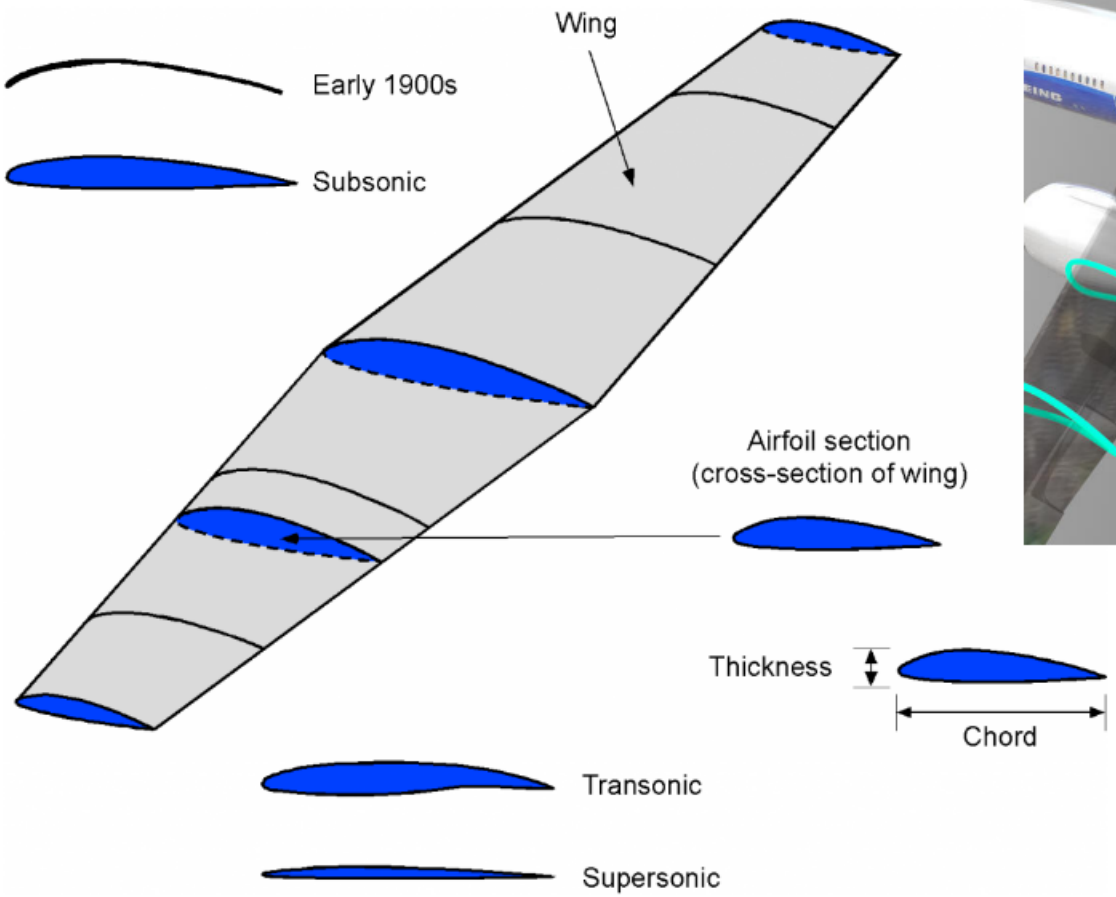
Código de vórtices discretos: una herramienta de análisis aerodinámico para perfiles

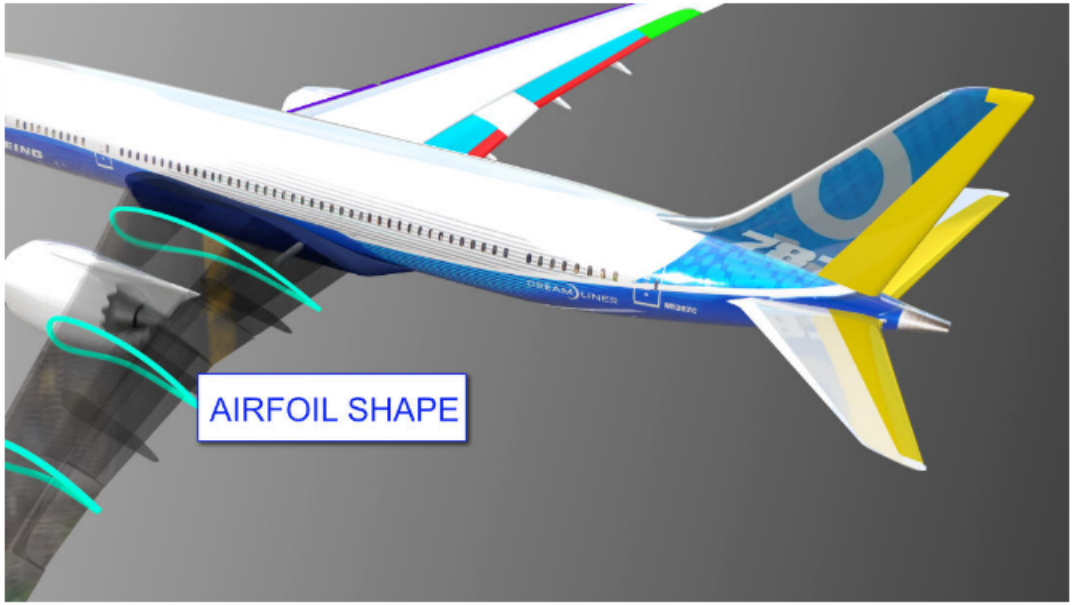
Pablo N. Zitelli
Ingeniería Aeronáutica - UTN Facultad Regional Haedo

Motivación

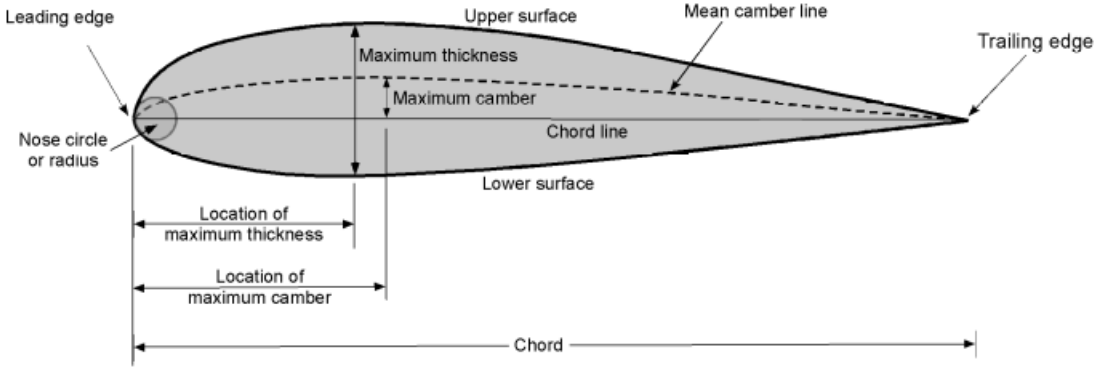
- Aprender y reforzar conceptos fundamentales de aerodinámica en la carrera de ingeniería.
- Motivar a los alumnos para aprender a programar en el contexto de computación científica.
- Comprender cómo pasar del dominio teórico al práctico mediante el código en lenguaje de programación.
- Analizar resultados, obtener conclusiones, fortalecer el criterio ingenieril y resolver problemas.
- Tener herramientas para atacar problemas más complejos.

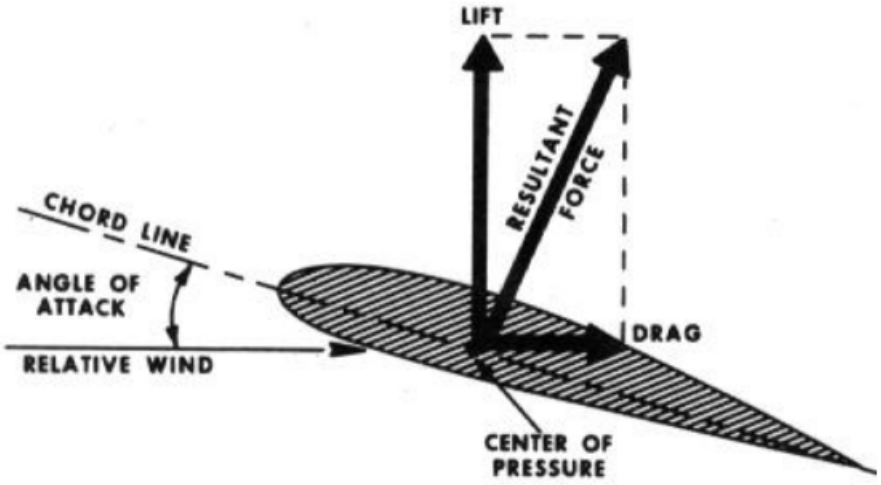
Intro Teórica (*avgeek alert!*): ¿Qué es un perfil aerodinámico?





