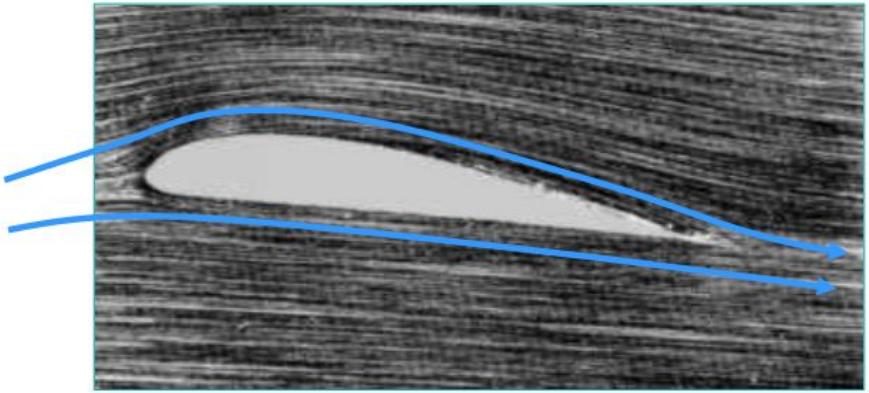
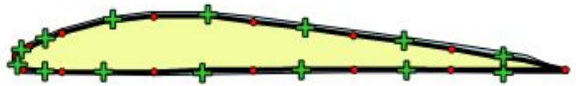
PiE : C++ Project Presentation

Kirtan Patel (s2935848)





$$\begin{pmatrix} V_n^{1,1} & V_n^{1,2} & V_n^{1,3} & \cdots & V_n^{1,N+1} \\ V_n^{2,1} & V_n^{2,2} & V_n^{2,3} & \cdots & V_n^{2,N+1} \\ V_n^{3,1} & V_n^{3,2} & V_n^{3,3} & \cdots & V_n^{3,N+1} \\ V_n^{4,1} & V_n^{4,2} & V_n^{4,3} & \cdots & V_n^{4,N+1} \\ V_n^{5,1} & V_n^{5,2} & V_n^{5,3} & \cdots & V_n^{5,N+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ V_n^{N,1} & V_n^{N,2} & V_n^{N,3} & \cdots & V_n^{N,N+1} \\ 1 & 0 & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \\ \gamma_5 \\ \gamma_6 \\ \vdots \\ \gamma_N \\ \gamma_{N+1} \end{pmatrix} = \begin{pmatrix} -V_{inf} * n_1 \\ -V_{inf} * n_2 \\ -V_{inf} * n_3 \\ -V_{inf} * n_4 \\ -V_{inf} * n_5 \\ \vdots \\ -V_{inf} * n_N \\ 0 \end{pmatrix}$$

Source : [Panel Methods : Mini-Lecture David Willis]

System of Linear Equations

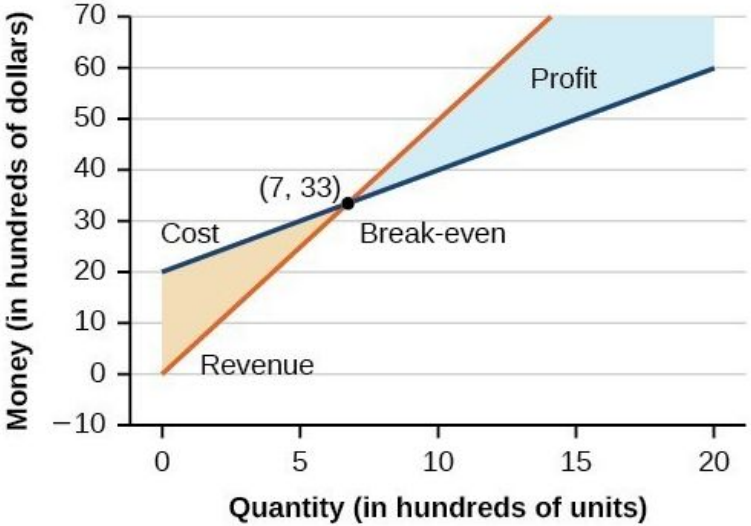
Whenever we have 2 or more variable quantities that satisfy some linear combination conditions, we can represent them as a System of Linear Equations

Other types are :

- System of Quadratic Equations (Stress Calculation)
- System of Differential Equations (Heat Transfer)

Examples of Systems of Linear Equations

- $8a - b = 9$
  $4a + 9b = 7$
- $3a - b + 14c = 7$
  $2a + 2b + 3c = 0$
  $a - 12b - 18c = 33$
- $3r + s - 7t = 15$
  $r - 12s + t = 0$
  $5s - 4t = 8$
- $x + 7y = 9$
  $2x - y = 18$

