ASSIGNMENT 2

Vishal Gautam | AE 244 | 13 March' 2024

1. Program development team

Name	Roll No.	Contribution level	Specifics of Contribution
Vishal Gautam	22B0065	5	Plotting camber line, Computing C_m & Circulation functions (2.1, 2.2, 2.5)
Devesh Mittal	22B0070	5	Computing C_l & plotting vector field (2.3, 2.4)
Shailesh Sharma	22B0021	1	Plotting camber line slope (2.2)

2. Algorithm

1. Algorithm for plotting camber line

1.1. Start

1.2. Import related libraries such as NumPy for matrix or array operations, matplotlib for plotting the graphs, and Pandas for data manipulation and analysis.

1.3. Define **m, p** | where **m** is the maximum camber in tenths of the chord and **p** is the position of the maximum camber along the chord in tenths of chord

1.4. Declare variable **n** # which is no. of points you need along the camber line

1.5. Generate n no. of random x-coordinates along the length of the airfoil, from 0 to c (which stands for chord or length) using linspace function of NumPy library and store them in array '**X**'.

1.6. Initialize an array named '**Y**' to store y-coordinates of camber line

1.7. Run a for loop over range **n** and for each **x** in array X, calculate y using the formula

$$y_c = \frac{m}{p^2}\left(2px - x^2\right) \qquad \text{from } x = 0 \text{ to } x = p$$

$$y_c = \frac{m}{(1-p)^2}\left[(1-2p) + 2px - x^2\right] \qquad \text{from } x = p \text{ to } x = c$$

1.8. Plot the camber line's coordinates (X, Y) using matplotlib library.

1.9. Return (X, Y) coordinate of the camber line as a 2-dimensional array.

1.10. STOP

---

2. Algorithm for plotting camber line slope

1. Define the function '**camber_slope**' with parameters '**x**' and '**clc**' where 'x' is x-coordinate of the point along the chord length for which you need slope and 'clc' is an array of (x, y) coordinates of the camber line.
2. Initialize variables '**left_point**' and '**right_point**' to **None**
3. Iterate through the list of coordinates '**clc**' to find the closest points to the left and right of '**x**'
    - Use a for loop to iterate through indices '**i**' from **0** to the length of 'clc' - 1
    - Check if 'x' is between **clc[i][0]** and **clc[i+1][0]**. If it is, assign **clc[i][0]** to '**left_point**' and **clc[i+1][0]** to '**right_point**', then break out of the loop.
4. Check if both '**left_point**' and '**right_point**' are not **None**. If either of them is none, raise a **ValueError**.
5. Calculate the differences in x and y coordinates between '**right_point**' and '**left_point**' ('**dx**' **and** '**dy**').
6. Calculate the slope using the formula '**dy/dx**'
7. Return the calculated slope

To plot the slope of the camber line along x (dy/dx)

8. Initialize 's' as a zeros array of size n
9. Run a for loop to iterate random x coordinates along chord length
10. find slope at each x using 'camber_slope' function and save it in s[i]
11. Now, use matplotlib to plot slope vs x that is (dy/dx) vs x graph

---

3. Algorithm for computing Cl

1. Convert angle of attack (`aoa`) from degrees to radians and store it in `alpha`.

2. Calculate the transformation angle `theta` for each point `X`:

- Initialize an array `theta` of size `n` filled with zeros.
- Loop through each point `X[i]` from 0 to `n-1`.
- Calculate `theta[i]` using the inverse cosine function `arccos` as `np.arccos(1 - 2*X[i])`.

3. Calculate the difference in theta (`d_theta`) between adjacent points:

- Initialize an array `d_theta` of size `n` filled with zeros.
- Loop through each point `i` from 0 to `n-2`.
- Calculate `d_theta[i]` as the difference between `theta[i+1]` and `theta[i]`.
- Set `d_theta[n-1]` to `d_theta[n-2] + 0.15` for normalizing the end boundary points.

4. Calculate the spacing `d_s` between points along the camber line:

- Initialize an array `d_s` of size `n` filled with zeros.
- Loop through each point `X[i]` from 0 to `n-1`.
- Calculate `d_s[i]` as the square root of `1 + (camber_slope(X[i], clc))**2` divided by `n`.

5. Calculate the coefficients `A`:

- Initialize an array `A` of size `n` filled with zeros.
- Loop through each index `i` from 0 to `n-1`.
- Initialize a variable `s` to 0.
- Loop through each index `j` from 0 to `n-1`.
- Calculate `s` as the sum of `camber_slope(X[j], clc) * cos(i * theta[j]) * d_theta[j]`.
- Calculate `A[i]` as `s * 2 / pi`.

6. Normalize `A[0]`:

- Update `A[0]` as `alpha - 0.5 * A[0]`.

7. Calculate the coefficient of lift (`cl`):

- Calculate `cl` as `pi * (2 * A[0] + A[1])`.
- Print the coefficient of lift.

8. Calculate the coefficient of moment about the leading edge (`Cm`):

- Calculate `Cm` as `pi * (A[0] + A[1] - 0.5 * A[2]) * 2`.
- Print the coefficient of moment about the leading edge.