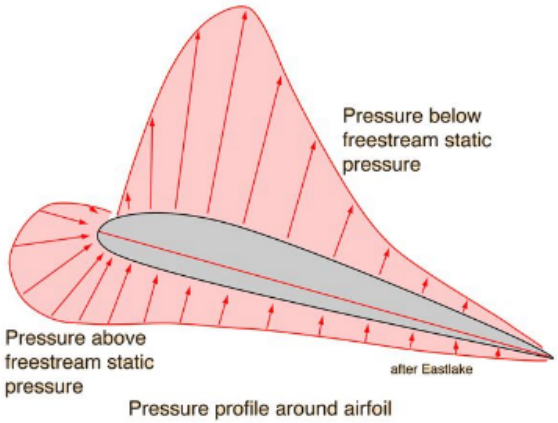
Intro Teórica: Fuerzas aerodinámicas sobre un perfil, a partir del campo de presiones y por *viscosidad* del fluido

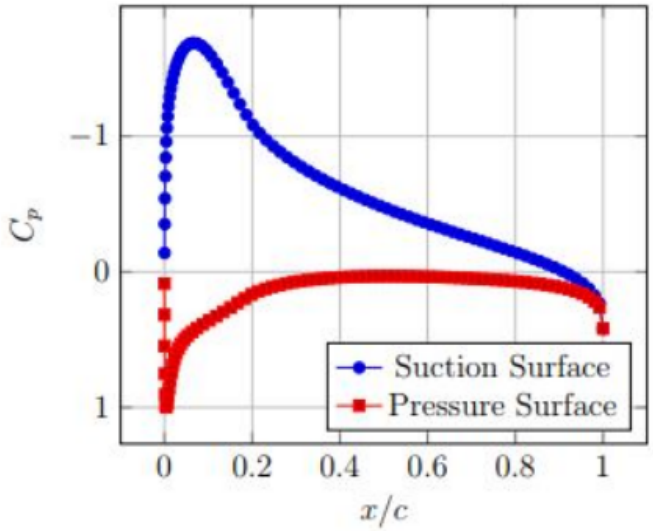




$$C_p = \frac{2\,(p - p_\infty)}{\rho V_\infty^2} = 1 - \left(\frac{v}{V_\infty}\right)^2$$

$$C_L = \frac{2L}{\rho V_\infty^2 c}$$





Intro Teórica: Concepto de Flujo Potencial

Ecuaciones de Navier-Stokes: gobiernan la dinámica de fluidos

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}\cdot\boldsymbol{\nabla}\mathbf{v}\right) = \rho\,\mathbf{f} - \boldsymbol{\nabla}p + \mu\nabla^2\mathbf{v}$$

Fuera de la *capa límite* y de la *estela de vorticidad* detrás el perfil o ala, el campo de velocidades es *irrotacional*, por lo tanto se puede asumir que se comporta *como si* no tuviese viscosidad. Entonces, se puede demostrar que el campo de velocidades es igual al gradiente de una función escalar, a la que denominamos *función potencial de velocidades*:

$$\boldsymbol{\nabla}\times\mathbf{v} = \mathbf{0} \quad \Rightarrow \quad \mathbf{v} = \boldsymbol{\nabla}\phi$$

Para flujo incompresible y estacionario se demuestra que el campo de velocidades tiene divergencia nula, por lo tanto:

$$\boldsymbol{\nabla}\cdot\mathbf{v} = 0 \quad \Rightarrow \quad \boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\phi = \nabla^2\phi = 0$$

Ahora el problema queda definido por la Ecuación de Laplace y dos condiciones de contorno:

$$\nabla^2\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = 0$$

$\mathbf{v}\cdot\hat{\mathbf{n}} = 0$ sobre contorno del objeto

$\mathbf{v} = \mathbf{V}_\infty$ lejos del objeto

Intro Teórica: Soluciones fundamentales, el *vórtice libre o discreto*

Existen varias soluciones fundamentales de la Ecuación de Laplace: fuentes y sumideros, dobletes, vórtices. Si asumimos un dominio bidimensional, la expresión del potencial de velocidades para un vórtice libre y las componentes de la velocidad inducida por el mismo serán (en coordenadas cartesianas y en polares):

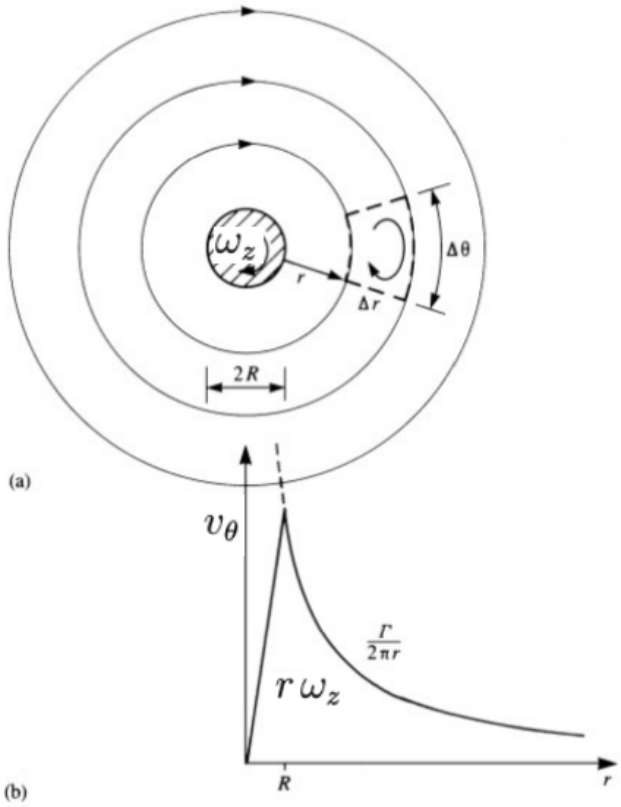
$$\phi(x,y) = -\frac{\Gamma}{2\pi}\,\text{arctg}\left(\frac{y-y_0}{x-x_0}\right)$$

$$u(x,y) = \frac{\Gamma}{2\pi}\frac{y-y_0}{(x-x_0)^2+(y-y_0)^2}$$

$$v(x,y) = -\frac{\Gamma}{2\pi}\frac{x-x_0}{(x-x_0)^2+(y-y_0)^2}$$

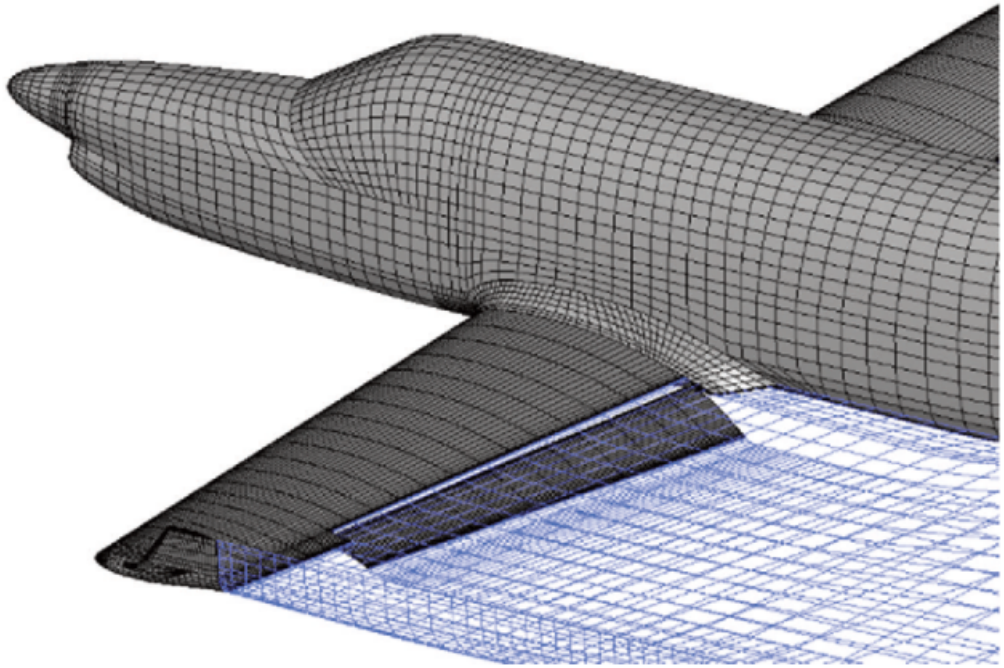
$$\phi(r,\theta) = -\frac{\Gamma}{2\pi}\theta$$

