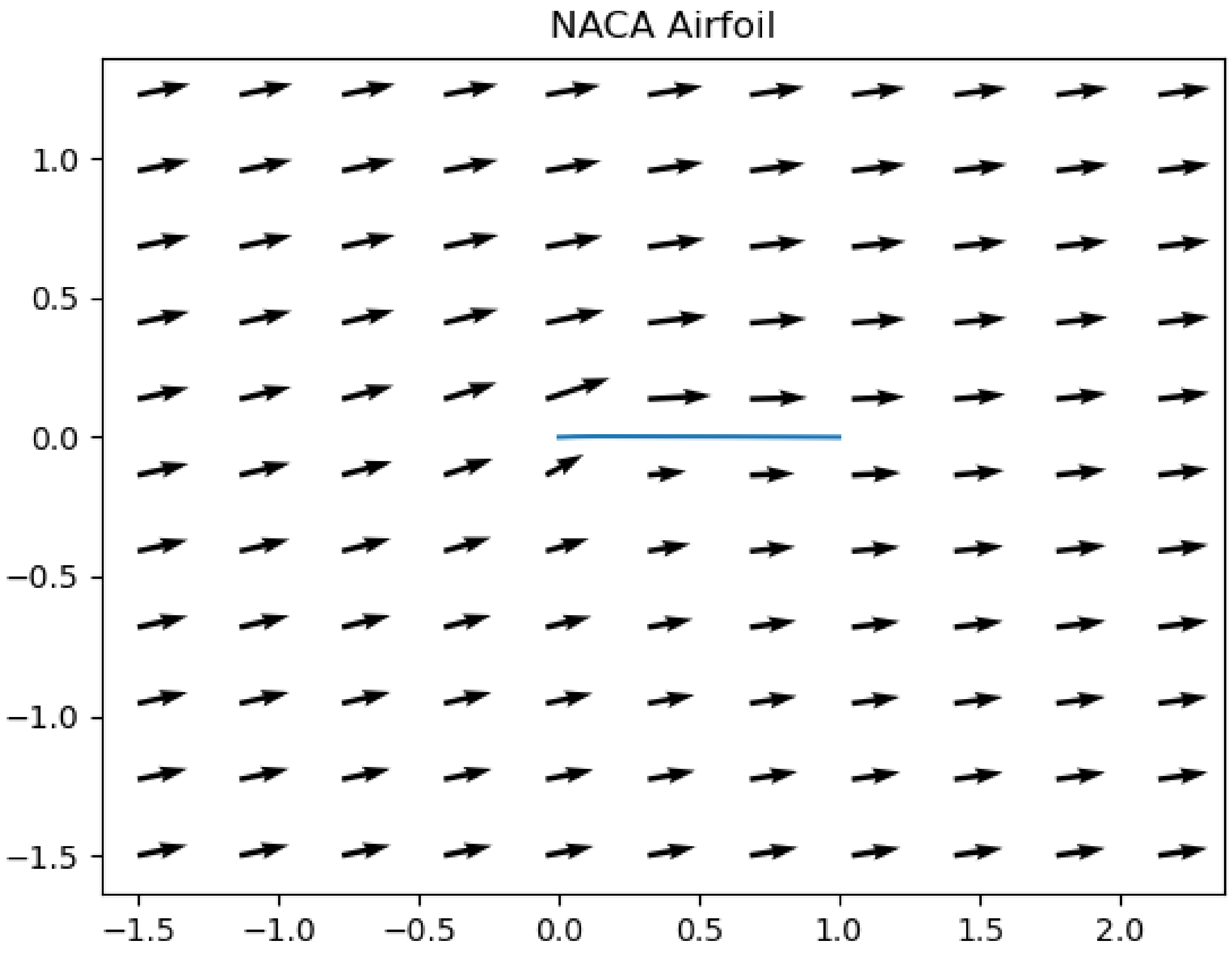

Figure 8: Velocity Field Plot

- Since the camber of my airfoil is very small (0.3%), it behaves similar to a thin plate. Due to this, the flow behaves nearly undisturbed. It accelerates a bit near the leading edge as observed from the longer arrows. Then the speed decreases as the air flows along the airfoil.

