STOL - Assignment 2

NAME: SAMANTH MARTIS

ROLL NUMBER: 23B0046

Section 1:
Team Workshare

1.1

[Note: Contribution level marked against each team member will be used as a scaling factor while assigning marks for the team tasks]

Sr. No	Roll Number	Name	Contribution Level (0 to 5)	Specifics of Contribution
1	23B0046	Samanth Martis	5	Complete project in Javascript
2	23B0047	Ritthika Gupta	5	Complete project in Python
3	23B0016	Ishtiyaq Ahmad Khatana	1	

Contribution Level Rubrics:
0: Was completely unresponsive and did not put any effort.
1: Responded, but didn't do the promised tasks, and didn't try to learn to do it either.
2: Did the promised/assigned tasks only partially/incorrectly and didn't try to learn to do it completely/correctly.
3: Did the promised/assigned tasks only partially/incorrectly but put some effort to learn to do it right.
4: Did the promised/assigned tasks to just acceptable quality with or without guidance from the other team members.
5: Did the promised/assigned tasks completely with or without guidance from other team members.

Section 2:
Algorithm

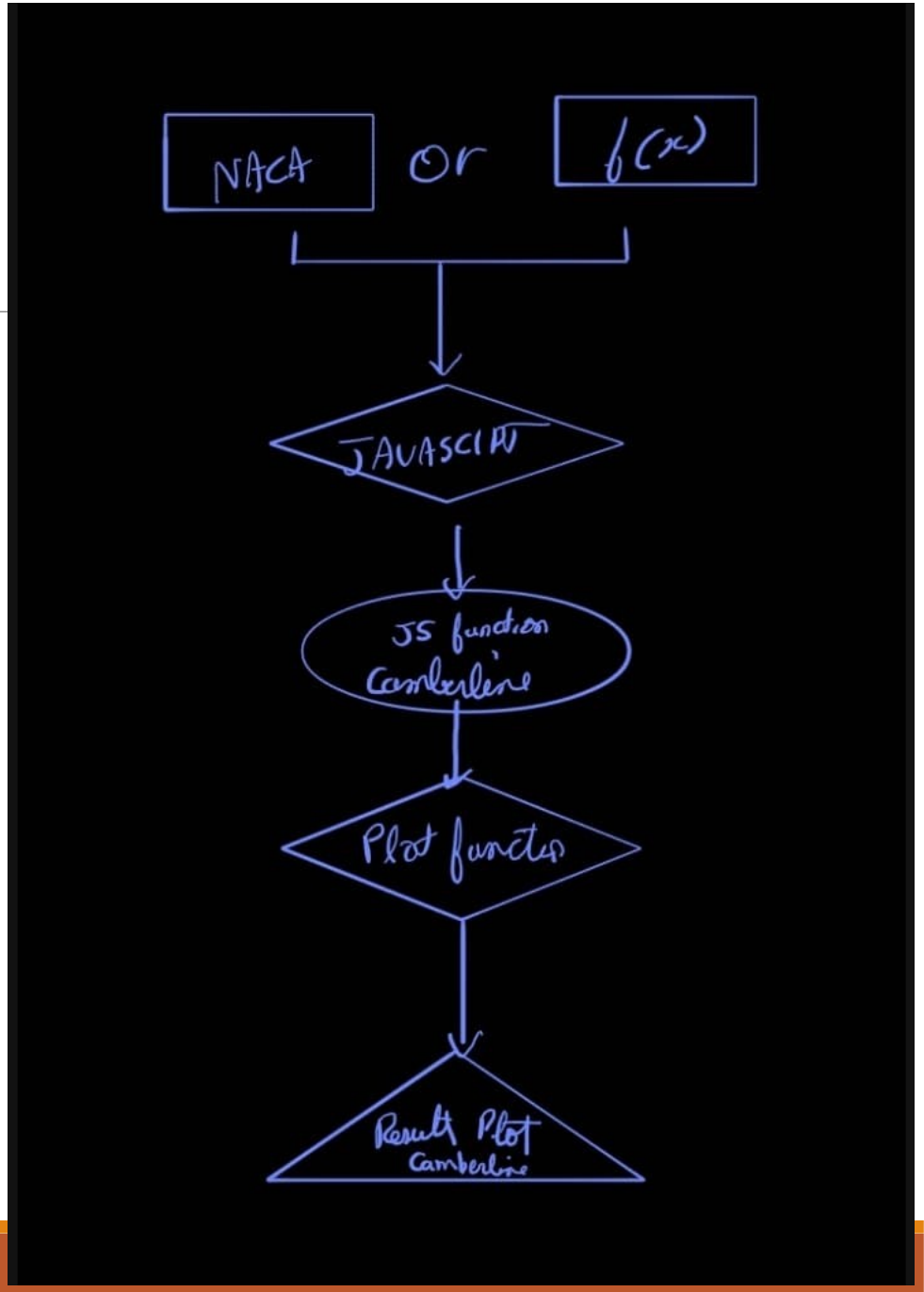
2.1:
Algorithm for plotting camber line

1. Create a general function for plotting an arbitrary function across a fixed interval via rectangle rule. Pass the function to be plotted as an argument to the plot function

2. Create a camber function based on NACA parameters or by using arbitrary user input. Former case involves using established NACA equations. In the latter case, we use JavaScript's builtin parser to parse the user into a function that we pass into the Plot function.

```
try {
    camberFunction = new Function("x", `return -x*(x-1)*${CustomFunctionInputArea.value};`) as (x: number) => number;
    if(camberFunction(0)!= 0 || camberFunction(1)!=0){
      throw Error("Invalid custom function. Must be 0 at xc = 0 and xc=1")
    }
} catch (e) {
    alert("Invalid custom function");
    console.error(e);
}
```