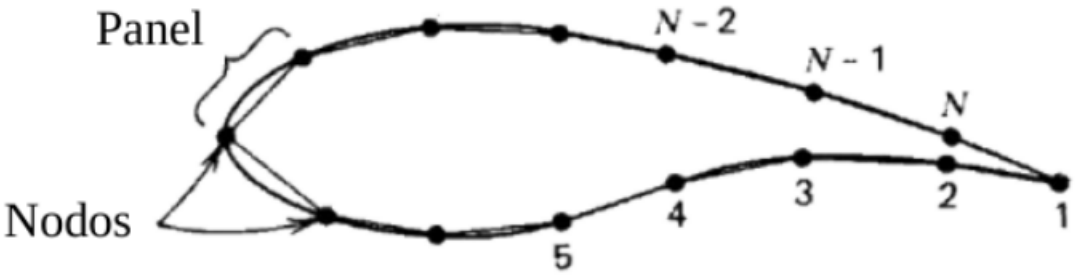
$$v_\theta(r,\theta) = -\frac{\Gamma}{2\pi r}$$



Método de los Paneles: Generalidades

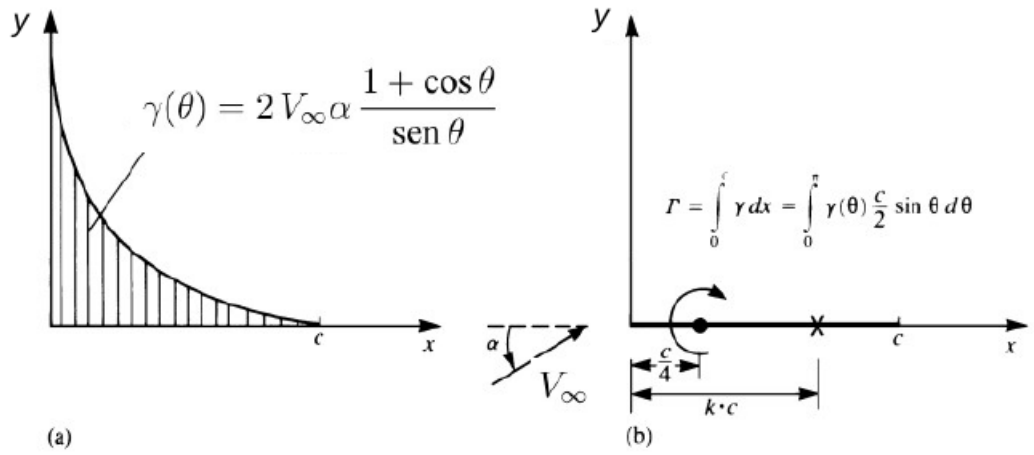
Se discretiza el ala (problema 3D) o perfil (2D) en paneles o segmentos, respectivamente, sobre los cuales se distribuye algún tipo de *singularidad* (solución fundamental de la Ecuación de Laplace) cuya intensidad es incógnita, y se impone la condición de contorno de tangencia del flujo sobre cada elemento. Para determinar la velocidad total en cada elemento se calcula la velocidad inducida por la influencia de la disrtribución de cada panel sobre el que es considerado localmente. De esta manera se construye un sistema lineal de ecuaciones, donde cada ecuación es la velocidad total en cada panel, debida a la influencia de todos los paneles (incluida la de sí mismo) multiplicada escalarmente por el versor normal de dicho panel, e igualada a cero.





Método de los Paneles: *lumped-vortex* o vórtices discretos

A partir de la solución para un perfil tipo *placa plana* empleando la *Teoría de Perfil Delgado*, tenemos:



De aplicar la BC de tangencia del flujo y teniendo en cuenta que la circulación equivalente de la placa plana es $\Gamma = \pi\alpha\, c\, V_\infty$, se obtiene que el *punto de colocación* corresponde a $x = 3/4\, c$. La sustentación de un perfil la obtenemos mediante el Teorema de Kutta-Joukowski:

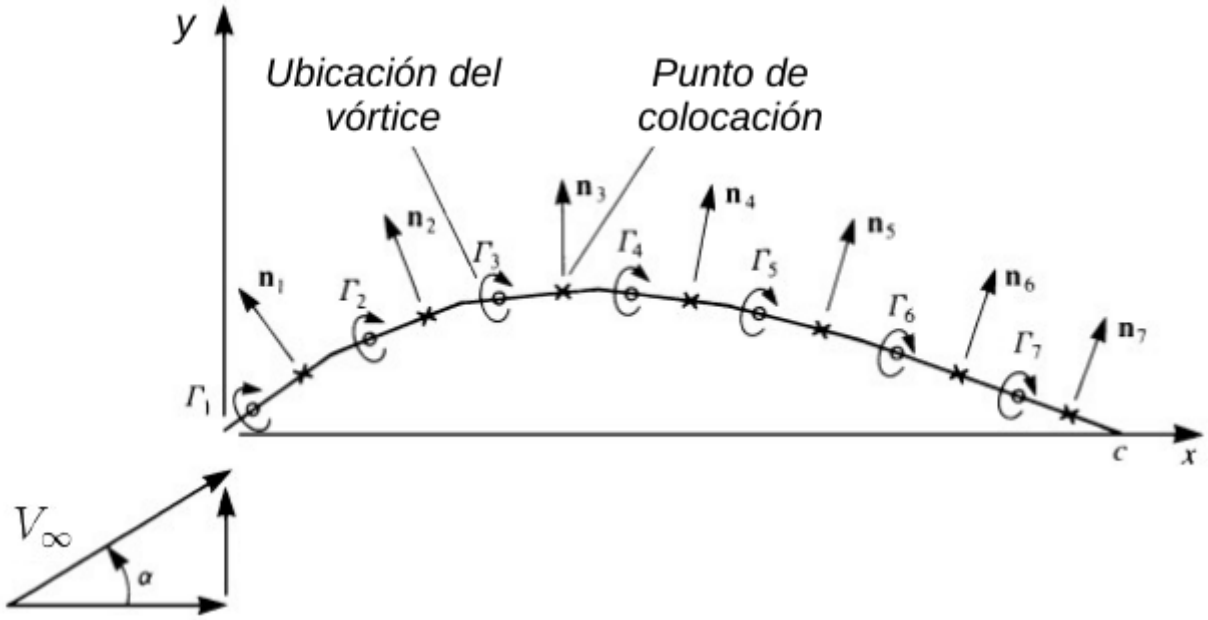
$$L = \rho V_\infty \Gamma$$

Y para el caso en el que dos o más perfiles interactúan entre sí, la sustentación de cada uno se calcula con el Teorema Generalizado de Kutta-Joukowski:

$$L_i = \rho V_\infty \Gamma_i \left(1 + \frac{\mathbf{V}_\infty \cdot \mathbf{v}_i^v}{V_\infty^2}\right)$$

Método de los Paneles: *lumped-vortex* o vórtices discretos

Con esta variante del método de lo paneles se discretiza la *línea de curvatura media*, en vez del contorno del perfil.



Una manera pragmática y metódica de escribir la velocidad inducida en el punto de colocación del panel i debida al vórtice del panel j, pensando en el código computacional, es la siguiente:

$$\begin{pmatrix} u \\ v \end{pmatrix}_i = \frac{\Gamma_j}{2\pi\, r_{ij}^2} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{pmatrix} x_i - x_j \\ y_i - y_j \end{pmatrix}$$

siendo $r_{ij}^2 = (x_i - x_j)^2 + (y_i - y_j)^2$

Código (*¡al fin!*): Estructura de archivos

El proyecto trabaja con los siguientes archivos:

- `main.py`: Contiene el workflow de todo el prodecimiento de definiciones, seteo de problema, solución del sistema lineal, obtención de resultados y ploteo.
- `utils.py`: Alberga la definición de clases y algunos métodos de uso recurrente.
- `coord_airfoil_front.dat` & `coord_airfoil_rear.dat`: Coordenadas de los nodos para definir los perfiles del problema (en este caso, dos).
- `results.dat`: Donde se escriben los resultados finales.

Código: archivo `utils.py`

```python
# node class definition
class Node:
    def __init__(self, coord, name):
        self.coord = coord
        self.id = name
```

```python
# panel class definition
class Panel:
    def __init__(self, nodes, name):
        self.unitVelInd = []
        self.nodes = nodes
        self.id = name
        self.lift = 0.0

        # tangent unit vector and panel length
        tVec = self.nodes[1].coord - self.nodes[0].coord
        self.len = np.linalg.norm(tVec)
        self.t = tVec/self.len
        ...
```

```python
# airfoil class definition
class Airfoil:
    def __init__(self, name, coordFile):
        self.name = name
        self.panels = []
        ...
```

Código: archivo `utils.py`

```python
# write results to file and terminal
def writeResults(airfoils):
    with open('results.dat', 'w') as out:
        print(40*'=' + '\n')
        out.write(40*'=' + '\n')

        for a in airfoils:
            print('Airfoil {0:s} aerodynamic characteristics:'.format(a.name))
            print('Gamma = {0:.4f} m2/s'.format(a.Gamma))
            print('L = {0:.4f} N/m'.format(a.L))
            print('Cl = {0:.4f}\n'.format(a.Cl))
            print(40*'=' + '\n')

            out.write('Airfoil {0:s} aerodynamic characteristics:\n'.format(a.name))
            out.write('Gamma = {0:.4f} m2/s\n'.format(a.Gamma))
            out.write('L = {0:.4f} N/m\n'.format(a.L))
            out.write('Cl = {0:.4f}\n'.format(a.Cl))
            out.write(40*'=' + '\n')
```

Código: archivo `main.py`

```
# aerodynamic parameters
alphaDeg = 10.0
vInf = 1.0

# actual code

# airfoils initialization
airfoils = []

airfoils.append(Airfoil('Front', 'coord_parabolic_front_20.dat'))
airfoils.append(Airfoil('Rear', 'coord_parabolic_rear_30.dat'))
#airfoils.append(Airfoil('Flat plate', 'coord_flat_plate_10.dat'))

# induced velocity on panel i collocation point due to unit vortex on panel j (all airfoils)
for am in airfoils:
    for pi in am.panels:
        for an in airfoils:
            for pj in an.panels:
                pi.unitVelIndCalc(pj.qPoint)

# RHS (independent vector) definition
N = 0
for a in airfoils:
    N = N + a.panelNum

b = np.zeros(N)
vectorInf = np.array([vInf*np.cos(alphaDeg*np.pi/180.0), vInf*np.sin(alphaDeg*np.pi/180.0), 0.0])

i = 0
for a in airfoils:
    for p in a.panels:
        b[i] = -np.dot(vectorInf, p.n)
        i = i + 1
```

Código: archivo `main.py`

```
# influence coefficients matrix definition
A = np.zeros((N, N))

i = 0
for a in airfoils:
    for pi in a.panels:
        for j in range(N):
            A[i,j] = np.dot(pi.unitVelInd[j], pi.n)

        i = i + 1

# linear system solution
gamma = np.linalg.solve(A, b)

# results
i = 0
for a in airfoils:
    for p in a.panels:
        p.setGamma(gamma[i])
        p.dCpCalc(vInf)
        i = i + 1

# airfoil lift
for a in airfoils:
    for p in a.panels:
        p.totalIndVelCalc(totalPanels)
        p.liftCalc(vectorInf)

    a.liftCalc(vInf)

writeResults(airfoils)
```

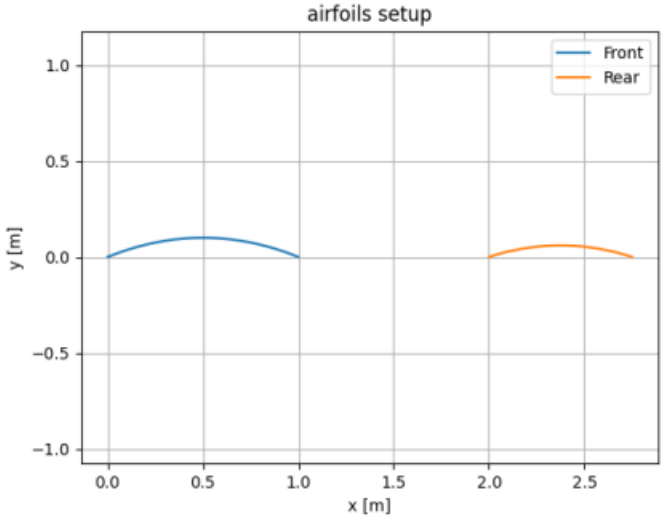
Código: archivo `main.py`

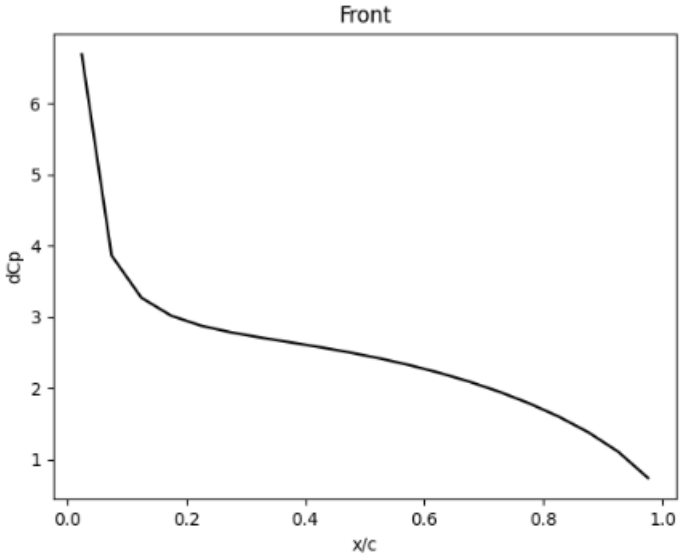
```
# plot airfoils
airfoilsNameList = []
fig, ax = plt.subplots()
for a in airfoils:
    airfoilsNameList.append(a.name)
    x = [p.nodes[0].coord[0] for p in a.panels]
    x.append(a.panels[-1].nodes[1].coord[0])
    y = [p.nodes[0].coord[1] for p in a.panels]
    y.append(a.panels[-1].nodes[1].coord[1])
    ax.plot(x, y, label=a.name)

plt.axis('equal')
plt.title('airfoils setup')
plt.xlabel('x [m]')
plt.ylabel('y [m]')
plt.legend(airfoilsNameList)
plt.grid()

# delta Cp along airfoils
f = 2
for a in airfoils:
    # chord vector
    cVec = a.panels[-1].nodes[1].coord - a.panels[0].nodes[0].coord

    sC = [np.dot(p.mPoint-a.panels[0].nodes[0].coord, cVec)/a.c for p in a.panels]
    dCpPlot = [p.dCp for p in a.panels]

    plt.figure(f)
    plt.title(a.name)
    plt.plot(sC, dCpPlot, 'k')
    plt.xlabel('x/c')
    plt.ylabel('dCp')
    f = f + 1

plt.show()
```

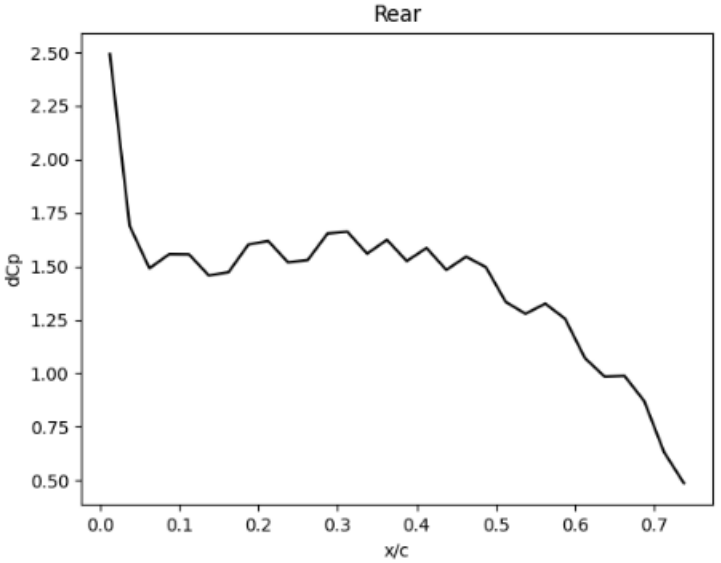
Código: Librerías usadas

- **SciPy**: Contiene módulos para optimización, álgebra lineal, integración, interpolación, funciones especiales, FFT, procesamiento de señales y de imagen, resolución de ODEs y otras tareas para la ciencia e ingeniería.
- **NumPy**: Da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas.
- **Matplotlib**: Es una biblioteca para la generación de gráficos en dos dimensiones, a partir de datos contenidos en listas o arrays

Código: Resultados








```
========================================
Airfoil Front aerodynamic characteristics:
Gamma = 1.3003 m2/s
L = 1.5737 N/m
Cl = 2.6228
========================================
Airfoil Rear aerodynamic characteristics:
Gamma = 0.5377 m2/s
L = 0.6319 N/m
Cl = 1.4043
========================================
```

¡Muchas gracias por su atención! (y paciencia...)

LinkedIn: @pablozitelli
GitHub: @pzitelli84