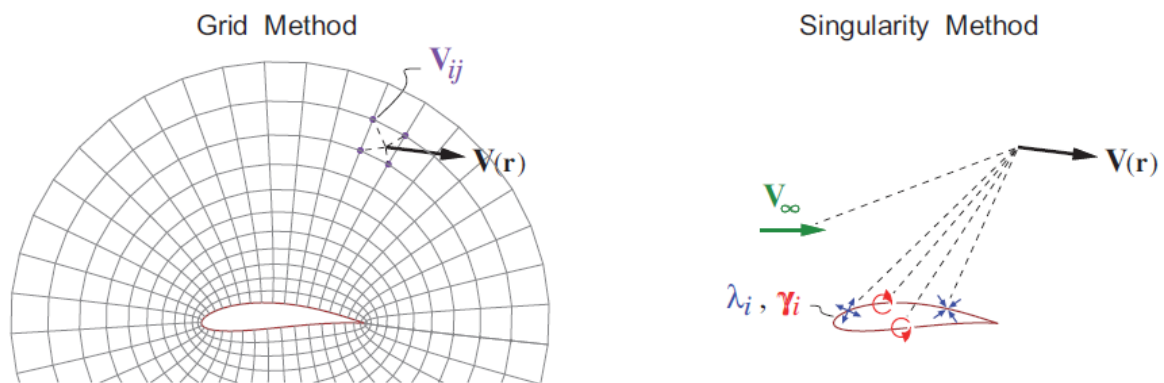


# Teoría potencial linealizada 2D

La teoría potencial linealizada bidimensional se basa en la resolución de la ecuación de Laplace en dos dimensiones para un fluido incompresible y estacionario. En esta teoría, se asume que las fluctuaciones en la velocidad y la presión son pequeñas en comparación con las velocidades y presiones características del fluido. Frente a una resolución de las ecuaciones en cada punto del espacio discretizado, como podría ser a través del método de elementos o volúmenes finitos, a través del método de singularidades se calcula la velocidad en el campo fluido a través de la integración o sumatorio de la contribución de éstas, como se muestra en la figura inferior. Se emplean manantiales, sumideros, torbellinos, que pueden estar localizados de forma puntual, o extendidos en líneas o planos. Esta metodología es la que se utiliza habitualmente en la elaboración de los conocidos Métodos de Paneles o entramados de torbellinos (*Vortex Lattice*)



**Método de malla discretizada frente a método de singularidades.**

Imagen modificada de "*Flight Vehicle Aerodynamics*" (MIT Press, 2014)

## Óvalo de Rankine

El óvalo de Rankine es un modelo matemático que representa un sistema de flujo en dos dimensiones. Se compone de dos singularidades, un manantial y un sumidero, alineados con la corriente incidente. La localización de los puntos de remanso y la forma del óvalo están determinados por la relación entre la intensidad de las singularidades y la corriente incidente, pudiendo estudiarse como una relación de gastos, como se ha visto en las clases de problemas.

## Código Python para el cálculo del flujo potencial asociado a el óvalo de Rankine

En los siguientes párrafos se presenta una propuesta de cálculo numérico para el modelado discreto del flujo potencial. Existen muchas formas de realizar este cálculo, en este texto se presenta una alternativa sencilla que hace uso del módulo *NumPy* de Python, así como una representación gráfica de la función de corriente y el campo de velocidades.

Nótese que en lo que corresponde al texto escrito en este documento, se usan variables en **negrita** para denotar vectores o matrices.

Junto a este documento se dispone de códigos MatLab o Python para su ejecución independiente, así como del Notebook original.