4. Algorithm for plotting vector field

4.1. Initialize an array `gamma` of size `n` filled with zeros to store vorticity at each point.

4.2. Loop through each point `i` from 0 to `n-1`:

- Initialize `s` to 0.
- Loop through each coefficient `A[j]` from 0 to `n-1`:
- Calculate `s` as the sum of `A[j] * sin (j * theta[i]) `.
- Calculate `gamma[i]` using the formula:

```

gamma[i] = 2 * u_infi * (A [0] * ((1 + cos(theta[i])) / (sin(theta[i]) + 0.000001), which is derived from
```

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

```

- Increment `i`.

4.3. Set `gamma[0]` and `gamma[n-1]` to 0.

4.4. Define a function `v_ind(Xx, Yy)` to compute the net induced velocity at each point `(Xx[i,j], Yy[i,j])`:

- Initialize arrays `uu` and `vv` of size `nn x nn` filled with zeros to store velocity components.
- Loop through each point `(i,j)` in the meshgrid:
- Retrieve `x` and `y` coordinates from `Xx[i,j]` and `Yy[i,j]`.
- Initialize `v_x` and `v_y` to 0.
- Loop through each vortex point `t` from 0 to `n-1`:
- Calculate distance `s` between point `(x, y)` and vortex point `(X[t], Y[t]) `.
- Calculate the induced velocity components `dv_x` and `dv_y` using Biot-Savart's law.
- Update `v_x` and `v_y` by adding `dv_x` and `dv_y` respectively.
- Store `v_x` and `v_y` in `uu[i,j]` and `vv[i,j]` respectively.
- Return `uu` and `vv`.

4.5. Generate meshgrid `Xx` and `Yy` using `linspace` for `Aa` and `Bb`.

4.6. Compute induced velocities `U` and `V` using function `v_ind(Xx, Yy)`, then add freestream component `u_infi` to `U`.

4.7. Iterate over each point `(i, j)` in the meshgrid:

- Check the distance between the point ` (Xx[i, j], Yy[i, j]) ` and each vortex point `(X[t], Y[t]) `.
- If the distance is less than a threshold value (0.005), set induced velocities `U[i, j] ` and `V [i, j] ` to 0.

4.8. Plot the vector field:

- Set the figure size to (12, 10).
- Set the x-limits and y-limits of the plot.
- Plot the quiver plot using `quiver` function to represent velocity vectors.
```

- Plot the camber line using `plot` function.
- Display the plot.

---

5. Algorithm for calculating circulation through circulation distribution

Using formula $\Gamma = \int_0^c \gamma(s)ds$ where $\gamma(s)$ is circulation distribution at 's' and ds is an infinitely small element over a mean camber line

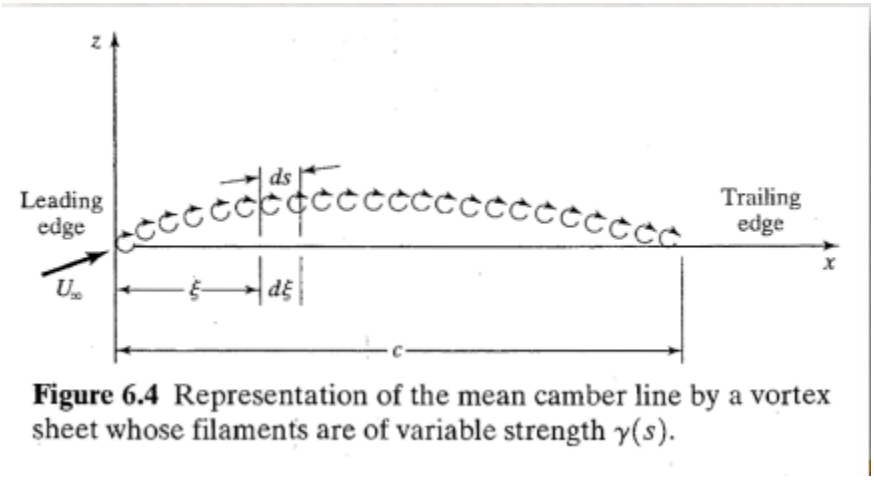


Figure 6.4 Representation of the mean camber line by a vortex sheet whose filaments are of variable strength $\gamma(s)$.

5.1. Find $\gamma(s)$ at every point on a mean camber line using gamma function defined in algorithm 4

5.2. Calculate ds using formula $\cos(\varphi) = \frac{dx}{ds}$ where φ is a slope angle of a camber line at that infinitely small element.

- 5.2.1. Initialize d_X as a zeros array of size n
- 5.2.2. Run a for loop to calculate d_X using d_X[i] = X[i+1]-X[i]
- 5.2.3. Now initialize s as a zeros array of size n
- 5.2.4. Run for loop to store slopes of all points on a camber line in s using s[i] = camber_slope(x,clc)
- 5.2.5. Now calculate values using arctan(s) because $\tan(\varphi) = \frac{dy}{dx}$
- 5.2.6. Calculate using cos () function NumPy library
- 5.2.7. Now finally calculate ds by using $\cos(\varphi) = \frac{dx}{ds}$

5.3. Since we have $\gamma(s)$ and ds at all over the camber line so we will find now Circulation (Γ) using formula $\Gamma = \int_0^c \gamma(s)ds$

5.4. To implement it declare circ = 0 use a for loop iterate it over range n and calculate circulation using `circ += gamma[i]*ds[i]`
5.5. And finally, we got Total circulation = circ.

---

6. Algorithm for calculating circulation through line integral

1. Divide the ellipse into 360 parts by generating an array of angles ranging from 0 to 2π.

2. Compute the coordinates of points on the ellipse using parametric equations.

3. Calculate the velocity components at each point on the ellipse using a separate function (`vel`).

4. Compute the differentials of the x and y coordinates to prepare for line integral calculation.

5. Initialize a variable to store the total circulation and set it to 0.

6. Loop through each element on the ellipse (except the last one) and perform line integral calculation:

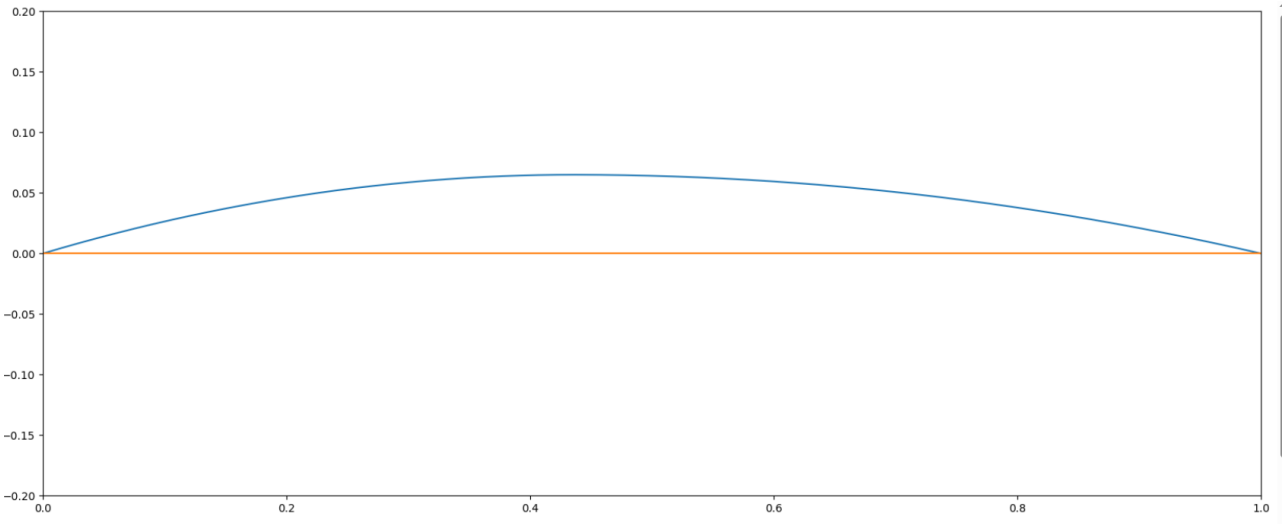
- Calculate the circulation contribution of each small element by multiplying the velocity components with the differentials of x and y, respectively, and summing them up.
- The negative sign is used because circulation is being computed in the clockwise direction.

7. Print the total circulation obtained using the velocity approach.

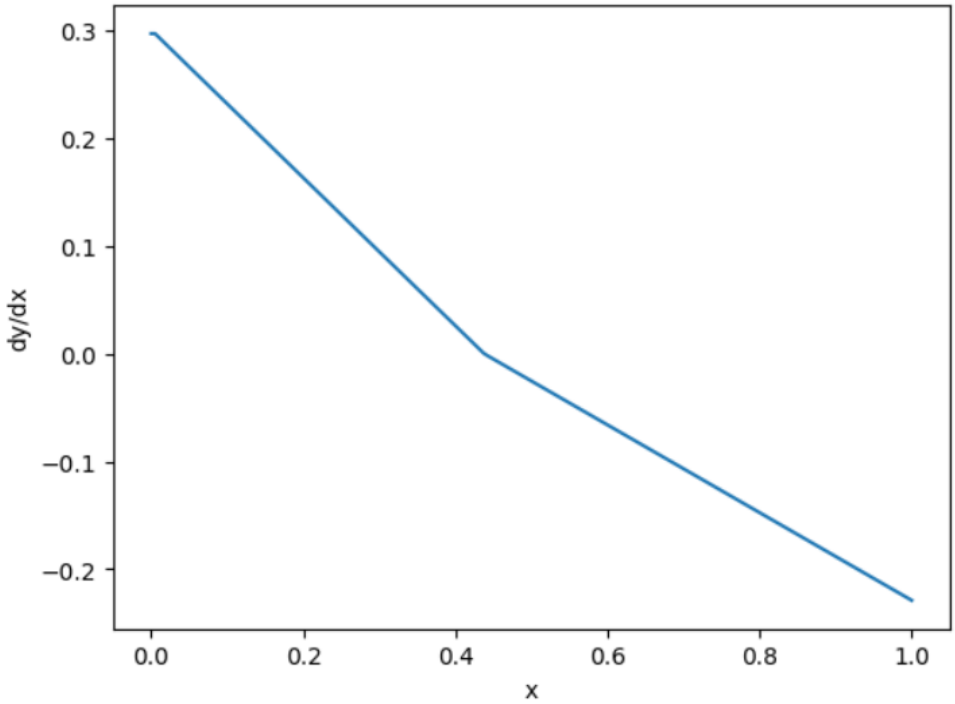
---

3. Airfoil Simulation and Results

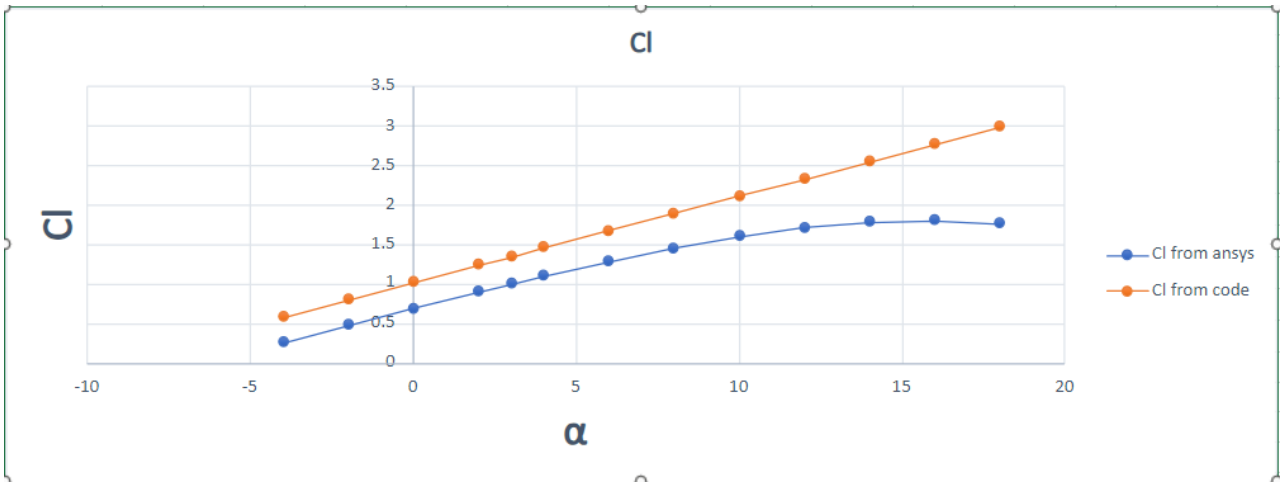
1. Camber line ('y' vs 'x') for the NACA airfoil from assignment 1



2. Slope of the camber line along x (dy/dx vs x)



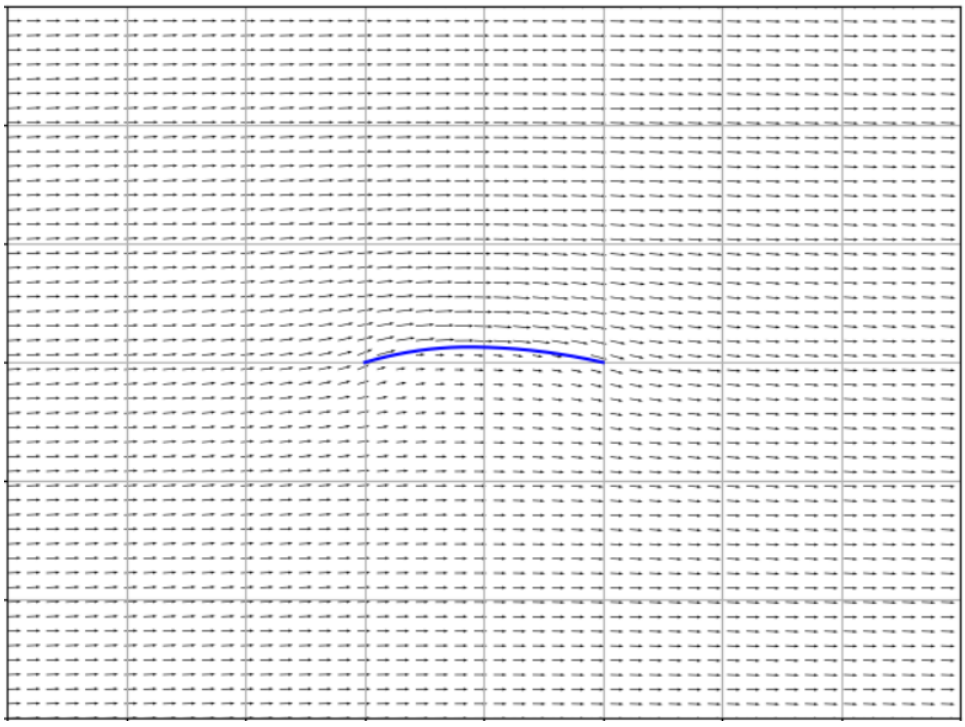
3. C_l vs α for the NACA airfoil from the program and CFD simulations on same chart



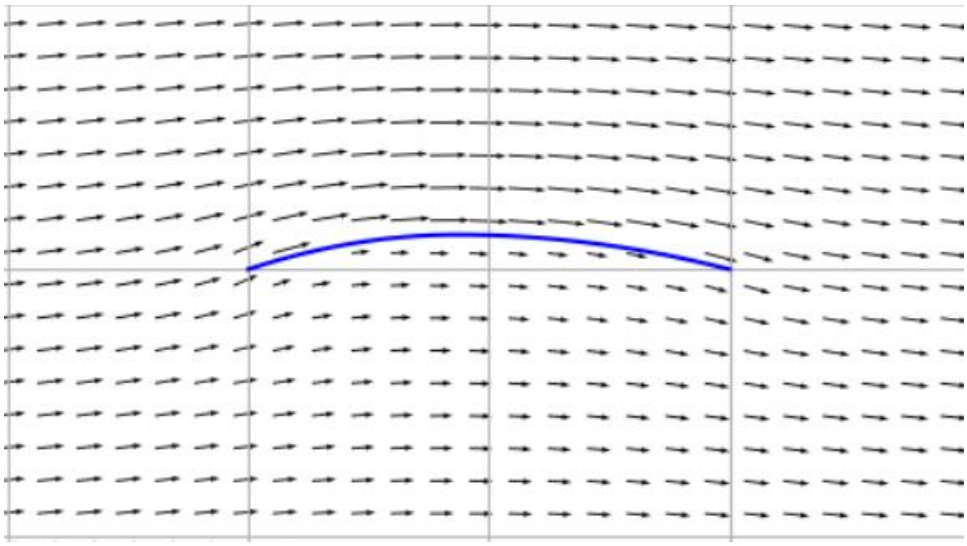
4. Comment on/discuss the results obtained in 3.3.

   Since we coded Coefficient of Lift function considering thin airfoil assumptions. And we know for thin airfoil we will never get stall condition and same you can see in plot, the C_l curve from code has never attained stalled condition whereas C_l curve from Ansys have stall angle 14^{o}. And as the coded C_l value is an approximate one so it is somewhat different from the value we got from simulation at same angle of attack.

5. .Vector field plot around the airfoil (domain size: 4c times 3c) at $\alpha = 3^{o}$



Zoomed View



6. Comment on/discuss the plot obtained in 3.5

   The plot reveals the formation of distinct flow regions characterized by varying velocities and directions. Near the leading edge, the airflow appears accelerated, as indicated by the denser concentration of vectors and their orientation. This acceleration corresponds to the generation of lift.

   The inclusion of the airfoil's camber line enhances the understanding of the airfoil's shape and curvature, further emphasizing the lift generation and airflow behavior. By adhering to the assumptions of the thin airfoil theory, the plot provides a simplified yet insightful representation of the aerodynamic phenomena.

7. Circulation around the entire airfoil using velocity line-integral approach at $\alpha = 3^{o}$
   Total Circulation = 19.7774787

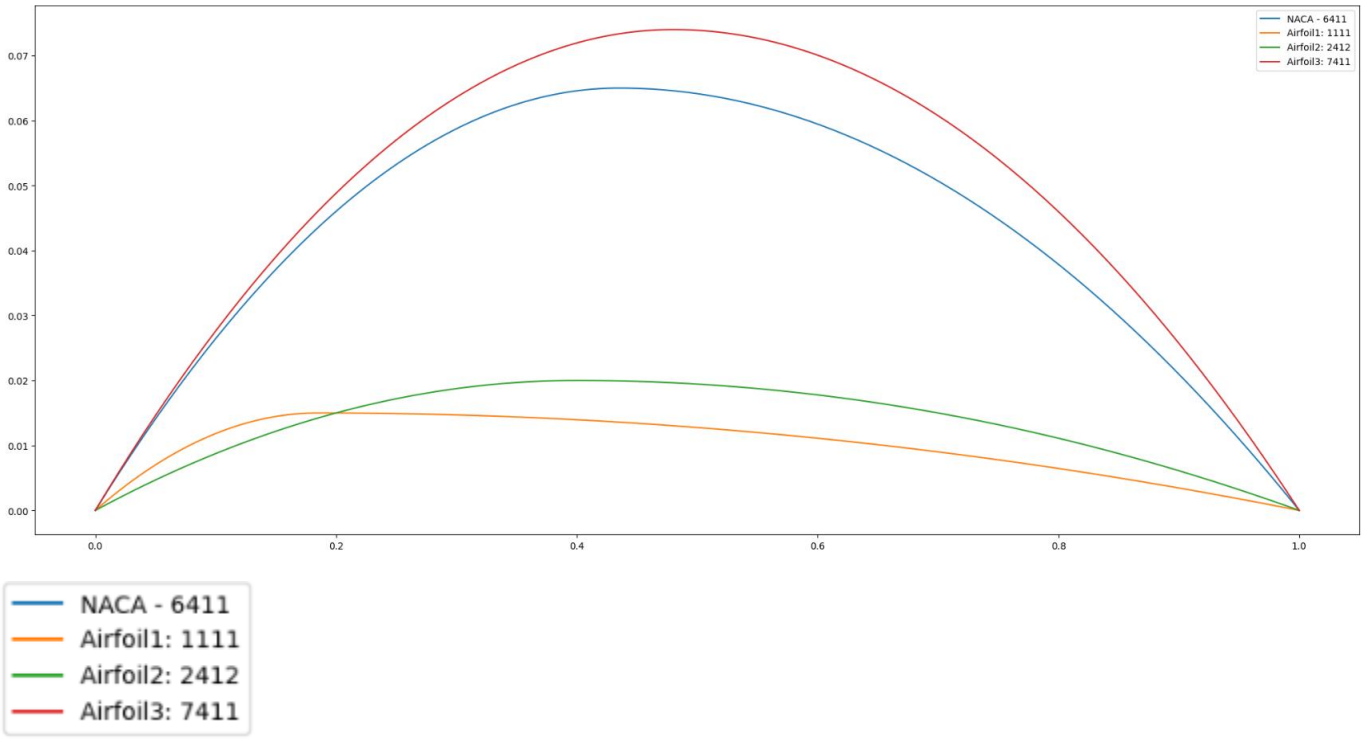
8. Bound circulation by integrating circulation distribution along camber line. $\alpha = 3^{o}$
   Total bound circulation = 19.778488

9. Compare and comment on the values obtained in 3.7 and 3.8.

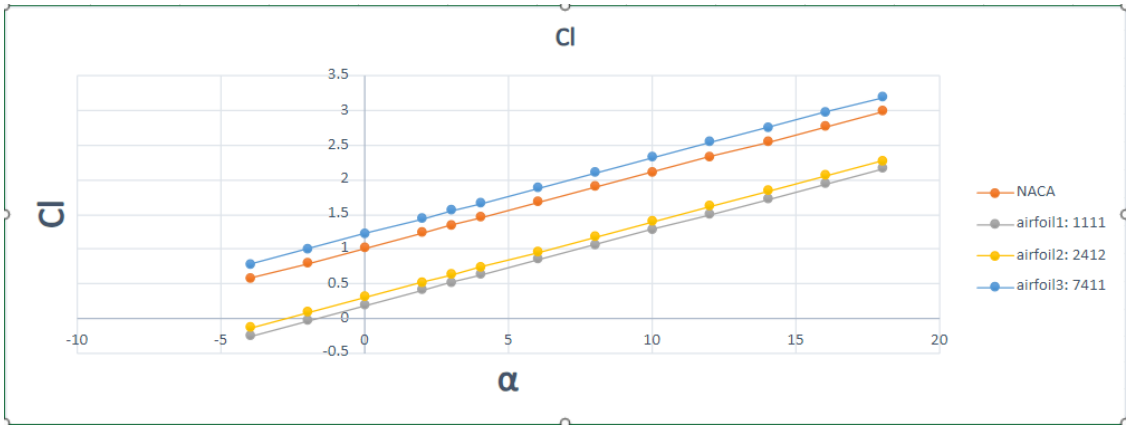
   Both values are almost the same so both codes work fine and give the accurate result.

4. Novel Airfoil Properties

4.1.Camber line ('y' vs 'x') for your 3 airfoils and the NACA airfoil on same chart



4.2.Cl vs α of the three airfoils along with that of the NACA airfoil on same chart

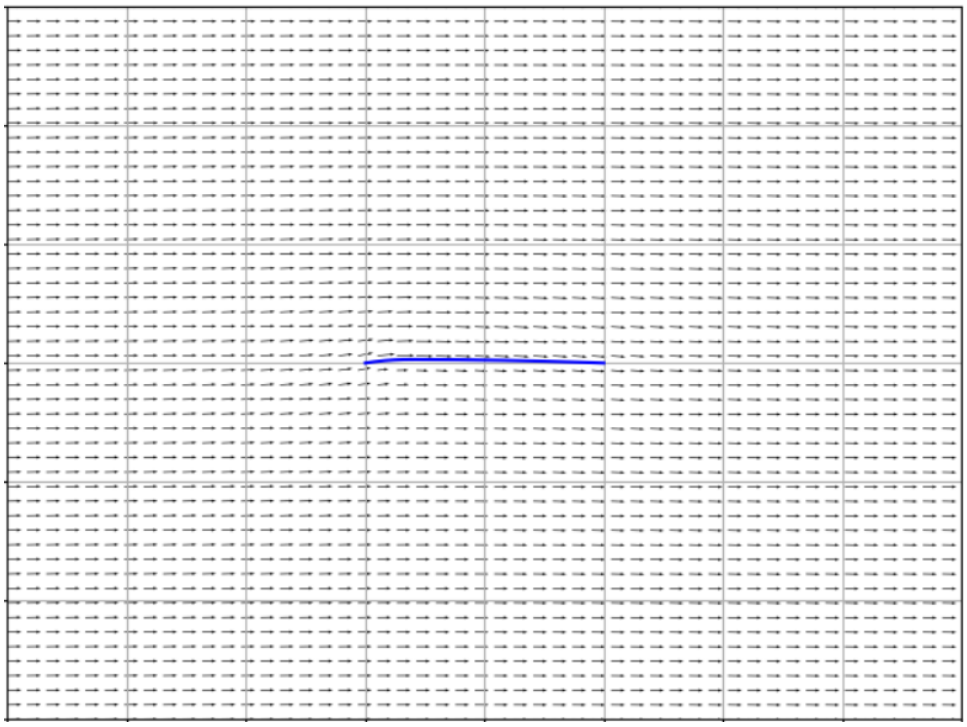


4.3.Comment on/discuss the results obtained in 4.2

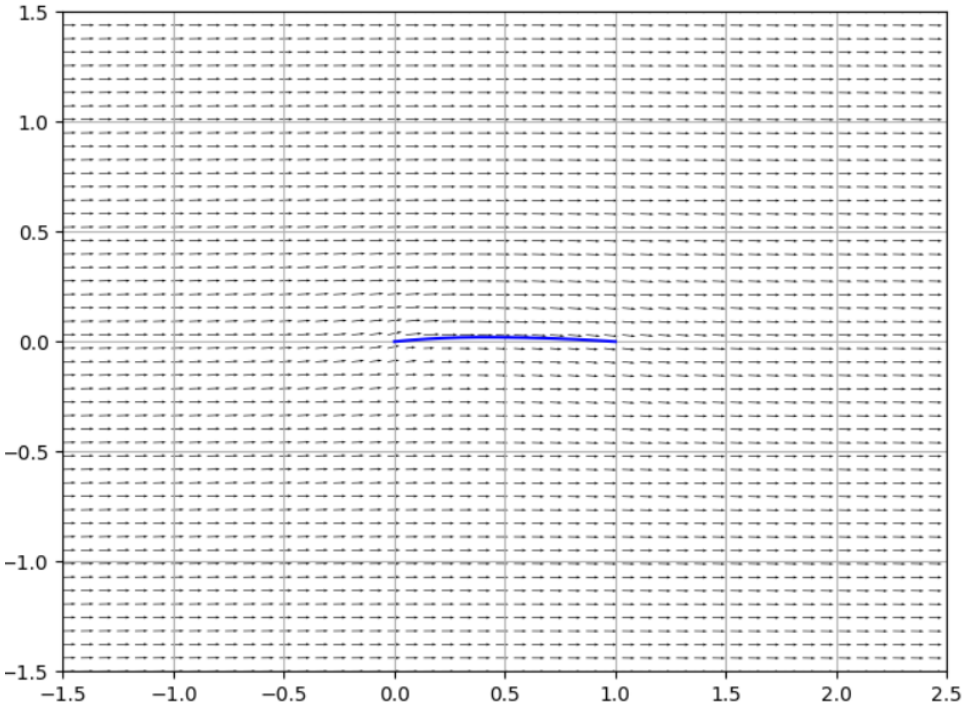
Code seems to be working fine as for Airfoil3: 7411 it is generating more lift than our NACA one which is NACA 6411, and it should be as Airfoil3 has a maximum camber of 7.4% whereas NACA has a maximum camber of 6.5%. There is a similar interpretation you can make for airfoil1 which has maximum camber of 1.5% and airfoil 2 which has maximum camber of 2%.

4.4.Vector field plot around the three airfoils (domain size: 4c times 3c) at $\alpha = 3^{o}$

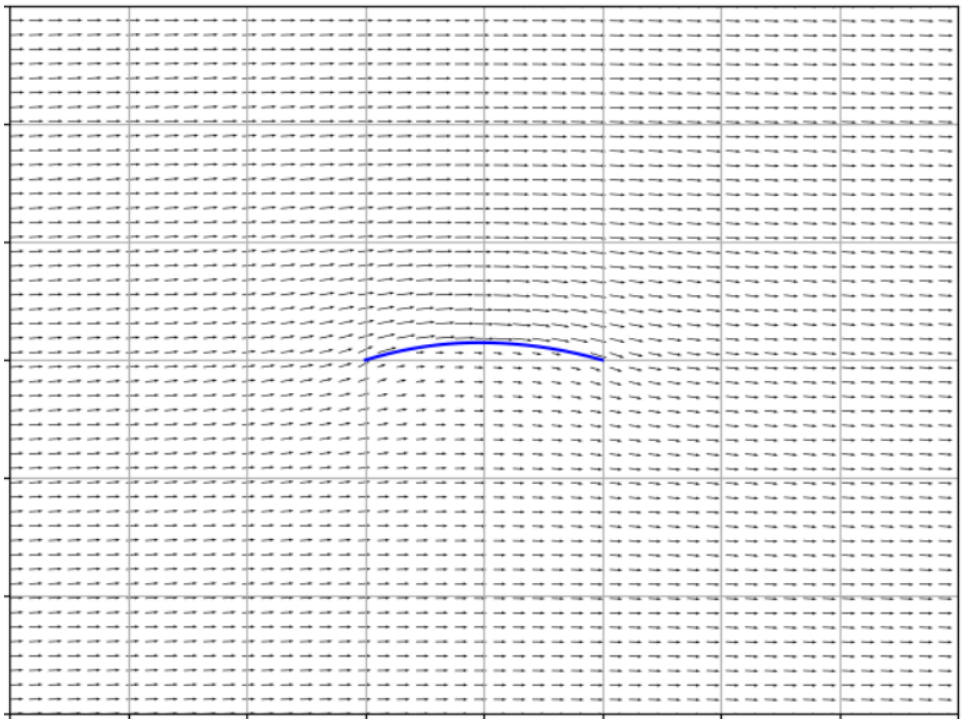
- Airfoil1 – 1111



- Airfoil2 – 2412



- Airfoil3 - 7411



4.5.Comment on/discuss the results obtained in 4.4

The plots reveal the formation of distinct flow regions characterized by varying velocities and directions. As you can see near the leading edge, the airflow appears more accelerated, as indicated by the denser concentration of vectors and their orientation. This acceleration corresponds to the generation of lift.

See for airfoil3 that's our high cambered airfoil, near the leading edge, you can see denser concentration of vectors and steeper orientations w.r.t airfoil1 and airfoil2 which depicts airflow accelerated and that corresponds to high lift generation.

And it should be as cambered airfoil generates more lift as compared to symmetrical or low cambered airfoil.

5. Novel Airfoil Properties

5.1. Overall take on your code's performance as compared to Ansys simulation, and possible reasons for deviations, if any

Possible reason for the deviation which is coming in C_l values from our code and the simulation values at same angle of attack could be this approximation (Consider normalized airfoil with leading edge at x = 0 and trailing edge at x = 1.)

because the camber line coordinates coming from the function or even the NACA airfoil's leading edge doesn't needs to have leading edge always at x = 0 it could be somewhere at x = 0.001 or x = -0.001 just a guess.

Other functions give accurate results as we have seen for Circulation, both circulation functions give the same accurate values.

5.2. Overall take on the performance of your airfoil as compared to the NACA airfoil

The airfoil we got after considering thin airfoil theory assumptions performed almost same as the NACA airfoil. Since we got precise results for vector field plot around the airfoil based on circulation distribution, also for circulation we got approx accurate result from both methods circulation around the entire airfoil using velocity line-integral approach one and the bound circulation by integrating circulation distribution along camber line at alpha = 3 degree.

And as we know the velocity line –integral approached circulation around the airfoil used to be same as of NACA airfoil since in that we did not used any thin airfoil theory assumption. And in bound circulation by integrating circulation distribution along camber line used to work on thin airfoil theory assumptions only and we are getting the same results from both, so our airfoil is performing same as NACA, so in conclusion thin airfoil assumption's really works and make the computation easy.

---

6. Code

6.1. Proper working of the camber line plotting code

```python
def airfoil_plot(m, p, n):

    # dividing X into n points
    X = np.linspace(0,1,n, dtype = float).round(4)
    # making n Y points
    Y = np.zeros_like(X)

    # plotting X, Y coordinates of desired airfoil's camber line using NACA formula
    for i, x in enumerate(X):
        if x<p:
            Y[i] = (m/(p**2)) * (2*p*x - x**2)
        if x>=p:
            Y[i] = (m/((1-p)**2)) * ((1-2*p) + 2*p*x - x**2)

    Xaxis = [0]*len(X)

    # Plotting the airfoil camber line
    plt.figure(figsize=(20,8))
    plt.xlim(-0,1)
    plt.ylim(-0.2,0.2)
    plt.plot(X,Y, )
    plt.plot(X,Xaxis)
    plt.show()

    # creating list of camber coordinates
    clc = []
    for i in range(len(X)):
        clc.append([X[i], Y[i]])

    return X, Y, clc
```

6.2. Proper working of the camber line slope plotting code

```python
def camber_slope(x, clc):

    left_point = None
    right_point = None

    for i in range(len(clc) - 1):
        if clc[i][0] <= x <= clc[i + 1][0]:
            left_point = clc[i]
            right_point = clc[i + 1]
            break

    if left_point is None or right_point is None:
        raise ValueError("Unable to find points on the camber line for the given x-coordinate.")

    # Calculating slope using the two closest points
    dx = right_point[0] - left_point[0]
    dy = right_point[1] - left_point[1]

    slope = dy / dx


    return slope
# This function gives the slope of the camber line at a point
# To plot the slope of the camber line along x (dy/dx vs x) run the slope plotting code
```

```python
# Now plotting the slope of the camber line along x (dy/dx vs x)
# clc is the list of camber coordinates calculated using Function aifoil_plot
def slope_plot (clc):
    n = len(clc)
    s = np.zeros(n)
    X = np.linspace(0,1,n, dtype = float).round(4)
    for i,x in enumerate(X):
        s[i] = camber_slope(x,clc)

    plt.plot(X,s)
    plt.xlabel('x')
    plt.ylabel('dy/dx')
    # plt.plot(X,Xaxis)
    plt.show()
```

6.3. Proper working of Cl and Cm code

```
def coefficient_of_lift(n, aoa, X, clc):

    ## coverting deg to radians
    alpha = aoa*np.pi/180

    ## Finding theta transformation of X coordi
    theta = np.zeros(n)
    i = 0
    for i in range (n):
        theta[i] = np.arccos(1-(2*X[i]))

    ## Finding d_theta
    d_theta = np.zeros(n)
    i = 0
    for i in range (n-1):
        d_theta[i] = theta[i+1]-theta[i]

    d_theta[n-1] = d_theta[n-2]+0.15 # normalizing end boundary points

    ## finding d_s
    d_s = np.zeros(n)
    i = 0
    for i in range (n):
        d_s[i] = np.sqrt(1+(camber_slope(X[i],clc))*(camber_slope(X[i],clc)))/n

    ## Finding A(n)'s
    A = np.zeros(n)
    i = 0
    while i<n:
        s = 0
        j = 0
        for j in range(n):
            s = s + camber_slope(X[j],clc)*np.cos((i)*theta[j])*d_theta[j]
        A[i] = s*2/np.pi
        i = i+1
    A[0] = alpha - 0.5*A[0]  # normalizing A[0]

    ## coefficient of lift
    cl = np.pi*(2*A[0]+A[1])
    print('Coefficient of lift:', cl)

    ## coefficient of moment about leading edge
    Cm = np.pi*(A[0]+A[1]-A[2]*0.5)*2
    print("Coefficient of Moment about leading edge:", Cm)
```

6.4. Proper working of vector field plotting code

```
def vector_field(n, nn, u_infi, A, theta, X, Y, d_s ):

    ## vorticity at each nth point
    gamma = np.zeros(n)
    i = 0
    while i<n:
        s = 0
        j = 0
        for j in range(n):
            s = s + A[j]*np.sin(j*theta[i])

        gamma[i] = 2*u_infi*( A[0]*((1+np.cos(theta[i]))/(np.sin(theta[i])+0.000001)) + s)
        i = i+1

    gamma[0]=0  # going for 0, instead of infi
    gamma[n-1]=0 # value was anways close to 0, so made it o for kutta condition

    ## defining a funtion to get the net induced velocity on a point (x,y)
    def v_ind(Xx, Yy):
        # in biot savarts law, assuming abs(dl)=1
        # contains the net velocity_x induced on point (x,y)
        uu = np.zeros((nn, nn))
        vv = np.zeros((nn, nn))

        for j in range(nn):
            for i in range(nn):

                x = Xx[i,j]
                y = Yy[i,j]

                # x component velocity
                v_x = 0
                v_y = 0
                for t in range(n):
                    s = np.sqrt((y-Y[t])*(y-Y[t]) + (x-X[t])*(x-X[t]))
                    dv_y = -gamma[t]*(x-X[t])/(2*np.pi*s*s)*d_s[t]      #infi line of vortex
                    v_y = v_y + dv_y                                        #assuming this magnitude of
                                                                            # stays constant through length ds
                    dv_x = gamma[t]*(y-Y[t])/(2*np.pi*s*s)*d_s[t]       #infi line of vortex
                    v_x = v_x + dv_x
                vv[i, j] = v_y
                uu[i, j] = v_x

        return uu, vv
```

```
## data collection for plotting
# Creating data set

Aa = np.linspace(-1.5, 2.5, nn)
Bb = np.linspace(-1.5, 1.5, nn)

Yy, Xx = np.meshgrid(Bb, Aa)

U, V= v_ind(Xx, Yy)
U = U+u_infi # freestream component


# removing points close to the airfoil
for j in range(nn):
    for i in range(nn):
        for t in range(n):
            if np.sqrt((Yy[i, j]-Y[t])*(Yy[i, j]-Y[t]) + (Xx[i, j]-X[t])*(Xx[i, j]-X[t]))<0.005:
                U[i, j] = 0
                V[i, j] = 0

# Create the plot

plt.figure(figsize=(12,10))

# Set the x-limits and y-limits of the plot
plt.xlim([-1.5, 2.5])
plt.ylim([-1.5, 1.5])

plt.quiver(Xx, Yy, U, V, scale=2000, width=0.001) #, angles='xy', scale_units='xy', scale=1, color='r')

# camber line
plt.plot(X,Y, color ='blue')

# Show the plot along with the grid
plt.grid()
plt.show()
```

6.5. Proper working of circulation calculation

Bound circulation by integrating circulation distribution along camber line

```
[65]
def net_bound_circulation(n, d_s, gamma):

    # calculating circulation due to airfoil through line integral
    circ = 0
    for i in range (n):
        circ += gamma[i]*d_s[i]

    print ("Total bound circulation is:", circ)
```

CIRCULATION USING VELOCITY LINE APPROACH

```
# Line-integral approch by integrating along an ellipse a = 3, b = 2 centre at (0.5,0)

def net_circulation_vel_approach(u_infi, clc, aoa):

  # dividing ellipse in 360 part to create a infinitely small element over it
  d_t = np.linspace(0, 2*np.pi, 360)

  X_ = 3*np.cos(d_t)
  Y_ = 2*np.sin(d_t)
  U, V = vel(X_, Y_,u_infi, clc,aoa)
  dx = np.diff(X_)
  dy = np.diff(Y_)
  circ = 0
  for i in range (359):

    circ  -= dx[i]*U[i]+ dy[i]*V[i]

  print ("Total circulation using velocity approach is:", circ)
```

You can check the working of the code using the .ipynb notebook I have uploaded with this report or using this link: https://colab.research.google.com/drive/1GzhR1SksEELaJRjG6ONsVTsGtT-uoDom?usp=sharing

7. Acknowledgement

People with whom I have discussed

- Alvin Bunny Simphson (22B0015)
- Pranav Yadav (22B0074)
- Prem Sagar (22B0052)
- Anuj (22B0064)
- Balakirshna (22B0028)

8. References

- Microsoft Word - The NACA airfoil series.doc (stanford.edu) for plotting the camber line coordinates
- Gas Dynamics (iitb.ac.in)