Nombre: David Urbaez León

Informe Trabajo Mecánica de Fluidos en Python.

Se desea programar un software en el lenguaje "Python" con el objetivo de mostrar las líneas de corriente y líneas equipotenciales para un campo de velocidad. Dada la alta versatilidad del lenguaje Python se propone realizar el diseño de un programa que permita graficar los campos anteriormente mencionados dada una función de corriente arbitraria o una función potencial de velocidad arbitraria.

Para cumplir el objetivo propuesto se hace uso de la librería sympy, para la realización de los cálculos.

Inicialmente se calcula el campo velocidad con las ecuaciones:

$$V = \vec{\nabla}\phi = \begin{pmatrix} \frac{d\phi}{dx} \\ \frac{d\phi}{dy} \end{pmatrix} = \begin{pmatrix} \frac{d\psi}{dy} \\ -\frac{d\psi}{dx} \end{pmatrix}$$

Posteriormente se procede a graficar las líneas de corriente con la función "Streamplot" ofrecida por la librería MatplotLib. En el caso de iniciar desde una función potencial de velocidad se pueden graficar las líneas equipotenciales (LEP) con la función "Contour" ofrecida por la misma librería.

Ejemplos:

1) Ovalos de Rankine.

Se consideran las siguientes ecuaciones para un fluido incompresible:

Fuente

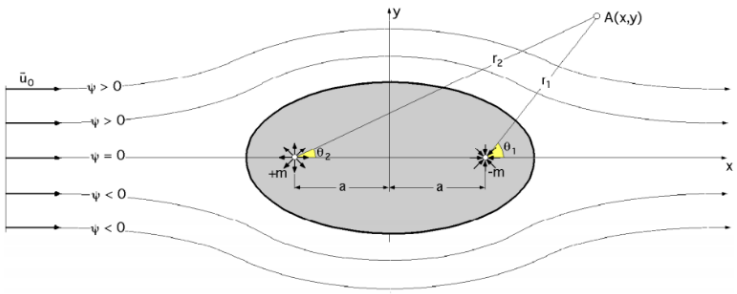
$$\psi = \frac{Q}{2*\pi} * \theta \quad ; \quad \varphi = \frac{Q}{2*\pi} * \ln r$$

Sumidero

$$\psi = -\frac{Q}{2*\pi} * \ln r \quad ; \quad \varphi = -\frac{Q}{2*\pi} * \theta$$

Flujo Uniforme

$$\psi = U_o * y \quad ; \quad \varphi = U_o * x$$



Cuando los 3 tipos de campos se superponen, se está ante un ovalo de Rankine, se considerará la función potencial de velocidad para el correspondiente gráfico.

Sumando los 3 campos escalares:

$$\phi = u0x - \frac{Q}{2*\pi}\ln(r1) + \frac{Q}{2*\pi}\ln(r2) = u0x + \frac{Q}{2*\pi}\ln\left(\frac{r2}{r1}\right)$$

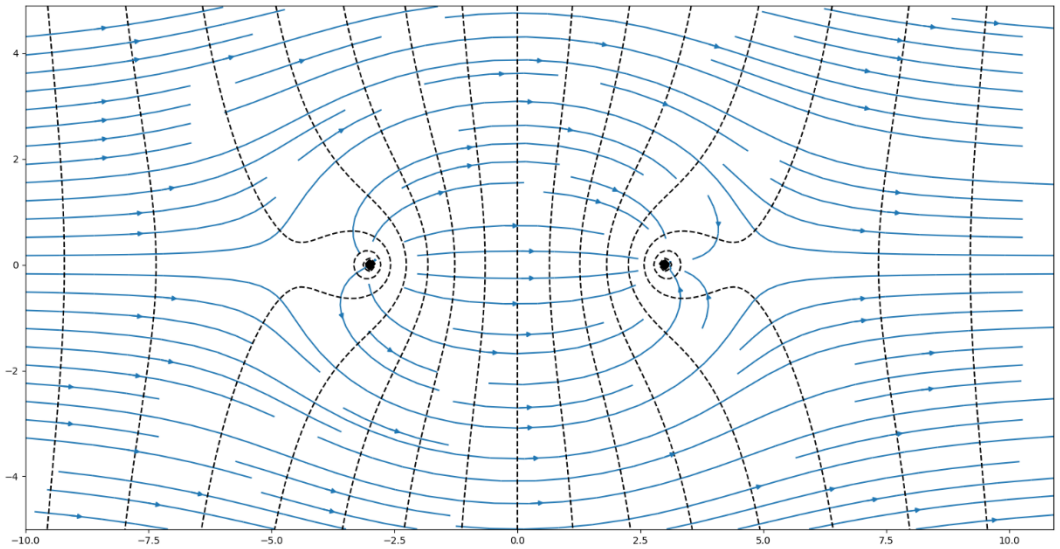
Con $r1^2 = (x-a)^2 + y^2$ y $r2^2 = (x+a)^2 + y^2$

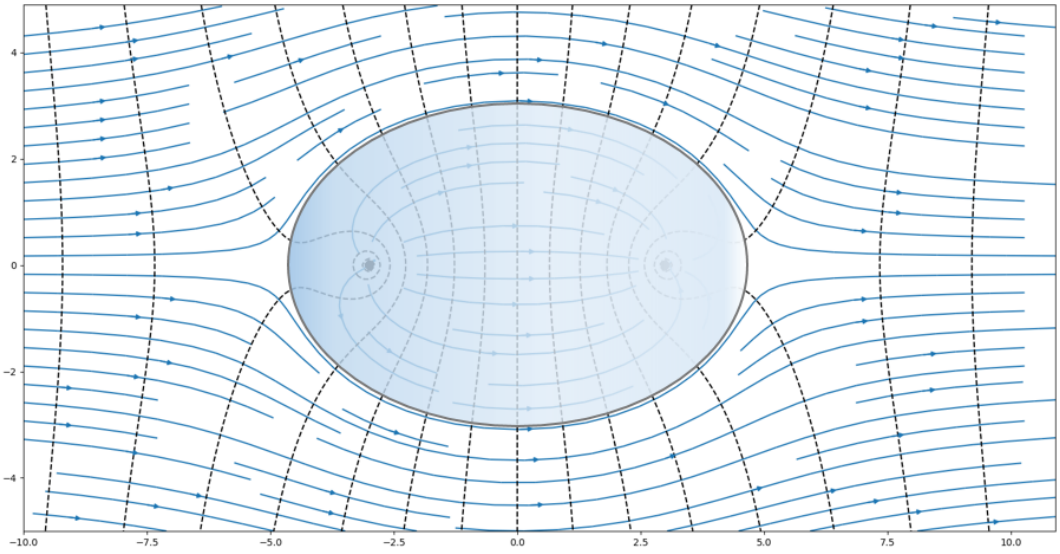
Reemplazando las ecuaciones:

$$\phi = u0x + \frac{Q}{4 * \pi} \ln\left(\frac{(x+a)^2 + y^2}{(x-a)^2 + y^2}\right)$$

Se tomará: $u0 = 1\left[\frac{m}{s}\right], Q = 6\left[\frac{m^3}{s}\right], a = 3[m]$

Y se obtiene:





Se observa claramente que las líneas de corriente (en azul) son perpendiculares a las líneas equipotenciales (en negro).

2) Modificación Óvalos de Rankine.

- En el caso anterior se tenía una fuente en las coordenadas $(0, -a)$ y un sumidero en la coordenada $(0, +a)$. La modificación planteada consiste en:

Fuentes:

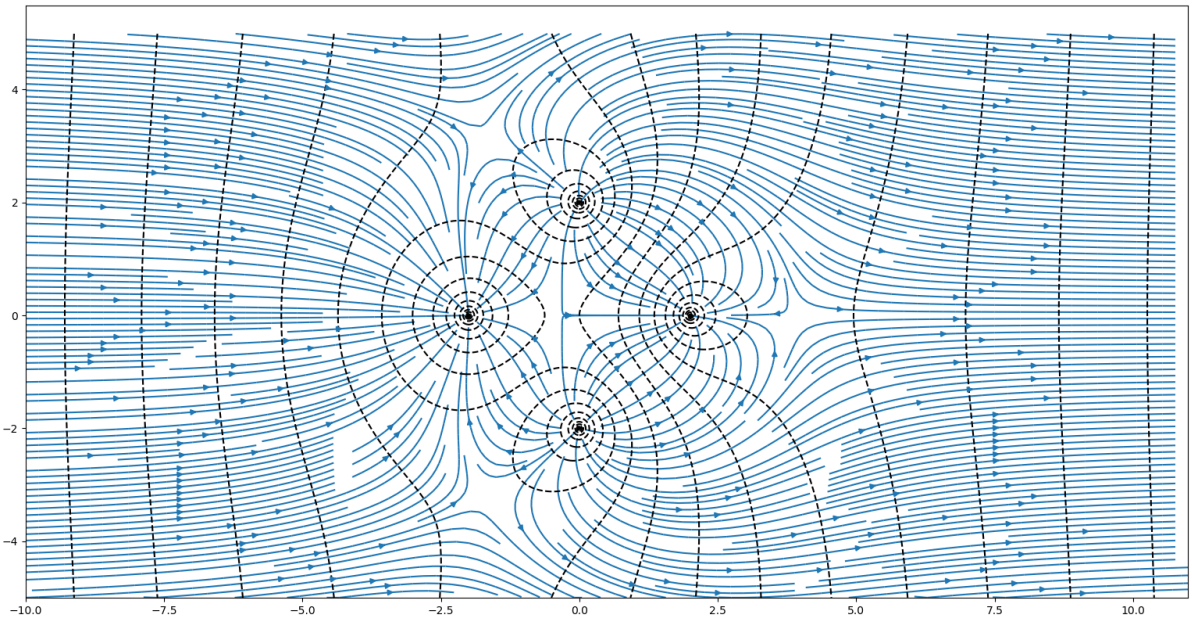
- $(0, +a)$
- $(0, -a)$

Sumideros:

- $(-a, 0)$
- $(+a, 0)$

Además, $u0 = 0.2\left[\frac{m}{s}\right], Q = 4\ \left[\frac{m^3}{s}\right], a = 2\ [m]$

Y se obtuvo:



Al igual que en el ejemplo 1, se observa que las líneas de corriente (en azul), son perpendiculares a las líneas equipotenciales (en negro), además se observa que las líneas de corriente entran en los puntos ubicados sobre el eje x $(-a, 0)\ y\ (+a, 0)$, mostrando así su comportamiento como sumideros, mientras que en los puntos ubicados sobre el eje Y $(0, -a)\ y\ (0, +a)$ las líneas de corriente salen, mostrando su comportamiento como fuentes.

Bibliografía:

Rock it, Science! (2016). *Python Flow Field Modeling: Example 1*. Disponible en: https://gist.github.com/anonymous/33ebeedc4bf7b67a0aad27a88282fbb5

Fernandez,P (2011). Flujo Incompresible No Viscoso. Disponible en: http://files.pfernandezdiez.es/MecanicaFluidos/PDFs/04MecFluidos.pdf

Anexo: Código realizado en Python.

```
from sympy import *
from sympy.physics.vector import *
import numpy as np
import matplotlib.pyplot as plt

R = ReferenceFrame('R')
x, y = R[0], R[1]

X, Y = np.meshgrid(np.arange(-10, 11, 0.1), np.arange(-5, 5, 0.1))

vector_components = lambda vectorField: (vectorField.dot(R.x),
vectorField.dot(R.y))

integrateGradient = lambda gradientField:
integrate(gradientField.dot(R.x), x) +
integrate(gradientField.dot(R.y), y).subs(x, 0)

discretize_field = lambda field: lambdify((x, y), field)(X, Y)


def plot_vector_field(vectorField):
    data = discretize_field(vector_components(vectorField))
    plt.figure()
    # plt.quiver(X, Y, data[0], data[1])
    plt.streamplot(X, Y, data[0], data[1], density=2.5)


def plot_contour(scalarField):
    data = discretize_field(scalarField)
    plt.contour(X, Y, data, 100, colors='k', linestyles='dashed')


u0, Q, a = 1, 4, 2
# phi = u0 * x - Q / (2 * np.pi) * ln(((x - a) ** 2 + y ** 2) /
((x + a) ** 2 + y ** 2))  # Velocity potential
r1 = ((x + a) ** 2 + y ** 2) ** (1 / 2)
r2 = ((x - a) ** 2 + y ** 2) ** (1 / 2)
r3 = ((x) ** 2 + (y - a) ** 2) ** (1 / 2)
r4 = ((x) ** 2 + (y + a) ** 2) ** (1 / 2)

phi = u0 * x - Q / (2 * np.pi) * ln(r1) - Q / (2 * np.pi) * ln(r2)
+ Q / (2 * np.pi) * ln(r3) + Q / (2 * np.pi) * ln(r4)
velocity = gradient(phi, R)
plot_vector_field(velocity)
plot_contour(phi)
plt.show()
```

```
from sympy import *
from sympy.physics.vector import *
import numpy as np
import matplotlib.pyplot as plt

R = ReferenceFrame('R')
x, y = R[0], R[1]

X, Y = np.meshgrid(np.arange(-10, 11, 0.01), np.arange(-5, 5, 0.01))

vector_components = lambda vectorField: (vectorField.dot(R.x),
vectorField.dot(R.y))

integrateGradient = lambda gradientField:
integrate(gradientField.dot(R.x), x) + integrate(gradientField.dot(R.y),
y).subs(x, 0)

discretize_field = lambda field: lambdify((x, y), field)(X, Y)


def plot_vector_field(vectorField):
    data = discretize_field(vector_components(vectorField))
    plt.figure()
    # plt.quiver(X, Y, data[0], data[1])
    plt.streamplot(X, Y, data[0], data[1], density=3)


def plot_contour(scalarField):
    data = discretize_field(scalarField)
    plt.contour(X, Y, data, 150, colors='k', linestyles='dashed')


u0, Q, a = 0.2, 4, 2
# phi = u0 * x - Q / (2 * np.pi) * ln(((x - a) ** 2 + y ** 2) / ((x + a)
** 2 + y ** 2))  # Velocity potential
r1 = ((x + a) ** 2 + y ** 2) ** (1 / 2)
r2 = ((x - a) ** 2 + y ** 2) ** (1 / 2)
r3 = ((x) ** 2 + (y - a) ** 2) ** (1 / 2)
r4 = ((x) ** 2 + (y + a) ** 2) ** (1 / 2)

phi = u0 * x - Q / (2 * np.pi) * ln(r1) - Q / (2 * np.pi) * ln(r2) + Q /
(2 * np.pi) * ln(r3) + Q / (2 * np.pi) * ln(r4)
velocity = gradient(phi, R)
plot_vector_field(velocity)
plot_contour(phi)
```