Del mismo modo, se puede ejecutar de forma online a través del siguiente enlace: [!\[\]\(d3102649f02e825ddb76dc3de0190154\_img.jpg\) launch !\[\]\(55ca3a38dbb940110628e54e3ea7505d\_img.jpg\) binder](#)

```
In [ ]: # 1. Importación de módulos
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
```

```
In [ ]: # 2. Definición de las funciones de cálculo del campo de velocidades
def velocity_field(x, z, Q, x_s, z_s):
    """
    Calcula el campo de velocidades inducido por una serie finita de manantiales y sumideros.

    Parámetros:
    x, z -- coordenadas en el plano xz
    Q -- magnitud de los manantiales/sumideros
    x_s, z_s -- coordenadas de los manantiales/sumideros

    Devuelve:
    u, w -- componentes del campo de velocidades
    """
    u = np.zeros_like(x, dtype='f8')
    w = np.zeros_like(z, dtype='f8')
    N = len(Q)
    for i in range(N):
        r = np.sqrt((x - x_s[i])**2 + (z - z_s[i])**2)
        u += Q[i] * (x - x_s[i]) / (2 * np.pi * r**2)
        w += Q[i] * (z - z_s[i]) / (2 * np.pi * r**2)
    return u, w
```

La función `velocity_field` toma como entrada las coordenadas **x** e **z** en el plano xz, la magnitud **Q** de los manantiales y sumideros, y las coordenadas **x\_s**, **z\_s** de los manantiales y sumideros. La función devuelve las componentes **u** y **w** del campo de velocidades.

Para calcular las componentes del campo de velocidades, se utiliza un ciclo for que recorre cada uno de los manantiales y sumideros y calcula la contribución de cada uno de ellos en las componentes **u** y **w** utilizando la formulación matemática correspondiente (que puede ser deducida analíticamente a partir de los contenidos teóricos de la asignatura).

Por ejemplo, la formulación matemática utilizada aquí para calcular la velocidad horizontal inducida por un manantial de intensidad  $Q$ , que se encuentra localizado en  $(x_Q, z_Q)$ , en un punto del espacio,  $p$ , situado a una distancia radial  $r$ , es la siguiente:

$$U = \frac{Q}{2\pi r} \cos \theta = \frac{Q}{2\pi r} \frac{(x_p - x_Q)}{r}$$

Podemos utilizar esta función para calcular el campo de velocidades en los distintos ejes del sistema de referencia empleado, tanto a nivel vectorial como a nivel puntual, y usar esos datos para visualizar la solución en un gráfico.

```
In [ ]: # Definimos la magnitud y ángulo de la velocidad incidente
U_inf = 2
alpha = 0

# Definimos las coordenadas de los manantiales y sumideros
Q = np.array([15, -15])
x_s = np.array([-5, 5])
z_s = np.array([0, 0])

# Se definen también sus coordenadas en el plano complejo, para el cálculo del potencial.
t_s = x_s + 1j*z_s
num_sources = len(Q)

# Definimos una malla para las coordenadas x e z
nx, nz = 40, 40
x = np.linspace(-10, 10, nx)
z = np.linspace(-10, 10, nz)
Xm, Zm = np.meshgrid(x, z)
```

```

Tm = Xm + 1j*Zm

# Calculamos las componentes u y v del campo de velocidades inducido por los manantiales o sum
u, w = velocity_field(Xm, Zm, Q, x_s, z_s)

# Añadimos el componente de la velocidad incidente
u += U_inf * np.cos(alpha*np.pi/180)
w += U_inf * np.sin(alpha*np.pi/180)

# Construimos la función potencial
pot = 0
for s in range(num_sources):
    pot += Q[s]*np.log(Tm - t_s[s]) / (2*np.pi)
pot += U_inf * np.exp(-1j*alpha*np.pi/180) * Tm

# De la función potencial extraemos tanto el potencial de velocidades (Phi) como la función de
phi = pot.real
psi = pot.imag

```

Una vez calculados el campo de velocidades, se calcula la función potencial de la forma habitual. Se recuerda que  $f(t) = \Phi - i\Psi$ , por lo que tras construir  $f(t)$  es sencillo extraer tanto el potencial de velocidades,  $\Phi$ , como la función de corriente,  $\Psi$ .

Tras la fase de cálculo, es conveniente representar gráficamente la solución. Para ello hacemos uso de las funciones *contourf* y *contour* de Matplotlib, para representar el campo de la Función de Corriente ( $\Psi$ ), y la línea de corriente correspondiente a  $\Psi = 0$ .