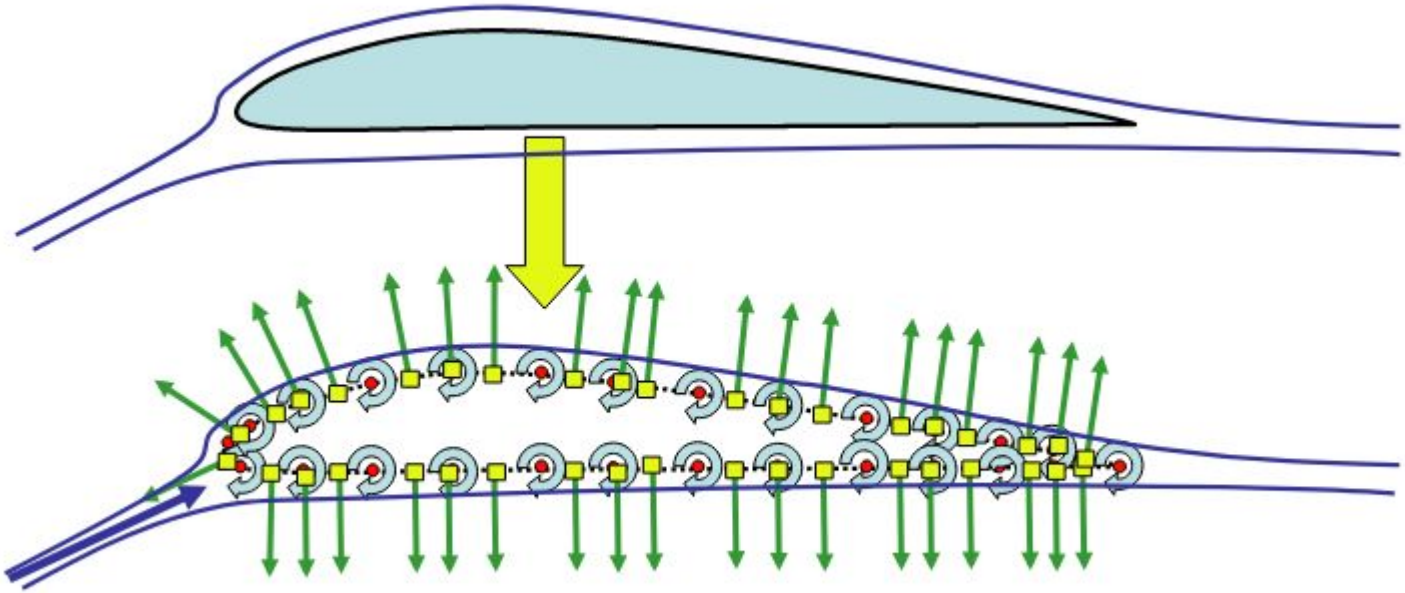


$$\begin{pmatrix} V_n^{1,1} & V_n^{1,2} & V_n^{1,3} & \cdots & V_n^{1,N+1} \\ V_n^{2,1} & V_n^{2,2} & V_n^{2,3} & \cdots & V_n^{2,N+1} \\ V_n^{3,1} & V_n^{3,2} & V_n^{3,3} & \cdots & V_n^{3,N+1} \\ V_n^{4,1} & V_n^{4,2} & V_n^{4,3} & \cdots & V_n^{4,N+1} \\ V_n^{5,1} & V_n^{5,2} & V_n^{5,3} & \cdots & V_n^{5,N+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ V_n^{N,1} & V_n^{N,2} & V_n^{N,3} & \cdots & V_n^{N,N+1} \end{pmatrix} \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \\ \gamma_5 \\ \gamma_6 \\ \\ \\ \gamma_N \\ \gamma_{N+1} \end{pmatrix} = \begin{pmatrix} V_{inf} * n_1 \\ V_{inf} * n_2 \\ V_{inf} * n_3 \\ V_{inf} * n_4 \\ V_{inf} * n_5 \\ \\ \\ \\ V_{inf} * n_N \end{pmatrix}$$