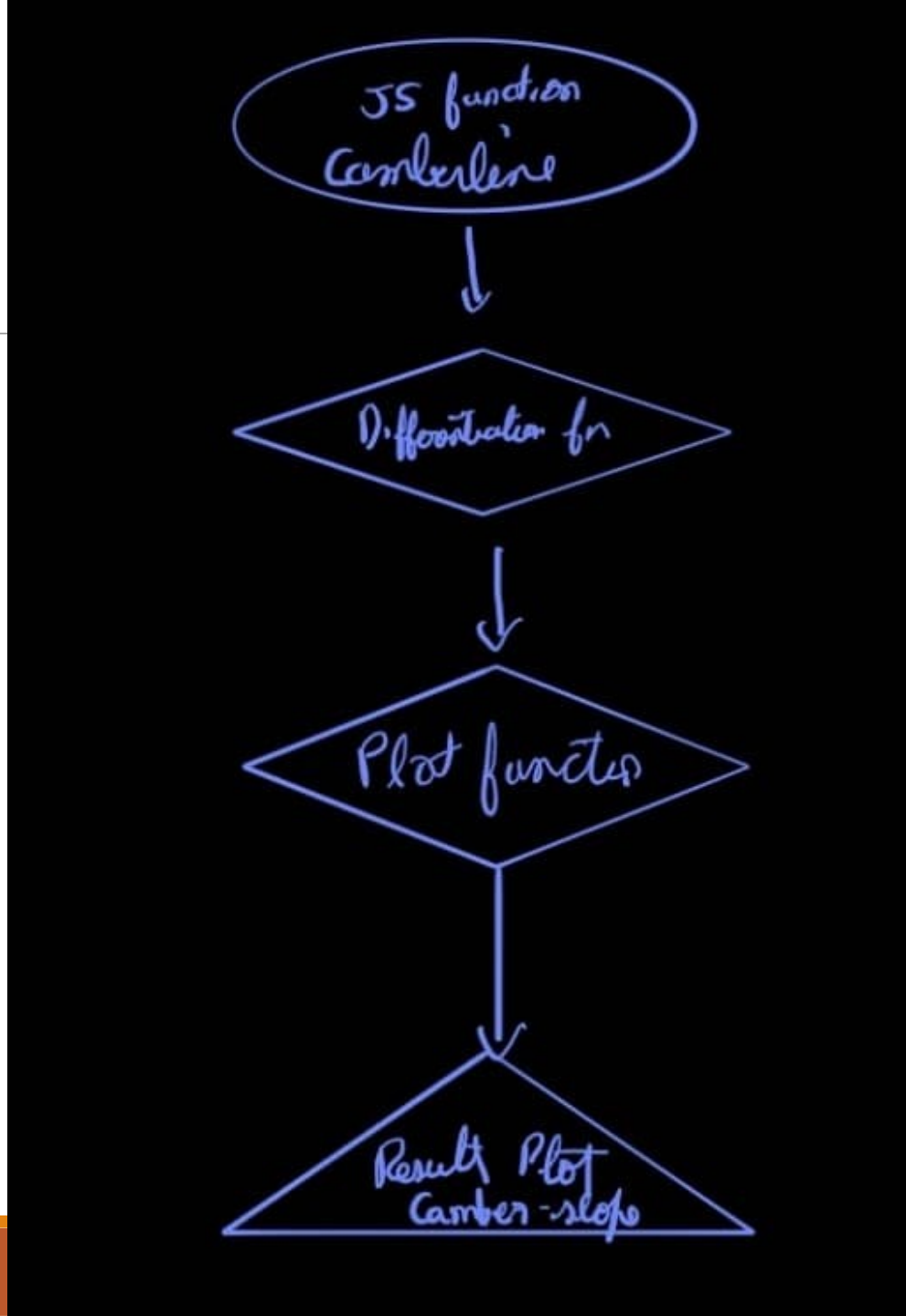
3. The plot function then uses HTML5 Canvas to plot the function with axes



2.2:
Algorithm for plotting camber line slope

1. **Create the camber slope function by using finite forward difference on the camber function, whatever it may be.**
2. **Pass the function into the general Plot function defined earlier**

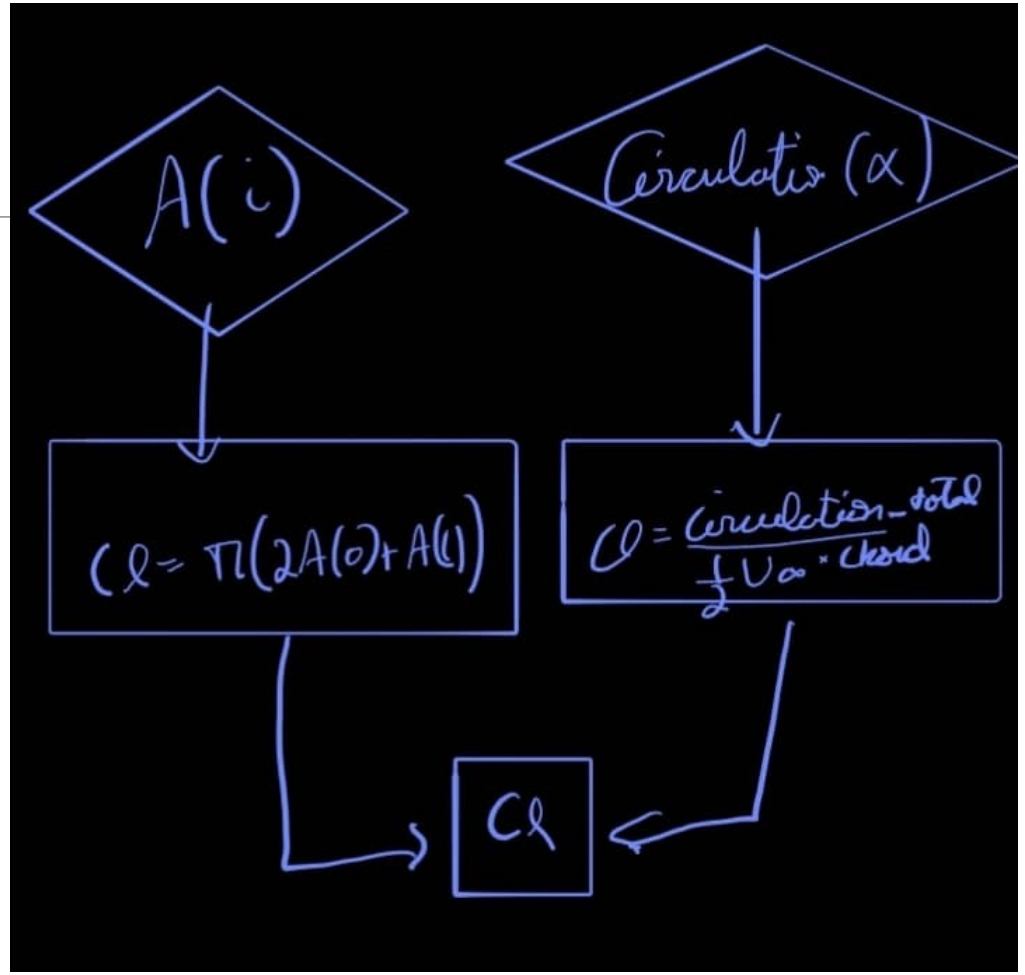
```
const camberSlope = (xVal: number): number => {
  const dh = 0.0001;
  return (camberFunction(xVal + dh) - camberFunction(xVal - dh)) / (2 * dh);
};
```



2.3:
Algorithm for computing C_l

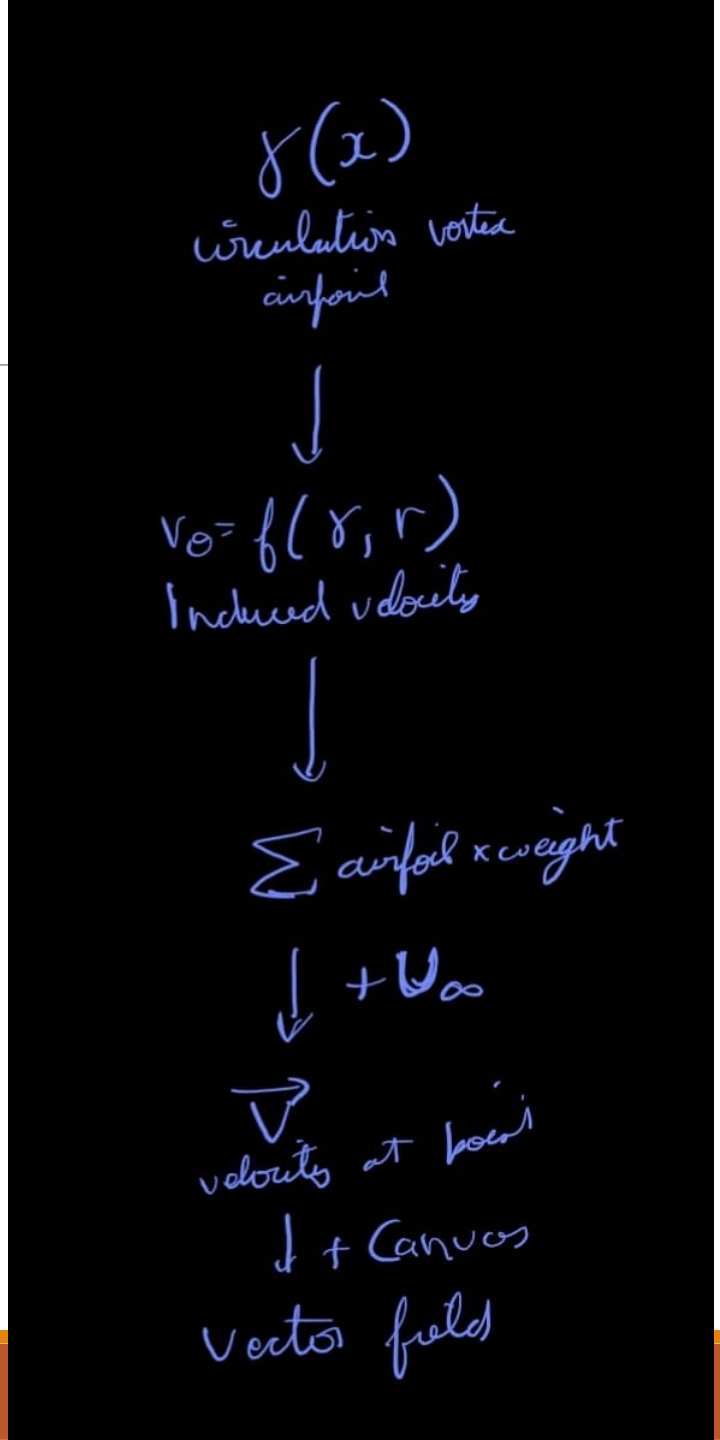
1. Create a general function for finding A_n, where A_n are the coeffecient of the fourier expansion of vortex circulation in terms of a parameter theta
2. Cl = pi*(2*A_0 + A1)
3. Alternatively, we can compute Cl by first evaluate circulation and equate lift/Span generated via circulation by Kutta Joukowski condition and by 0.5*rho*Cv*chord*(Freestream_Velocity^2)
4. Both methods are implemented and results displayed for user to compare.

```
const CLViaLift = circVal/(0.5*(Uinfty**1)*chordLength)
const CLViaAn = Math.PI*(2*AnCache[0] + AnCache[1])
```