- The flow above the airfoil is faster compared to flow below the airfoil. This observation comes from the fact that the velocity vector arrows above the airfoil are longer compared to the ones below. This eventually means that on applying the Bernoulli's equation, pressure above the airfoil is less compared to pressure below the airfoil. Hence, a net upward force is experienced by the airfoil which we famously call lift. Drag is also generated as a consequence of this. But the skin friction drag is typically so small that is usually fine to neglect it for practical purposes.
- Also, the direction of airflow after passing the airfoil is slightly tilted below compared to before encountering the airfoil. This net downwash induced by the airfoil onto the airflow generates a reaction force onto the airfoil by the surrounding air which leads to lift.

Total Circulation
Total circulation is calculated via 2 methods:
1) Using points on camber line

$$\Gamma = \int_0^c \gamma(\zeta)d\zeta$$

2) Calculating the following over the boundary of a circle C containing the camber line. The circle has centre at (0.5c, 0) and has radius of 1c.

$$\Gamma = \oint_C \vec{U}.\vec{dl} + \oint_C \int_0^\pi \frac{\gamma(\theta)}{4\pi r^2} sin\theta(-\vec{k} \times \vec{r} d\theta.\vec{dl})$$

$$\Gamma^{(1)} = 5.352\ m^3/s$$

$$\Gamma^{(2)} = 5.346\ m^3/s$$

$$\Gamma^{(1)} \approx \Gamma^{(2)}$$

There exists a velocity potential function at all points where its derivative exists and is continuous. The presence of a potential function indicates that the flow is irrotational, i.e net circulation is zero. This is given by the following result.

$$\nabla \times \nabla\phi = 0$$

This potential function is absent only at the camber line. Due to this, there may exist a net non-zero circulation at the points in the camber line, which it does. Apart from these points, no circulation exists. Therefore, if we move along any closed path containing the camber line, the net circulation obtained around the path should be equal to the total circulation considering all the points on the camber line. Since the simulation results using both approaches led to exactly same results, hence the theoretical explanation is validated.

Novel Airfoil Analysis

Types of airfoils analysed
In all the following cases, assume that-

m: maximum camber
c: chord length
$0 \leq$ x $\leq$ c

- Airfoil with parabolic camber line.