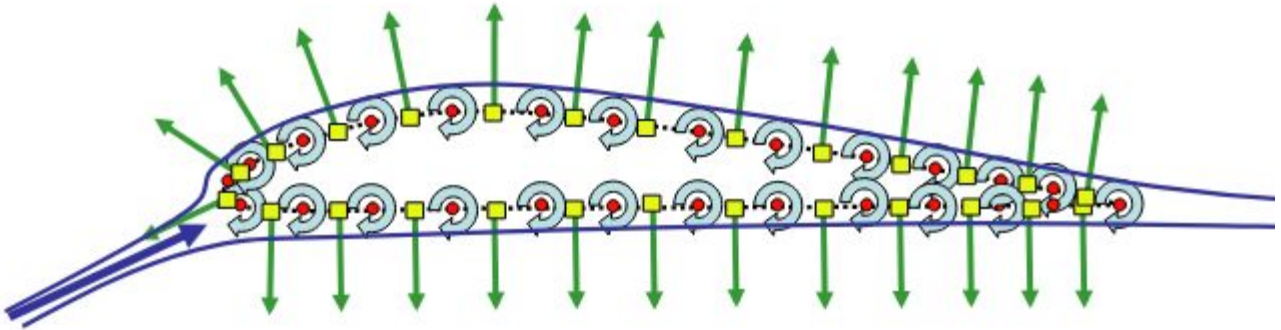
Vortex Panel Method

Vortex Panel Method

Panel methods belong to a broader class of methods called Boundary Element Methods (BEM).



Vortex Panel Method



- Flow tangency: The equation we want to satisfy at each control point j, on the airfoil:

$$\vec{V}_\infty \cdot \hat{n}_j + \sum_{i=1}^{NP} (\vec{v}_{\Gamma_i} \cdot \hat{n}_j) = 0$$

This equation can be solved numerically, and generates a system of linear equations, which need to be solved in order to obtain the strength of the panel vortices.

Vortex Panel Method

$$\vec{V}_\infty \cdot \hat{n}_j + \sum_{i=1}^{NP} (\vec{v}_{\Gamma_i} \cdot \hat{n}_j) = 0$$

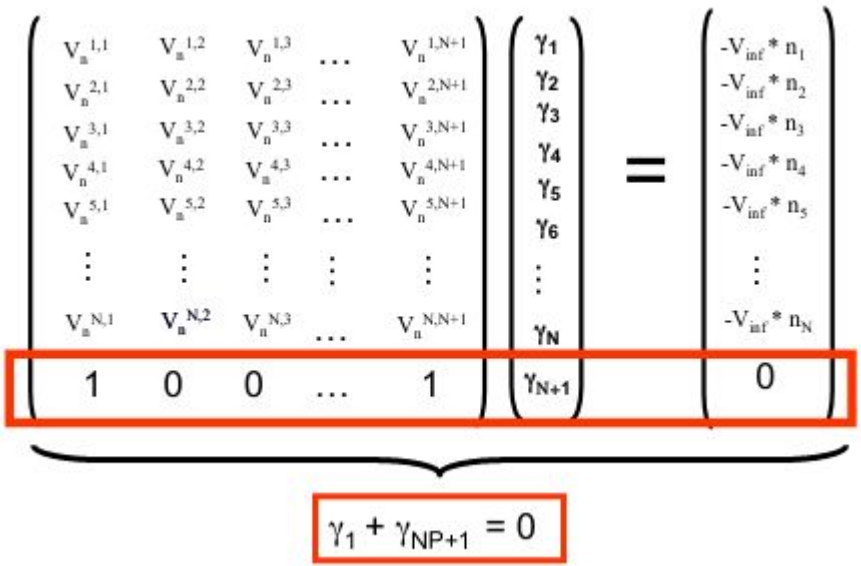
Translates to :

$$\begin{pmatrix} V_{inf} * n_1 \\ V_{inf} * n_2 \\ V_{inf} * n_3 \\ V_{inf} * n_4 \\ V_{inf} * n_5 \\ \\ \\ \\ V_{inf} * n_N \end{pmatrix} + \begin{pmatrix} V_n^{1,1} & V_n^{1,2} & V_n^{1,3} & \cdots & V_n^{1,N+1} \\ V_n^{2,1} & V_n^{2,2} & V_n^{2,3} & \cdots & V_n^{2,N+1} \\ V_n^{3,1} & V_n^{3,2} & V_n^{3,3} & \cdots & V_n^{3,N+1} \\ V_n^{4,1} & V_n^{4,2} & V_n^{4,3} & \cdots & V_n^{4,N+1} \\ V_n^{5,1} & V_n^{5,2} & V_n^{5,3} & \cdots & V_n^{5,N+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ V_n^{N,1} & V_n^{N,2} & V_n^{N,3} & \cdots & V_n^{N,N+1} \end{pmatrix} \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \\ \gamma_5 \\ \gamma_6 \\ \\ \\ \gamma_N \\ \gamma_{N+1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \\ \\ 0 \end{pmatrix}$$

Here, we can see that there are $n+1$ Vortex Panel Strengths, while the velocity matrix is $n \, X \, n$

This is due to the Kutta Condition, which ensures that there is no net vorticity at the Airfoil Trailing Edge

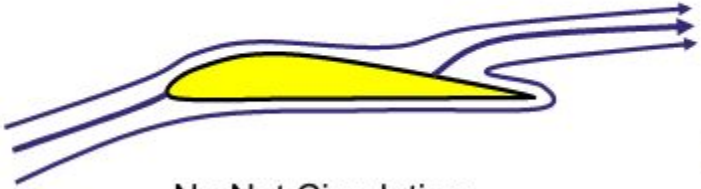
Vortex Panel Method



The last row has been replaced to [1,0,0……0,0,1], instead of it's original value in formula to ensure that the Kutta Condition is satisfied, since otherwise the system of equations becomes over-constrained.

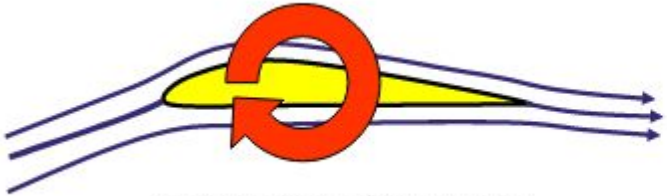
Target-Audience

Now we want to check whether the solution we get (without the Kutta Condition) gives us a lifting flow. Without the condition, we can get the different possible solutions :

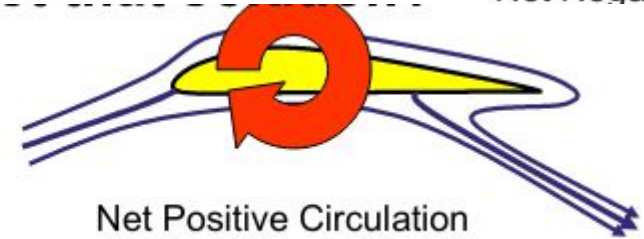


No Net Circulation

Net Negative Circulation



Net Positive Circulation



Net Positive Circulation

A Positive Circulation means that the wing generates lift, and that is what we want. This occurs when the Kutta Condition is satisfied by the vortex panel strengths. ***My C++ implemented project solves the system of linear equations for the Vortex Panel Strength, and checks if the Kutta Condition is satisfied or not.***

The LUDecomp.h and solve.h files can be used in general to solve a System of Linear Equations

LU Decomposition

$$A = LU, \text{ where } A \text{ is a } 2 \times 2 \text{ square matrix}$$

$$(1)\ A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix} = LU, \text{ where } A \text{ is a } 2 \times 2 \text{ square matrix}$$

$$(2) A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} = LU,$$

$$\text{when } A \text{ is } 3 \times 3 \text{ square matrix}$$

Example :

$$\begin{bmatrix} 1 & 1 & -1 \\ 6 & 2 & 2 \\ -3 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 6 & 1 & 0 \\ -3 & -1.75 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 \\ 0 & -4 & 8 \\ 0 & 0 & 12 \end{bmatrix}.$$

Design Decisions

Variables Used

```
float A[n][n];
float B[n][1];
```

The only 2 large (memory complexity) variables that have been used in main() are the ones which store the input. Operations have been carried out on them using pointers.

Effective use of operations and algorithm

```
float lowertriangle[n][n];

float *uppertriangle_ptr,*lowertriangle_ptr,*B_ptr;

lowertriangle_ptr = (float*)lowertriangle;
uppertriangle_ptr = (float*)A;
B_ptr = (float*)B;
```

Upper Triangular Matrix is what remains in A after the above LU Decomposition operations hence, we can just use the upper-triangle pointer to initially point to A. After the LU decomposition, A transforms into an upper-triangle matrix, and hence the pointer is valid this is useful when the size of A starts increasing it is better to use a pointer, instead of creating a copy, which is an inefficient use of memory

Design Decisions

Use of pointers

```
lowertriangle_ptr = (float*)lowertriangle;
uppertriangle_ptr = (float*)A;
B_ptr = (float*)B;

LUDecomposition(uppertriangle_ptr,lowertriangle_ptr,n);

float* Soln_X = Solve_LU_x_B(uppertriangle_ptr,lowertriangle_ptr,B_ptr,n);
```

Pointers have been used to call functions. This is better than passing the entire matrix as a parameter while calling the function

Using a Vector to store objects

```
std::vector<Variable> soln_variables;

for(i = 0; i < n; i++)
{
        // naming the solution variables as letters starting from 'a' using ASCII codes
        char var_name = (char)(97+i);
        soln_variables.push_back(Variable(var_name,(*(Soln_X+i))));
}
```

Since the number of variables of the system of equations is not hard-coded, we need to generate n objects to store their data. Thus using a vector to store objects is a convenient method.