Por último, empleamos la función *quiver* para representar el campo de velocidades vectorial de forma más visual. Por defecto, esta función representa la longitud de los vectores de forma inversa a su magnitud, por lo que aquí se realiza una normalización previa. Aunque sabemos que  $\dot{f}(t) = \Phi_x - i\Phi_z = U - iW$ , disponemos del campo de velocidades calculado previamente, en las variables **u** y **w**.

```

In [ ]: # Visualizamos la función de corriente y la función potencial
fig, axes = plt.subplots(1, 2, figsize=(10,8)) # Creamos la figura
ax1, ax2 = axes.flatten()

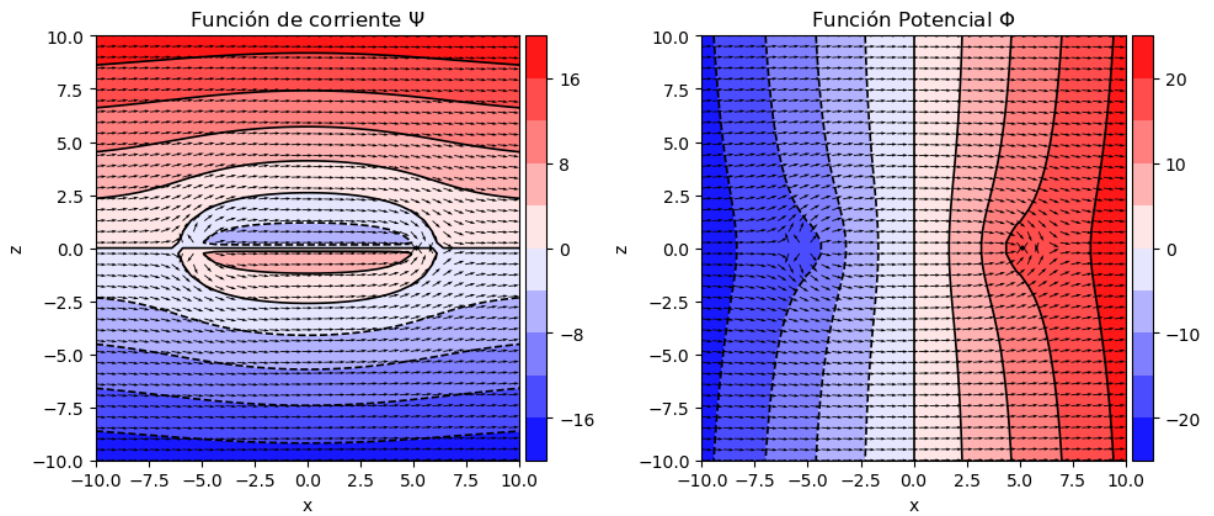
dividerPSI = make_axes_locatable(ax1) # Ajustamos la posición
caxPSI = dividerPSI.append_axes('right', size='5%', pad=0.05)
cont = ax1.contourf(Xm, Zm, psi, 10, cmap='bwr') # Pintamos los contornos
ax1.contour(Xm, Zm, psi, levels=10, colors=['black'], linewidths=1.2) # Representamos la línea de corriente
fig.colorbar(cont, cax=caxPSI, orientation='vertical') # Pintamos la barra de color
ax1.set_title(r'Función de corriente $\Psi$')

dividerPHI = make_axes_locatable(ax2) # Ajustamos la posición
caxPHI = dividerPHI.append_axes('right', size='5%', pad=0.05)
cont = ax2.contourf(Xm, Zm, phi, 10, cmap='bwr') # Pintamos los contornos
ax2.contour(Xm, Zm, phi, levels=10, colors=['black'], linewidths=1.2) # Representamos la línea de corriente
fig.colorbar(cont, cax=caxPHI, orientation='vertical') # Pintamos la barra de color
ax2.set_title(r'Función Potencial $\Phi$')

N = np.sqrt(u**2 + w**2)
ax1.quiver(Xm, Zm, u/N, w/N, scale=40) # Representamos el campo de velocidades
ax2.quiver(Xm, Zm, u/N, w/N, scale=40)

for ax in axes:
    ax.set_xlabel('x') # Etiquetamos de forma
    ax.set_ylabel('z')
    ax.set_aspect('equal', 'box')
plt.tight_layout()

```



En este caso, la visualización vectorial (a través de la función *quiver*) puede confundir al observador, al parecer que hay vectores que "atravesen la línea de corriente dibujada que conforma el óvalo. Téngase en cuenta que los vectores dibujados dependen de la discretización espacial utilizada, y su longitud está condicionada por el parámetro *scale* utilizado en la llamada a la función. La velocidad es **siempre** tangente a las líneas de corriente, y éstas **nunca** se pueden cruzar.