$$y = -4mx(x - c)$$

- Airfoil with elliptical camber line

$$y = 2m\sqrt{\frac{x}{c} - \frac{x^2}{c^2}}$$

- Airfoil with hyperbolic camber line

$$y = (1 + m) - \sqrt{1 + (m^2 + 2m)(1 - \frac{2x}{c})^2}$$

- Airfoil with sinusoidal camber line

$$y = msin(\frac{\pi x}{c})$$

- Airfoil with camber line as an arc of a circle

$$y = -\frac{c^2 - 4m^2}{8m} + \sqrt{(\frac{c^2 - 4m^2}{8m})^2 + x(c - x)}$$

Plot of Camber lines for all Airfoils

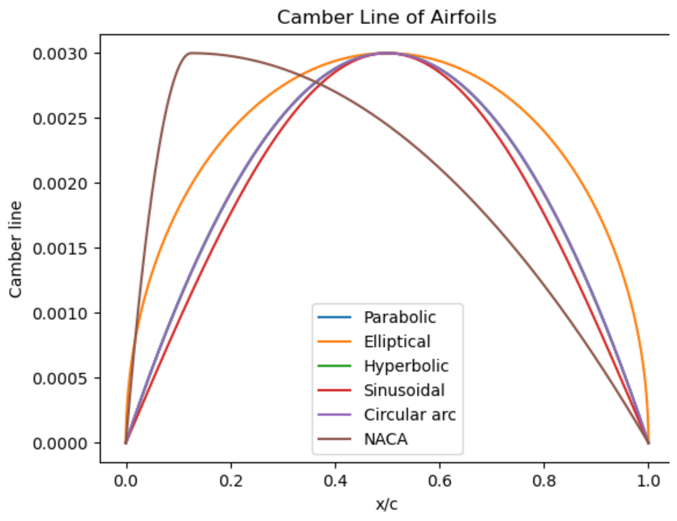

Figure 9: Unequal Axes

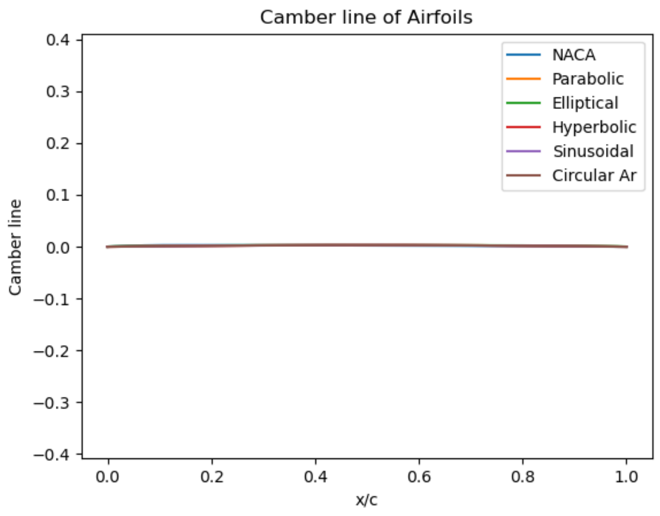

Figure 10: Equal Axes

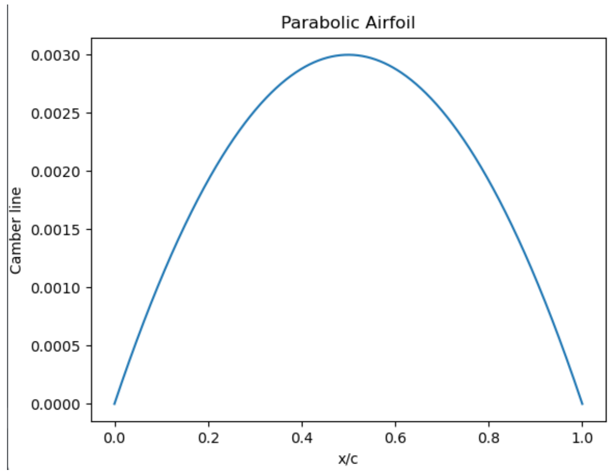

Figure 11: Parabolic Airfoil

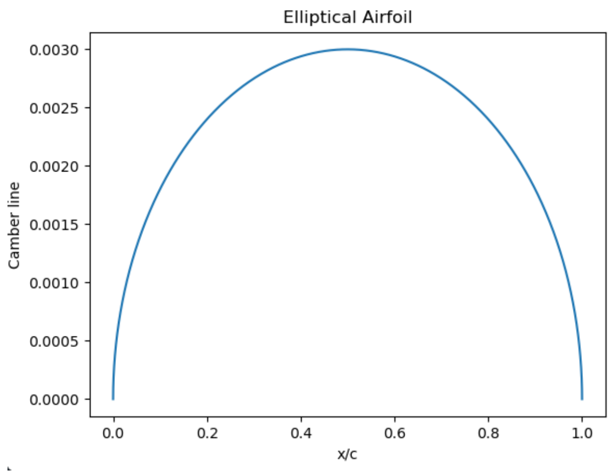

Figure 12: Elliptical Airfoil

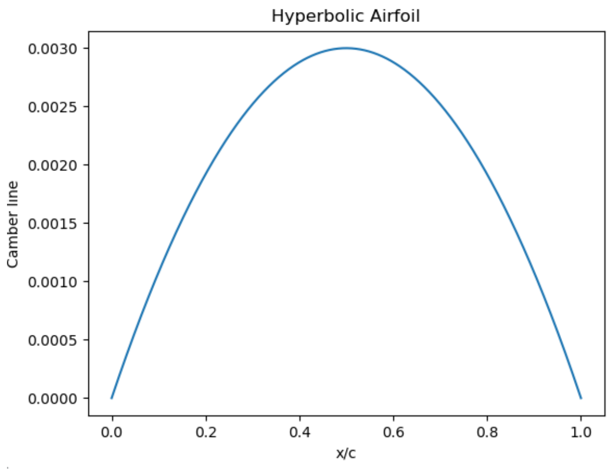

Figure 13: Hyperbolic Airfoil

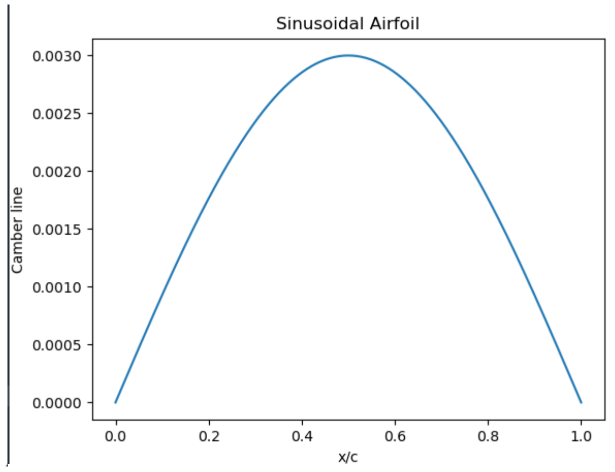

Figure 14: Sinusoidal Airfoil

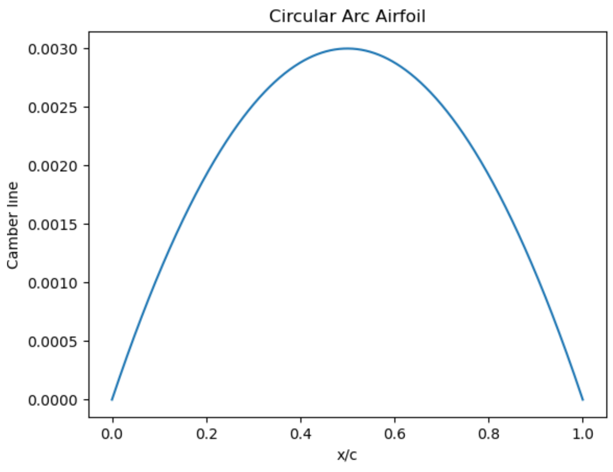

Figure 15: Circular Arc Airfoil

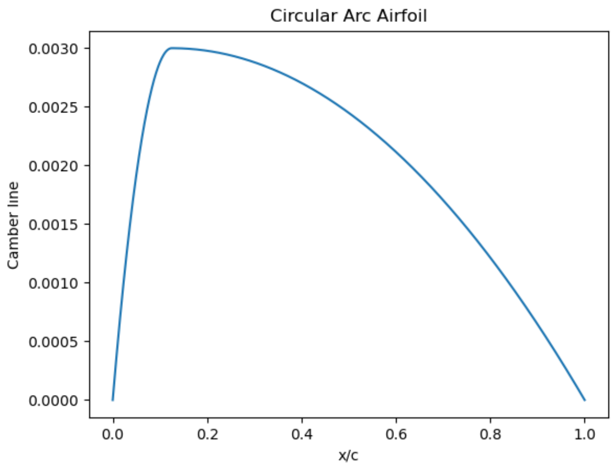

Figure 16: NACA Airfoil

Cl vs Alpha Plot

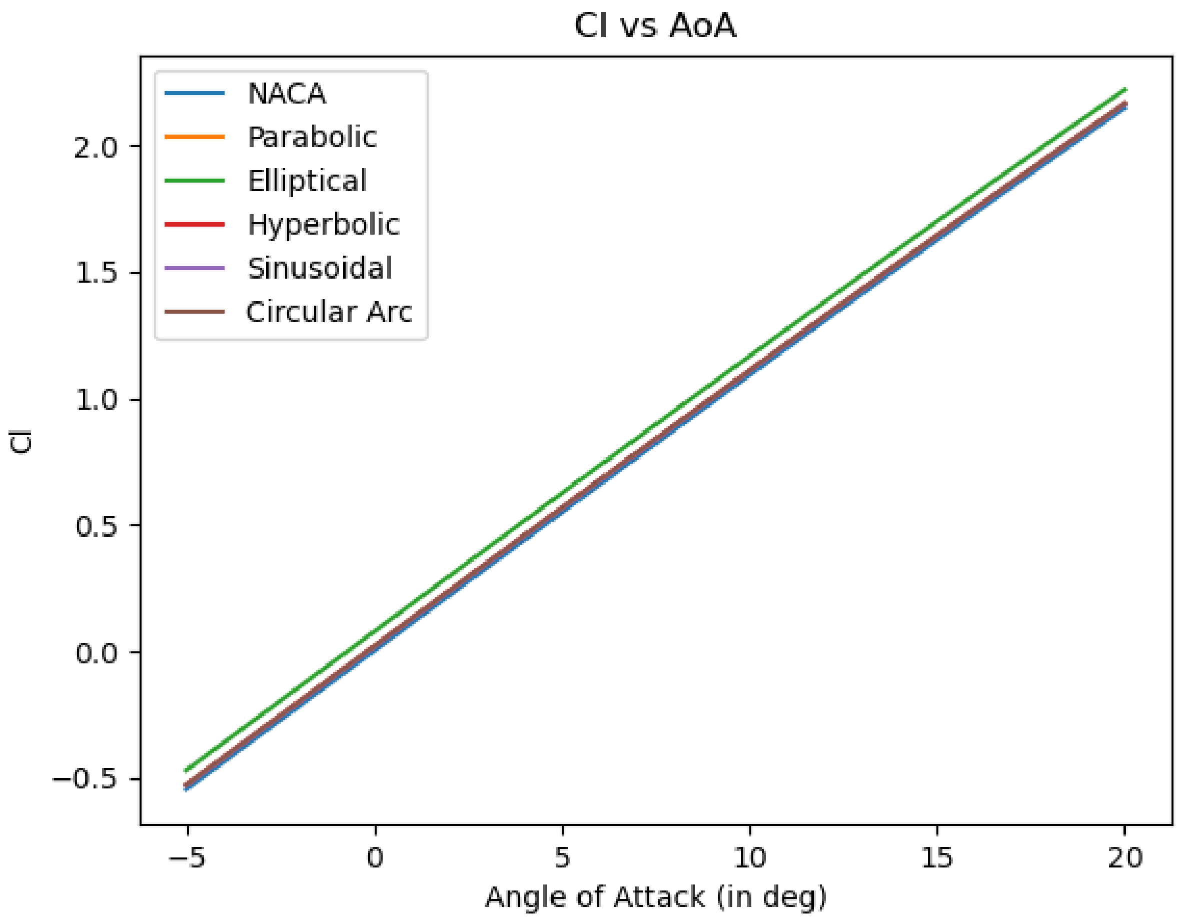

Figure 17: Cl vs Alpha plot of all airfoils

Some numerical results of Cl at $\alpha = 0^o$ & $\alpha = 10^o$ for all airfoils:

Airfoil	$\alpha = 0^o$	$\alpha = 10^o$
NACA	0.02846	1.12508
Parabolic	0.03766	1.13428
Elliptical	0.17789	1.27451
Hyperbolic	0.03763	1.13425
Sinusoidal	0.03353	1.13015
Circular Arc	0.03766	1.13428

Table 1: Cl at different α

By observing the above data, we can conclude that the elliptical camber line gives the largest Cl value at a given α while the NACA airfoil gives the least Cl value at same α.

$$Elliptical > Parabolic \approx CircularArc > Hyperbolic > Sinusoidal > NACA$$

Velocity Plot

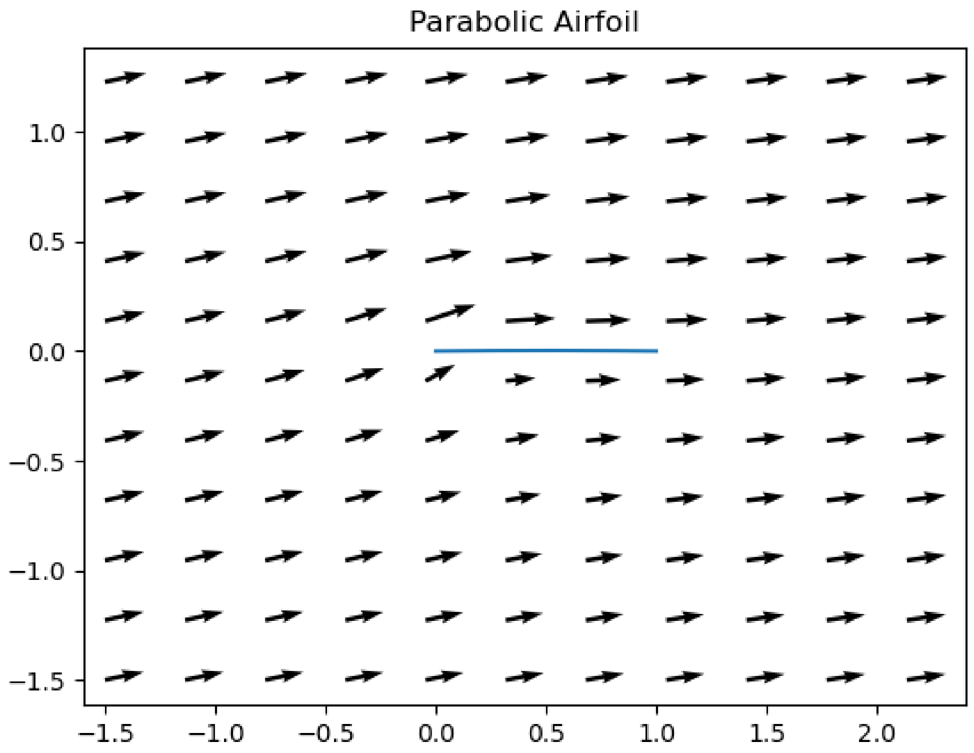

Figure 18: Parabolic Airfoil

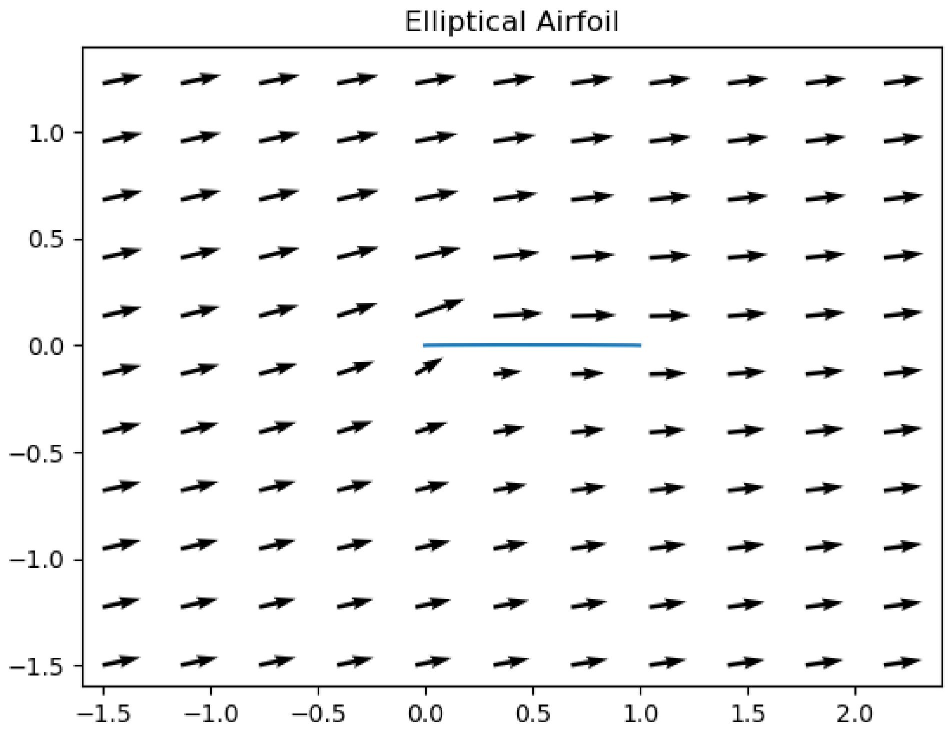

Figure 19: Elliptical Airfoil

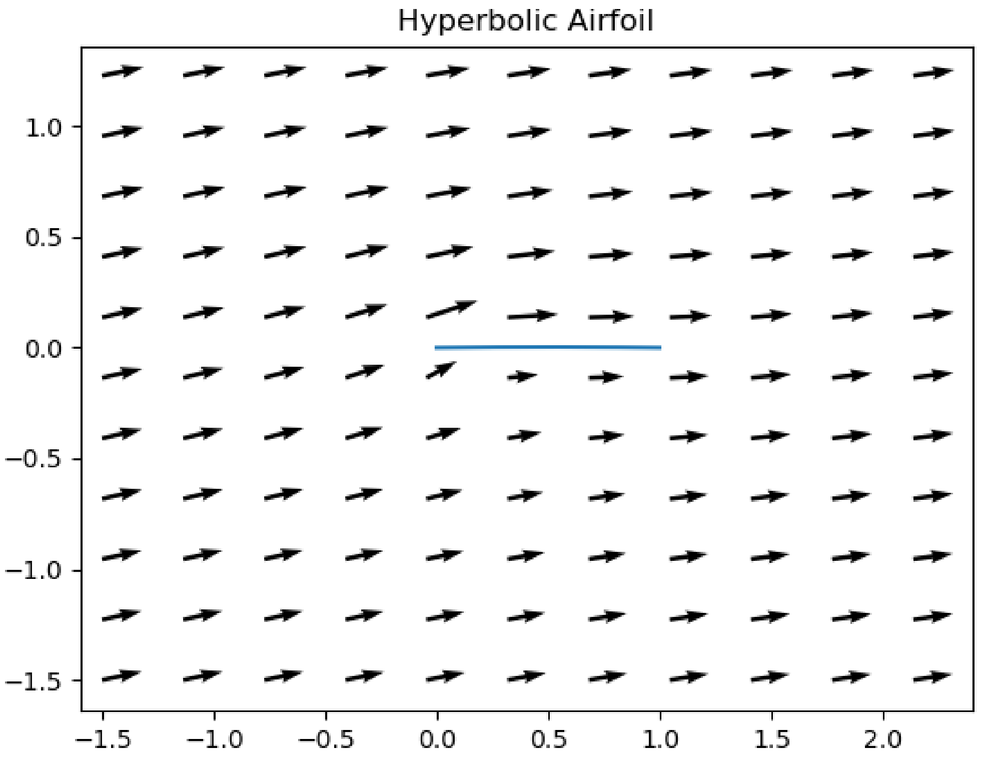

Figure 20: Hyperbolic Airfoil

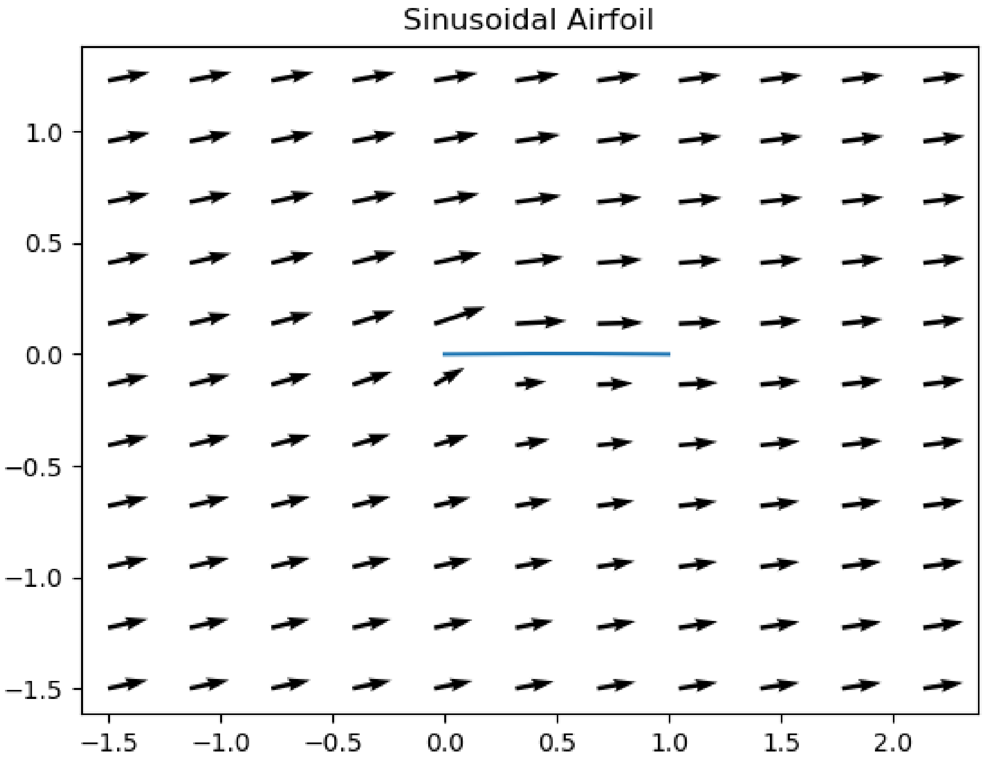

Figure 21: Sinusoidal Airfoil

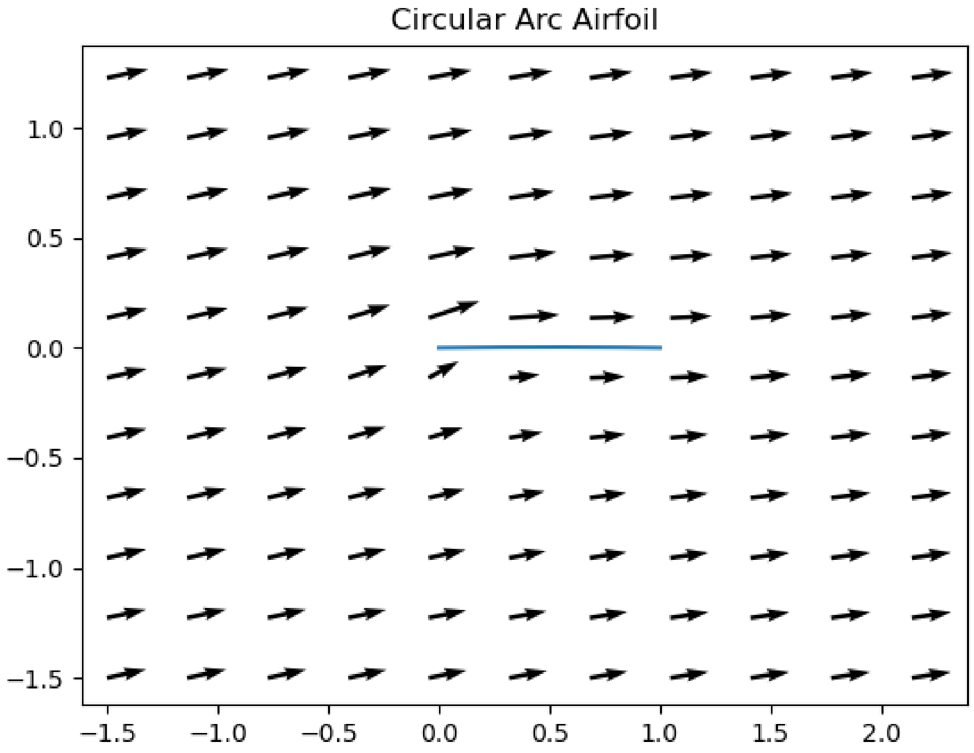

Figure 22: Circular Arc Airfoil

- The vector plots of my airfoils closely resemble that of the NACA airfoil. There exist some slight differences that can be observed if one observes keenly.
- The arrows are slightly longer right above the airfoil compared to NACA airfoil and slightly shorted below the airfoil compared to NACA airfoil. These minute differences lead to the slightly larger lift generated in the custom airfoils compared to NACA airfoil.
- We can also note that the downwash induced on the air flow by the presence of the airfoil is greater in custom designed airfoils compared to the NACA airfoils.

Conclusion

Comparation between Ansys & Python simulation

- Ansys simulation takes alot longer to complete the computations and bring the desired results compared to python simulation. Although Ansys does precise calculations considering all factors such as viscosity, pressure, temperature variations, etc. but such analysis takes correspondingly a lot more time and computational power. This is not always required. For more practical purposes, we need quicker results and approximate results will suffice. These requirements are fulfilled by the python simulation.

- As we can observe from the above Cl-α plots obtained from our python and Ansys simulation, the results are almost the same. The error is so small that is can be neglected for practical purposes. Similarly, the matching values of the bound circulation and the velocity-line integral obtained from the python simulations indicates the sufficiency and cost-effectiveness of our python code in terms of time and computational power.

- Creating an Ansys simulation requires knowledge of CAD modelling and a firm grasp on the Ansys fluent software. One needs to go through a lot of tutorials in order to efficiently utilise the software and get desired results. Plus, the vast number of options and technicalities lead to a lot of error. On the other hand, understanding and writing a python code is much faster and easier. It is also time-saving as one can quickly write the code and get the simulation results instantly. Even a beginner in Aerodynamics can firmly understand the equations and its working efficiently. Debugging the code is also much easier since the code is highly modular.

Performance analysis of custom designed vs NACA airfoils

- All of the custom designed airfoils generated higher lift coefficients compared to the NACA airfoil having same maximum camber and at same angle of attack.
- The Elliptical camber line gives the larges Cl value among all. This may be due to the fact that it disturbs the flow the largest due to its shape and hence leads to these values.
- The vector plots of my airfoils closely resemble that of the NACA airfoil. There exist some slight differences that can be observed if one observes keenly.
- The arrows are slightly longer right above the airfoil compared to NACA airfoil and slightly shorted below the airfoil compared to NACA airfoil. These minute differences lead to the slightly larger lift generated in the custom airfoils compared to NACA airfoil.
- We can also note that the downwash induced on the air flow by the presence of the airfoil is greater in custom designed airfoils compared to the NACA airfoils.

Acknowledgement

Alot of my friends helped me during my assignment. Some key mentions include Rohan, Nikhil, Shrivardhan, Binay and Devesh. They cleared my doubts throughout the assignment and made my concepts stronger. It would have been a lot more difficult to complete the assignment without their sincere help.
Our course instructor, Dhwanil sir also helped alot. His lectures made my theoretical concepts stronger.

References

- Lecture Slides
- Latex Documentation and help sites
- wikipedia page on airfoils $https://en.wikipedia.org/wiki/NACA_airfoil$
- Youtube videos on thin airfoil theory