2.4:
Algorithm for plotting vector field

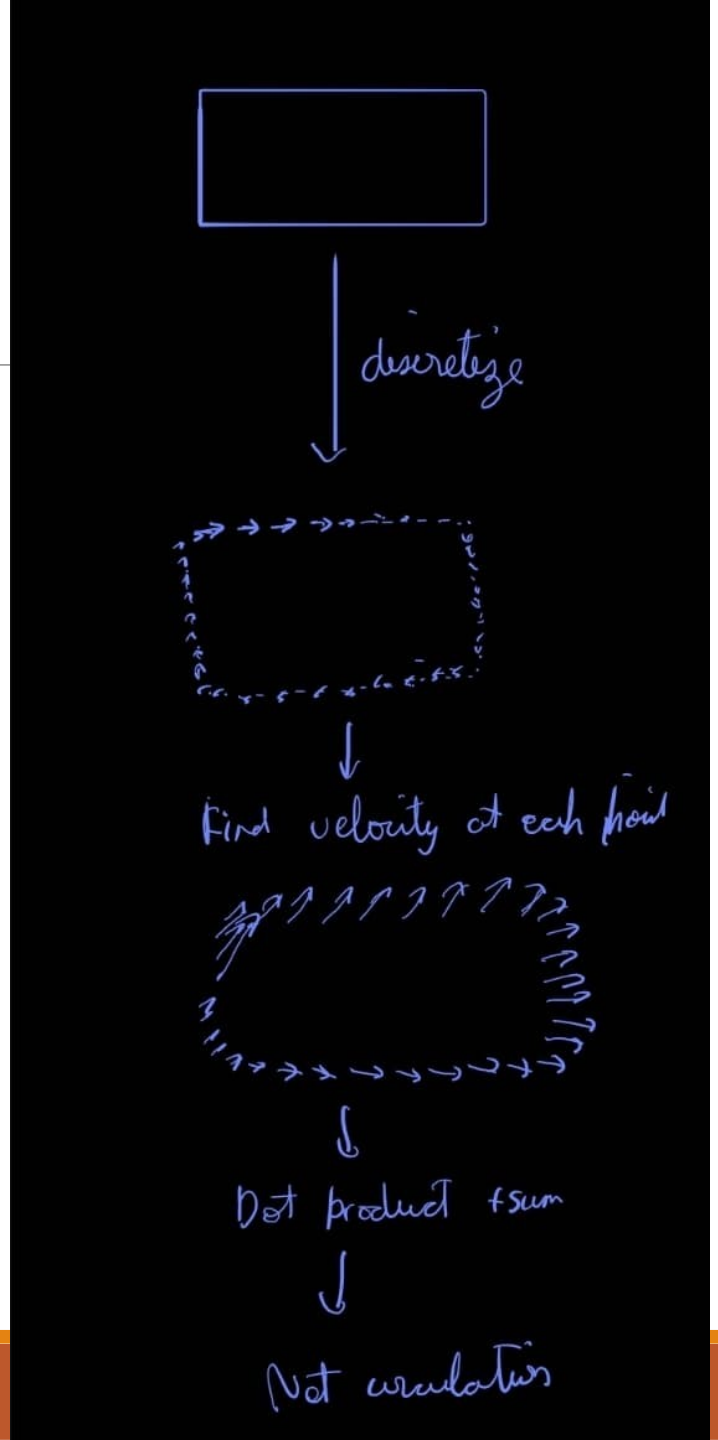
1. Create a function to plot velocity at any arbitrary point by using circulation of airfoil filaments.
2. Velocity is calculated by vector summing individual contributions of velocity induced from each filament times the filament length
3. Since space is discretized, we can calculate the length of the filament length using the camber slope by: dl = sqrt(1 + (dy/dx)^2) * dx.
4. The entire canvas is then discretized and velocity at node points is found and drawn using HTML5 Canvas.



Guide to writing algorithm:

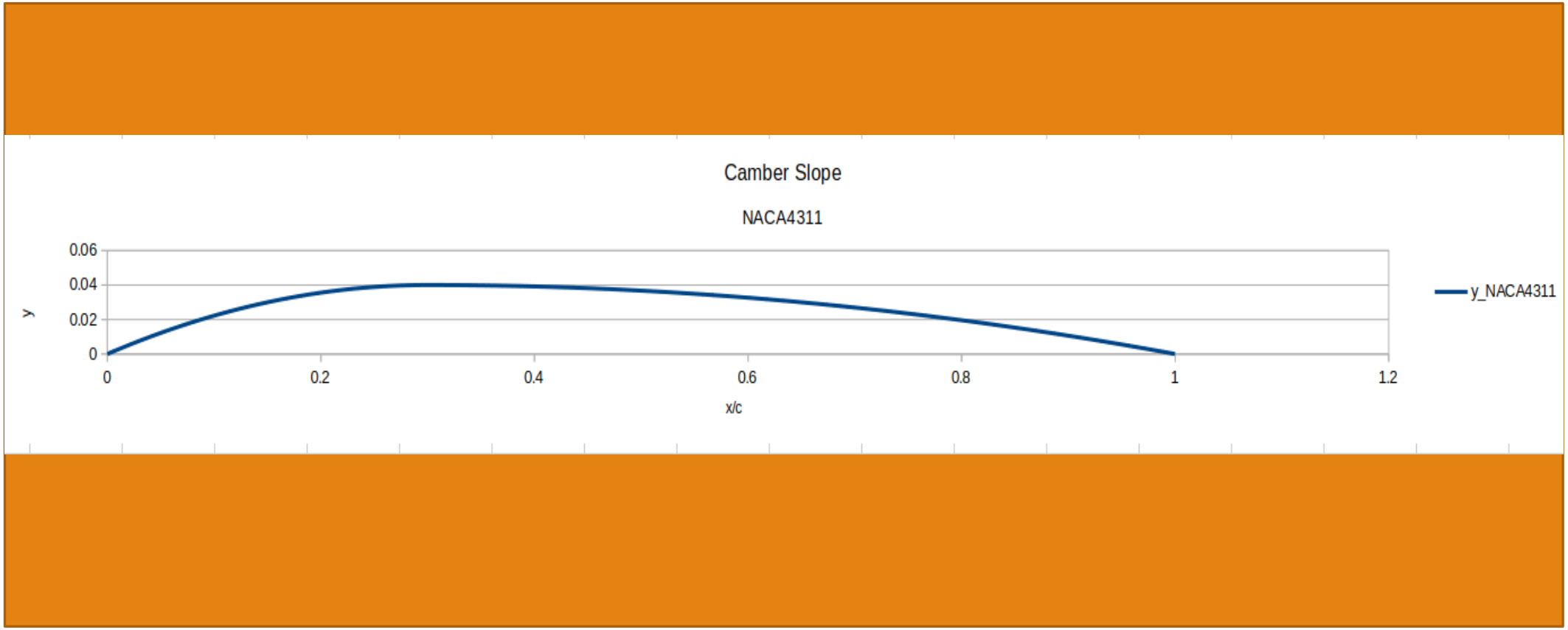
2.5: Algorithm for calculating circulation (through line integral)

1. Create an upright rectangular domain for convenience. Discretize this domain in regular manner
2. Find velocity at each node on this domain. For the vertical sides, use the y component of velocity and for the horizontal sides use the x component of velocity.
3. Find the delta circulation using V.ds, there V is the vector velocity and ds is the length of segment of domain. Note, when using the velocity-side combination as mentioned in step 2, the dot product reduces to a regular scalar product. (Sign must be taken care of as both +dx, +dy, -dx, -dy sides are present in the rectangle)
4. Sum up over each segment to get net circulation

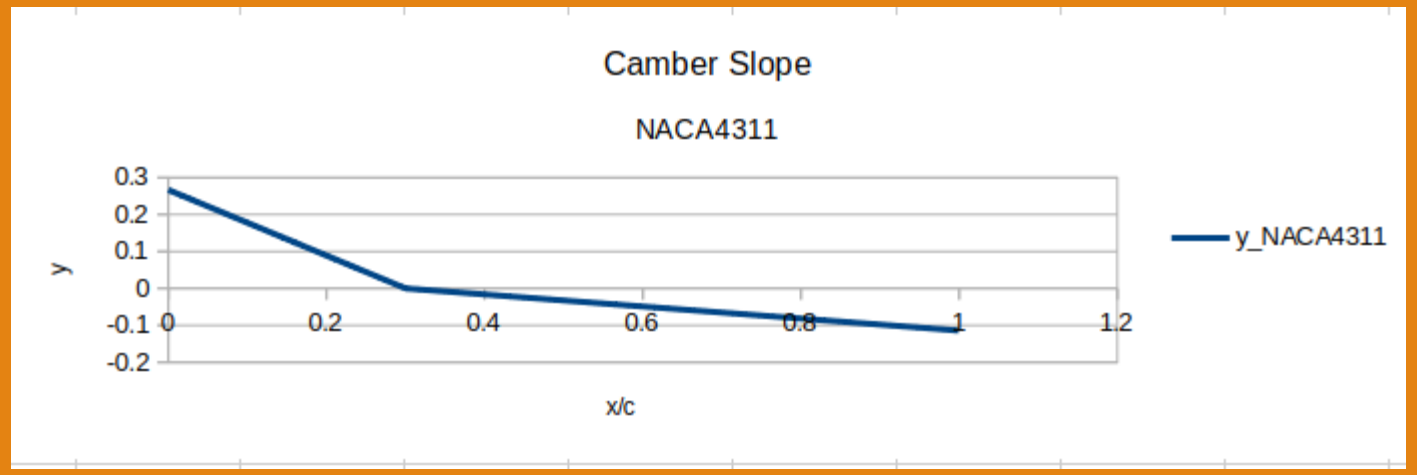


Section 3:
Airfoil Simulation and Results

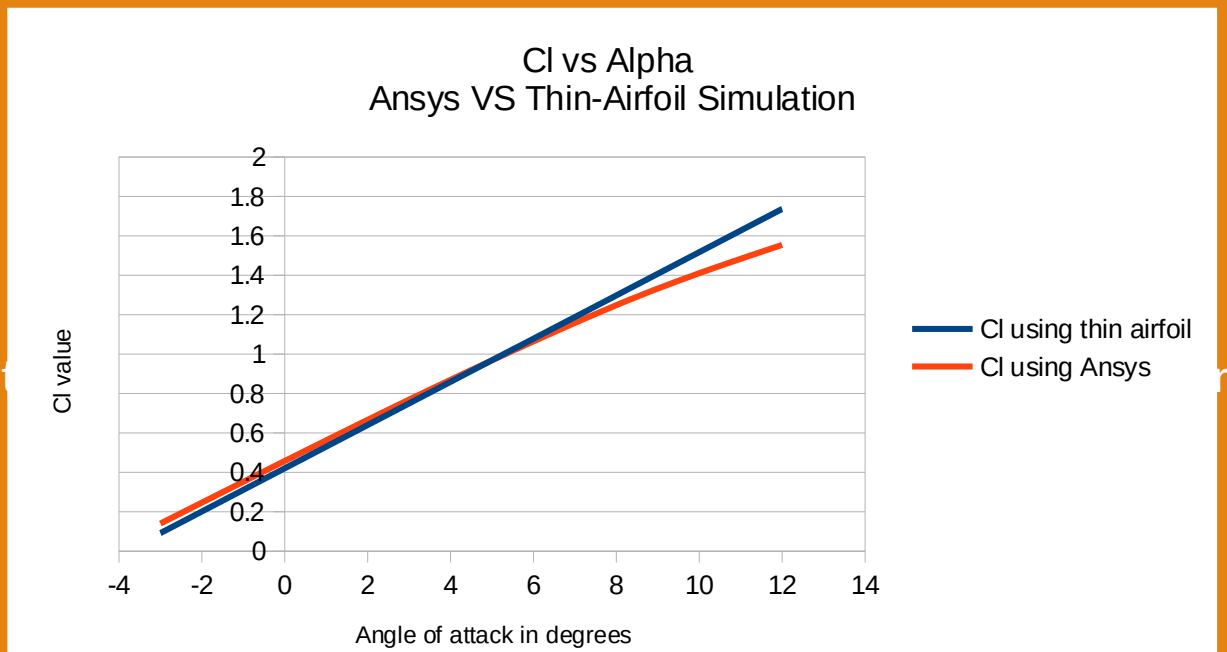
3.1:
Camber line (‘y’ vs ‘x’) for Roll Number airfoil



3.2:
Camber line slope (‘dy/dx’ vs ‘x’)



3.3, 3.4:
C_l vs α for the NACA airfoil

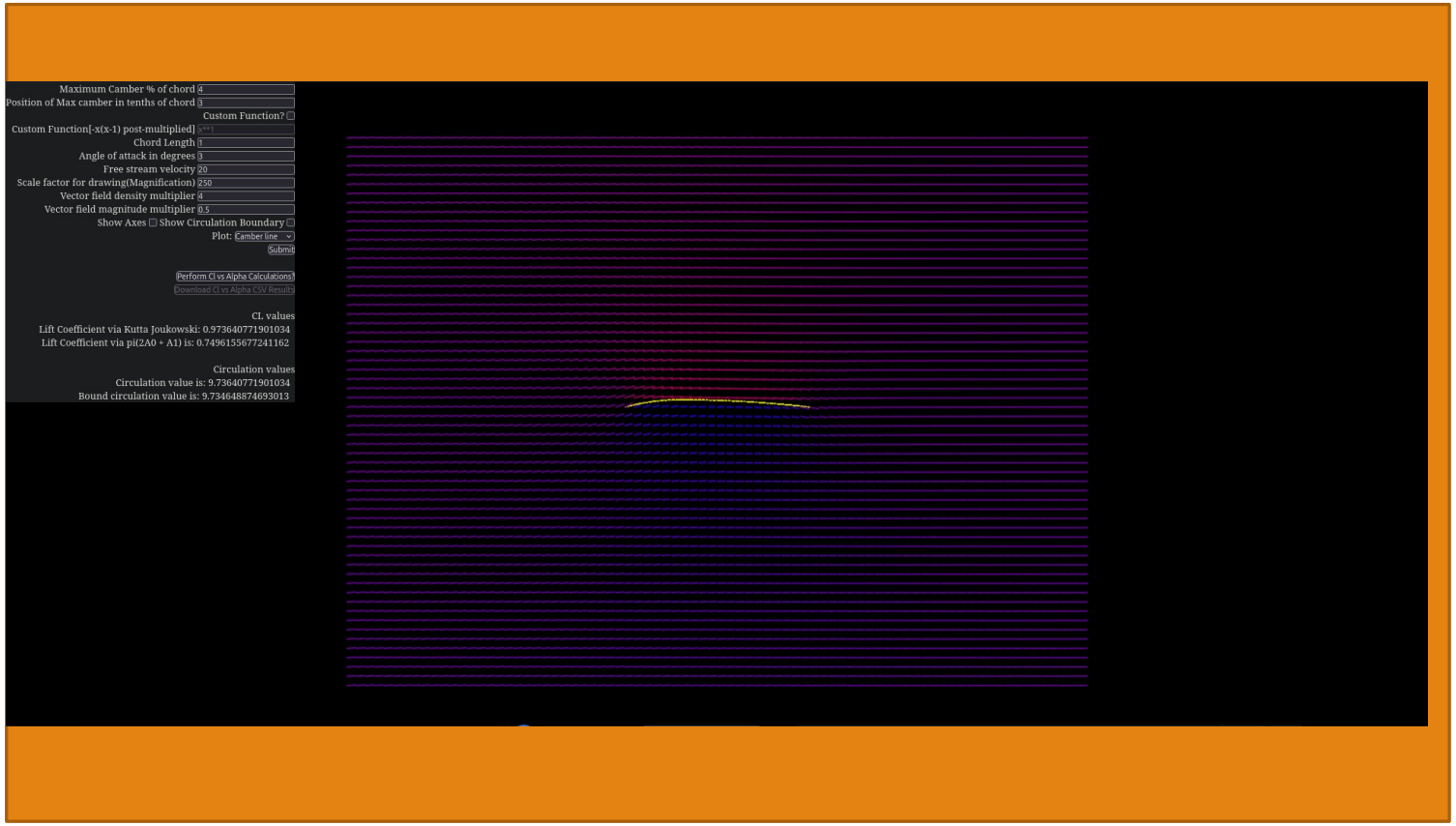


Plot

3.4: Discussion / Interpretation

- **Large similarity in the ~4 degrees region.**
- **Linearity is roughly similar.**
- **Deviation due to failure of stall and flow separation modelling**
- **Using thin-airfoil results in satisfactory answers for quick approximations**

3.5, 3.6: Vector field plot around the NACA airfoil



3.6: Discussion / Interpretation

- Streamlined vector field
- Acceleration above airfoil
- Valid physics

3.7, 3.8, 3.9: Circulation Computation and Discussion

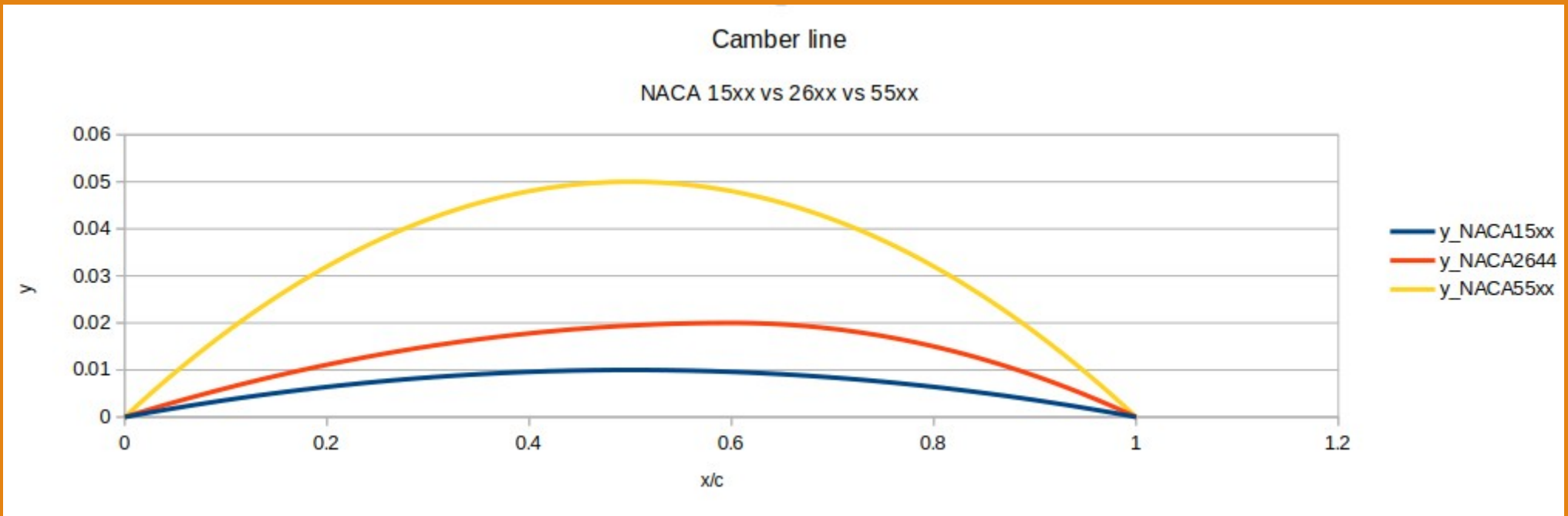
	Line Integral Method	Integrating Vortex Filaments
Circulation at $\alpha = 3°$	9.7364077	9.7346488

3.9: Compare and Discuss

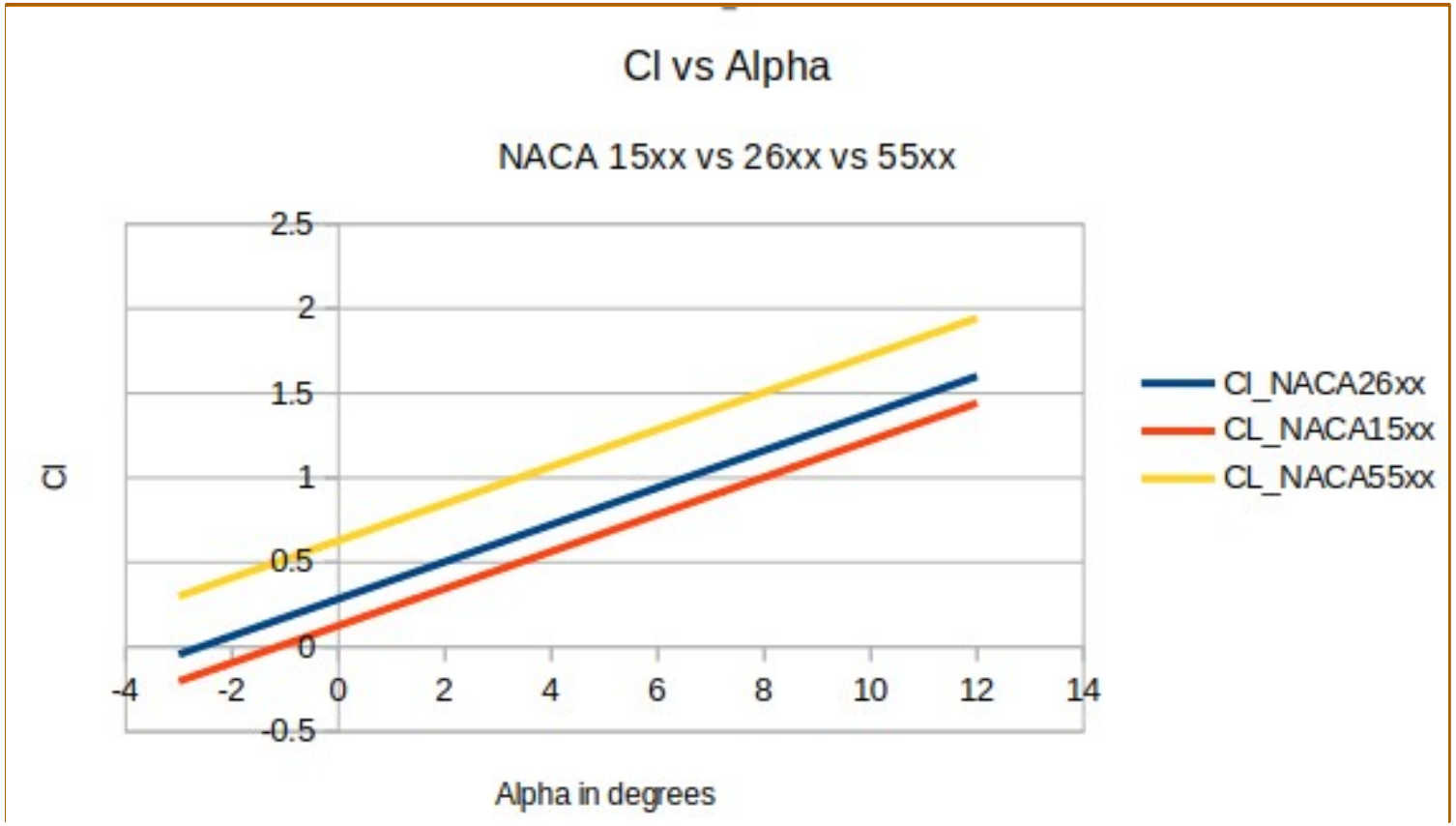
- **Strong agreement implying that vorticity to circulation conservation is roughly preserved**
- **Discretization of airfoil is good enough to provide accurate values**

Section 4:
Novel Airfoils

4.1:
Camber line (‘y’ vs ‘x’) for your 3 airfoils



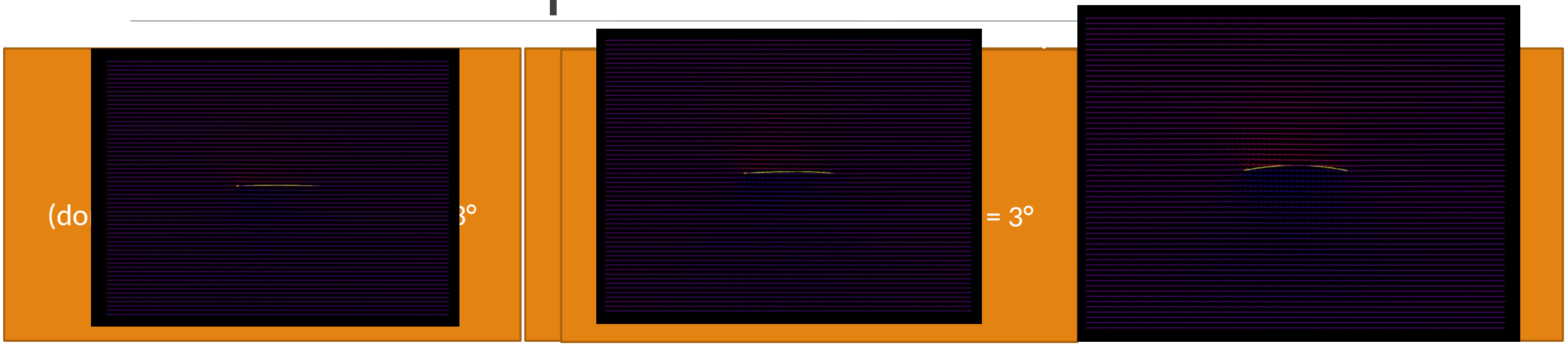
4.2, 4.3:
C_l vs α for your 3 airfoils



4.3: Discussion / Interpretation

- **Linear curve for all cambers**
- **Lack of flow separation due to theory limitation**
- **Higher Cl at zero degrees due to camber**
- **All lines parallel, same slope of 2pi**

4.4, 4.5:
Vector field plots around the 3 airfoils



NACA 15xx

NACA 26xx

NACA 55xx

4.5: Discussion / Interpretation

- **Streamlines roughly followed except in 55xx series**
- **Large camber => Deviation**
-

Section 5:
Conclusion

5.1: Comparison with Ansys

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

- **Significant speed up in terms of computation time for fairly accurate Cl values.**
- **Unable to model high camber airfoils unlike Ansys due to limitation of thin airfoil theory**
- **Additionally unable to loss of Cl due to flow separation/stall unlike Ansys**
- **Can be used for narrow band of operation to more optimal(faster + quality) solutions to airfoil parameters**

5.2: Comparison with NACA airfoil

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

- **For small angles of attack and thin airfoils the performance roughly matches.**
- **Deviation occurs at high AOA and at higher thickness values (last two digits of the MPXX series**
- **High AOA explained by failure of small angle assumption and flow separation neglection**
- **High thickness failure explained by inconsistency in modelling circulation of thick airfoil to lie condensed on chord**

Section 6: Code: Submit as a separate Zip File

MARKS COMMON TO THE TEAM MEMBERS

Section 7:
GUI

MARKS COMMON TO THE TEAM MEMBERS

7.1:

Attached GUI demonstration

Acknowledgement

None. This one I proudly did myself

References

None. Lot oof experience in JS.

Only reference: AE244 slides