De forma similar al proceso antes realizado, podríamos representar el campo de velocidades tanto horizontal como vertical:

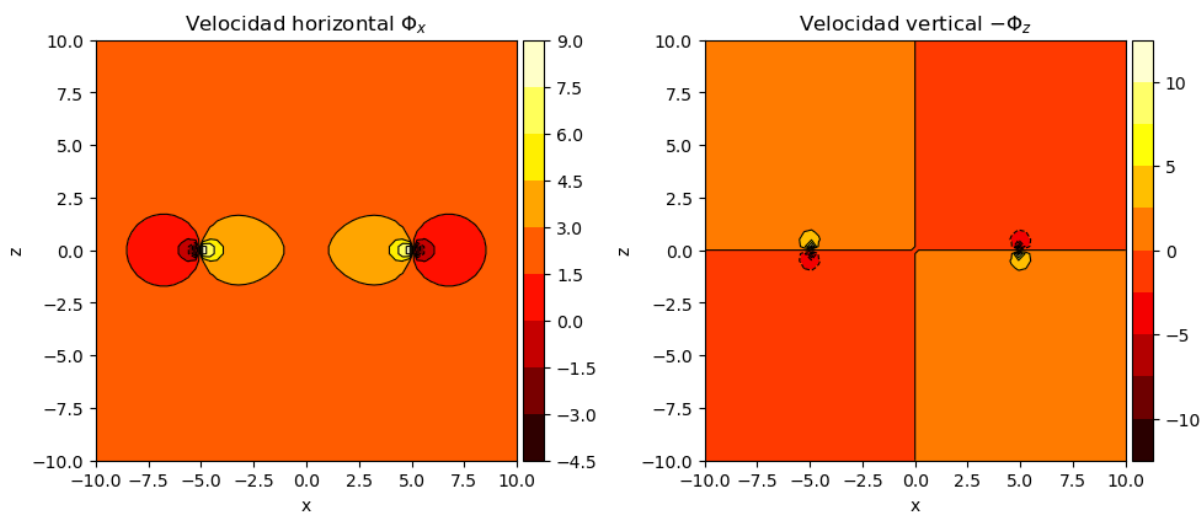
```
In [ ]: # Visualizamos el campo de velocidades y la función de corriente
fig, axes = plt.subplots(1,2, figsize=(10,8))
ax1, ax2 = axes.flatten()

dividerU = make_axes_locatable(ax1)
caxU = dividerU.append_axes('right', size='5%', pad=0.05)
dividerV = make_axes_locatable(ax2)
caxV = dividerV.append_axes('right', size='5%', pad=0.05)

ax1.set_title(r'Velocidad horizontal $\Phi_x$')
cont = ax1.contourf(Xm, Zm, u, 10, cmap='hot')
ax1.contour(Xm, Zm, u, levels=10, colors=['black'], linewidths=0.8)
fig.colorbar(cont, cax=caxU, orientation='vertical')

ax2.set_title(r'Velocidad vertical $-\Phi_z$')
cont = ax2.contourf(Xm, Zm, w, 10, cmap='hot')
ax2.contour(Xm, Zm, w, levels=10, colors=['black'], linewidths=0.8)
fig.colorbar(cont, cax=caxV, orientation='vertical')

for ax in axes:
    ax.set_xlabel('x')
    ax.set_ylabel('z')
    #ax.set(xlim=(-3, -1), ylim=(-1, 1))
    ax.set_aspect('equal', 'box')
plt.tight_layout()
plt.show()
```



In [ ]: