Universitat Politècnica de Catalunya

ESCUELA SUPERIOR DE INGENIERÍA INDUSTRIAL, AEROESPACIAL Y AUDIOVISUAL DE TERRASSA (ESEIAAT)

ASSIGNMENT 1: NON-VISCOUS POTENTIAL FLOW

Computational Engineering - Space and Aeronautical Engineering MSc

Author:

Jorge SIMÓN AZNAR

Contents

1 Abstract

The problem consists in the study of the behaviour of potential flow in three different cases.

Firstly, flow in a channel of H x L.

Secondly, the same flow along a static cylinder. In this case we will assume that the flow is incompressible ($M < 0.2$).

Finally, we will add rotation to the cylinder to study the variation of the parameters (lift, drag, circulation...).

2 Methodology

2.1 Theory

As we have mentioned before, this type of flow is called potential flow and it is considered irrotational and 2D.

Then, the stream function of this flow verifies the mass conservation equation:

$$\nabla \cdot (\rho \mathbf{v}) = 0 \tag{2.1}$$

Furthermore, for this problem we will consider that the velocity is based on the stream function as follows:

$$v_x = \frac{\rho_{ref}}{\rho}\frac{\partial \psi}{\partial y} \qquad v_y = -\frac{\rho_{ref}}{\rho}\frac{\partial \psi}{\partial x} \tag{2.2}$$

Then, assuming irrotational flow, we can obtain the stream function equation:

$$\frac{\partial}{\partial x}\left(\frac{\rho_{ref}}{\rho}\frac{\partial \psi}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{\rho_{ref}}{\rho}\frac{\partial \psi}{\partial y}\right) = 0 \tag{2.3}$$

Now, considering the Stoke's Theorem

$$\oint_C \mathbf{v} \cdot d\mathbf{r} = \iint_S (\nabla \times \mathbf{v}) \cdot d\mathbf{S} \tag{2.4}$$

So, due to the fact that the flow is irrotational

$$\Gamma = \oint_C \mathbf{v} \cdot d\mathbf{r} = 0 \tag{2.5}$$

If we approximate this integrals to second order we obtain the following expression:

$$\Gamma \approx v_{ye}\Delta y_P - v_{xn}\Delta x_P - v_{yw}\Delta y_P + v_{xs}\Delta x_P \tag{2.6}$$

Using equations 2.2 and **??**

$$\frac{\rho_{ref}}{\rho}\frac{\psi_E - \psi_P}{d_{PE}}\Delta y_P - \frac{\rho_{ref}}{\rho}\frac{\psi_N - \psi_P}{d_{PN}}\Delta x_P + \frac{\rho_{ref}}{\rho}\frac{\psi_P - \psi_W}{d_{PW}}\Delta y_P + \frac{\rho_{ref}}{\rho}\frac{\psi_P - \psi_S}{d_{PS}}\Delta x_P = 0 \tag{2.7}$$

In this way we obtain the discretization equation, which we will use to compute and to obtain the numerical solution:

$$a_P\psi_P = a_E\psi_E + a_N\psi_N + a_W\psi_W + a_S\psi_S + b_P \tag{2.8}$$

Although we have obtained the discretization equation for the flow, it is neccessary to establish boundary conditions since they have special properties.

At the top and bottom of the channel we apply the Neumann boundary condition (BC). It sais that the normal velocity to them has to be zero. In addition, the value of the stream function is known so we will have the inlet conditions at the same height.

For the inlet, the Dirichlet conditions are imposed since the stream function's value is known ($\psi_{in} = v_{in} \cdot y$).

The last condition is at the outlet where the velocity is supposed to be horizontal since parallel flow is assumed. Then, the gradient at normal direction is zero and the stream function is the same as the node of its left.

2.2 Blocking-off method

Once we have the discretization equations, we have to distinguish between fluid region and solid region in the case of the cylinder. For that, we will use the blocking-off method. The domain is discretized in such a way that each control volume (CV) belongs to the fluid region or to the solid region according to the following criteria

$$D = \sqrt{(x_P - x_0)^2 + (y_P - x_0)^2} \tag{2.9}$$

So, if $D \leq R$ then $\psi_P = \frac{\psi_{top}+\psi_{bottom}}{2}$, but if $D > R$ then ψ_P will be the suitable value of the flow. Where D is the distance between the node and the center of the cylinder and R is the radius of the cylinder.

2.3 Gauss-Seidel

The solver we have used is the Gauss-Seidel method. It is an iterative technique used to solve a system of linear equations. It starts with an initial guess (ψ_{ini}) for the solution and then updates each variable one at a time based on the latest values. This process is repeated until the solution converges to a desired accuracy, in our case it will be $\delta < 10^{-6}$.

3 Code Structure

To create the agorthm I have used the following structure:

- 1. Input Data :

  Firstly, variables such as length (L) and height (H) of the channel, thermodinamic parameters (V_{in}, T_{in}, P_{in}, ρ_{in}, γ), accuracy (δ), cyinder structure and stream function are defined.

- 2. Previous Calculations:

  Secondly, the mesh of the problem is generated (NxM) and the blocking-off method is applied.

- 3. Initial Values Estimation:

  Estimation of the initial values (already defined) of all nodes.

- 4. Gauss-Seidel method:

  Firstly, it calculates the discretization coefficients and the stream function, starting with the boundary (top, bottom, inlet and outlet) and after that it calculates the coefficients and the stream function of the internal nodes. It repeats the process iteratively until the average of the matrix of the stream function minus the average of the matrix of the stream function of the previous step is less than δ.

- 5. Velocities Calculation

  Once we have achieved the accuracy that we want, we calculate the velocities of the flow at all the nodes by using expressions 2.6 and 2.8. Then, the velocities at the main nodes will be:

$$v_{xP} = \frac{v_{xn} + v_{xs}}{2} \qquad v_{yP} = \frac{v_{ye} + v_{yw}}{2} \tag{3.1}$$

$$v_p{}^2 = v_{xP}^2 + v_{yP}^2 \tag{3.2}$$

- 6. Thermodynamic Parameters

  Apart from considering the flow uncompressible and 2D, we also consider that it is isentropic, so the entropy remains constant and the total energy is conserved. Therefore, we can obtain the following expression for the temperature

$$h_{ref} + e_{kref} = h_P + e_{kP} \tag{3.3}$$

  Considering

$$h_P - h_{ref} = \bar{c}_P (T_P - T_{ref}) \tag{3.4}$$

  Then,

$$T_P = T_{ref} + \frac{(v_{ref}^2 - V_P)}{2\bar{c}_P} \tag{3.5}$$

  Where

$$\bar{c}_P(T) = 1034.09 - 2.849 \times 10^{-1} \cdot T + 7.817 \times 10^{-4} \cdot T^2 - 4.971 \times 10^{-7} \cdot T^3 \tag{3.6}$$

  Now, we can also calculate the pressure at the nodes as

$$P_P = P_{ref} \left(\frac{T_P}{T_{ref}} \right)^{\frac{\gamma}{\gamma - 1}} \tag{3.7}$$

- 7. Final Calculations:

  Finally, once we have calculated all the thermodynamic parameters we can obtain lift and drag forces and pressure coefficient to complete the problem by using the following expressions:

$$D = \sum_{i}^{N_{cyl}} (P_i \cdot S_y)_w - (P_i \cdot S_y)_e \tag{3.8}$$

$$L = \sum_{i}^{M_{cyl}} (P_i \cdot S_x)_s - (P_i \cdot S_x)_n \tag{3.9}$$

4 Code Verification

In order to verify that our code works correctly, we have to ensure that te code is free of errors, so we are going to carry out some tests.

4.1 Uniform Flow

The first test consists in dropping the cylinder by taking its diameter to zero. Then, the expected result is an uniform flow with parallel stream lines.

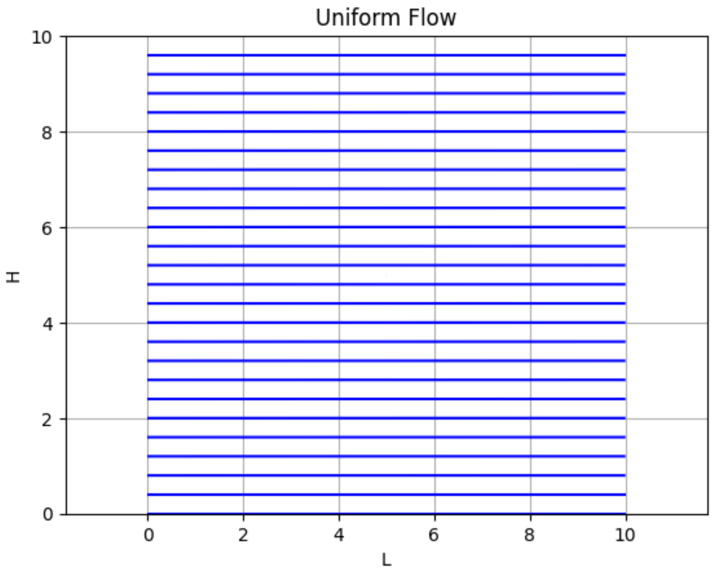
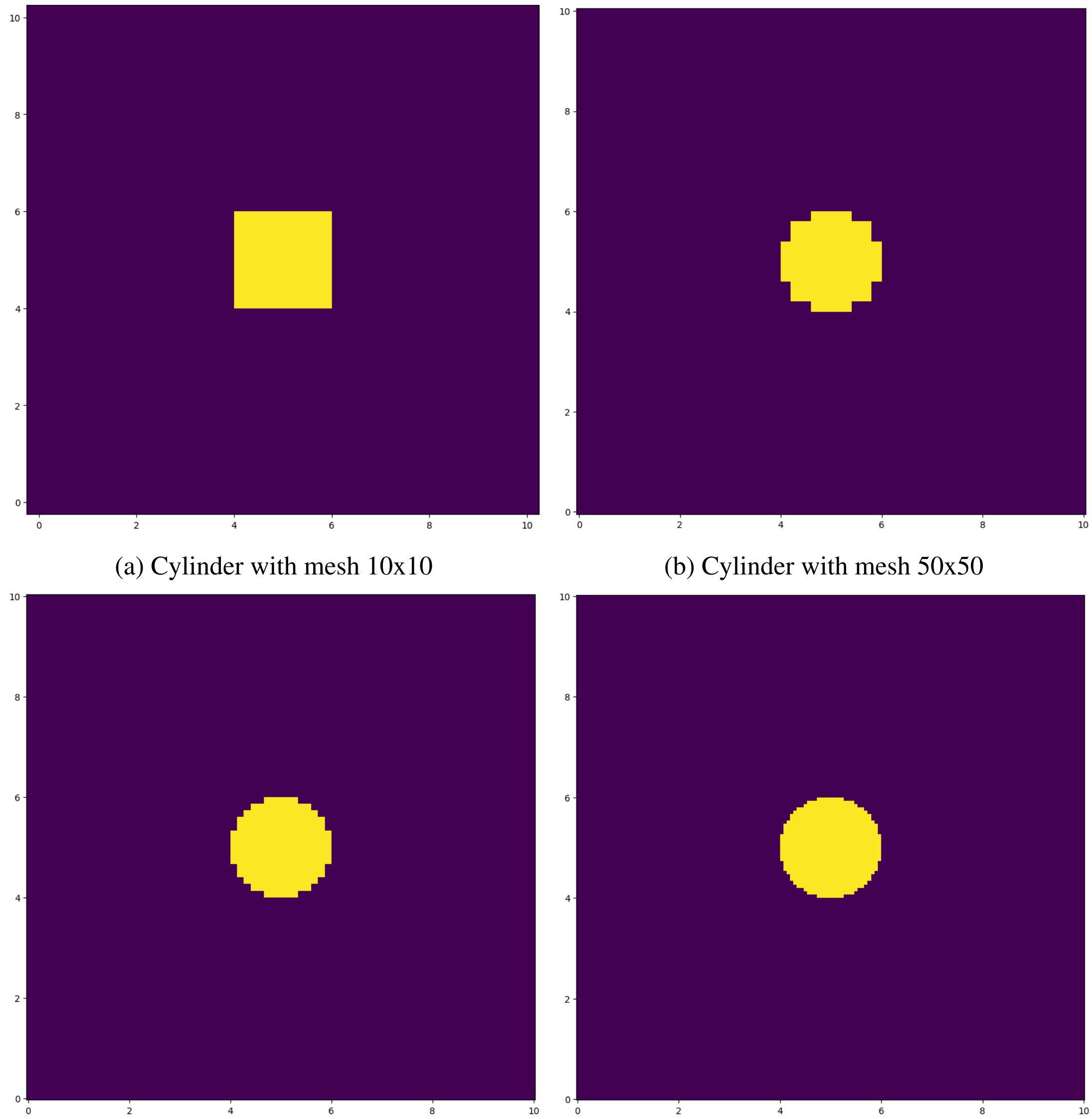


Figure 4.1: Uniform Flow

As we can see in the previous figure, the stream lines are completely parallel, giving rise to a continuous parallel flow without any type of disturbance, as expected, so the code for this case is correct.

4.2 Blocking-off method

In this second test we are going to test, not only whether the code works properly or not, but also if the dimension of the mesh is enough for the problem.



(a) Cylinder with mesh 10x10

(b) Cylinder with mesh 50x50

(c) Cylinder with mesh 75x75

(d) Cylinder with mesh 150x150

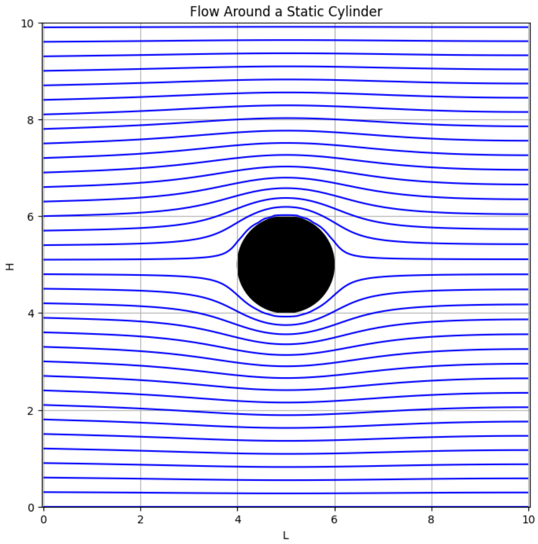
Figure 4.2: Comparison of different mesh

As we can observe, the greater dimension of the mesh the better approximation of the cylinder. So, the objective is to use an appropiate dimension that don't requires much computational time but has enough accuracy by plotting the cylinder.

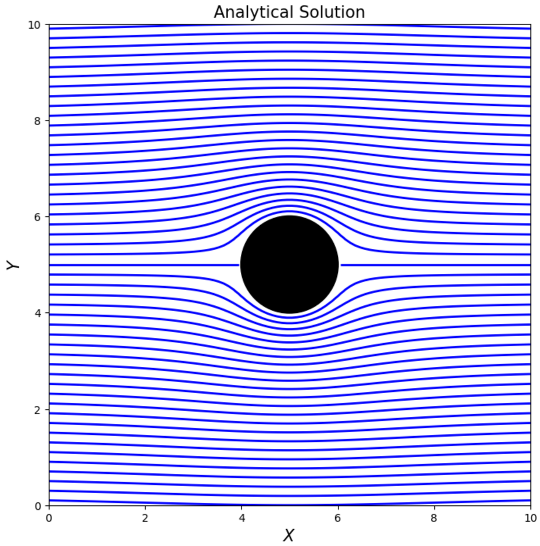
Then, we consider that $(N,M) = (150,150)$ will be enough for the problem. Fuerthermore, when we compute the velocity at top and bottom of the cylinder we obtain 3.6 m/s, it only differs in 10% from the theoretical value, which would be 4 m/s. So, taking into account that the computational time for 150x150 is 80 minutes, we will consider this result as valid.

4.3 Analytical Solution

Finally, we are going to compare our results with the analytical solution of the problem of non-rotating cylinder.



(a) Numerical solution



(b) Analytical solution

As we can see, the two figures are extremely similar, and the symmetry of the problem can be appreciated in both, so we consider the size of the mesh enough for our solver.

5 Physics of the problem

In this section we are going to analyze the results and the physics behind them.

Firstly, before starting to analyze the results, it is convenient to show the initial data, the input of the problem:

Input	**Value**
Length	10 m
Heigth	10 m
Initial velocity	2 m/s
Initial temperature	293 K
Initial pressure	101325 Pa
Initial density	1.204 kg/m^3
Cylinder diameter	2 m
Convergence (δ)	1×10^{-6}

Table 5.1: Input parameters

5.1 Non-rotating cylinder

The first case that we are going to analyze is the non-rotating cylinder. In this problem we have an important symmetry around the object, then we expect that the velocity, temperature and pressure fields to be symmetric too.

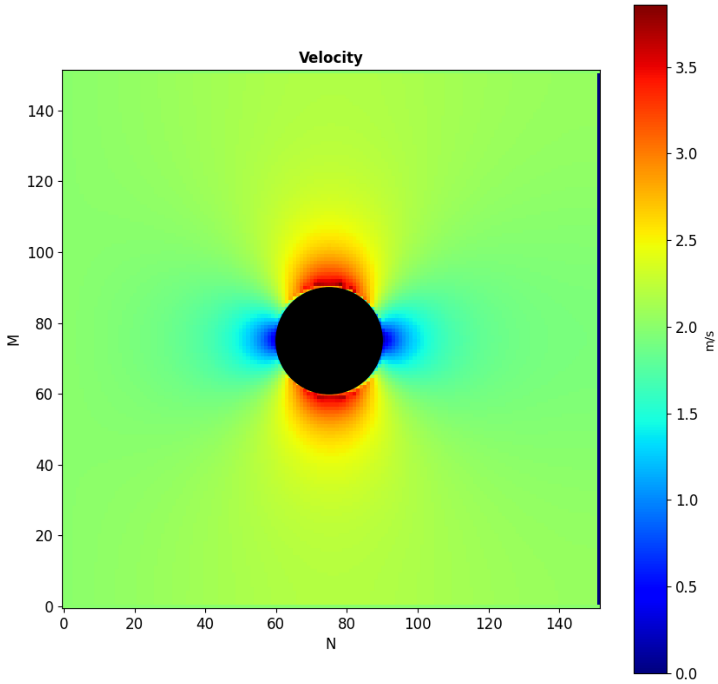
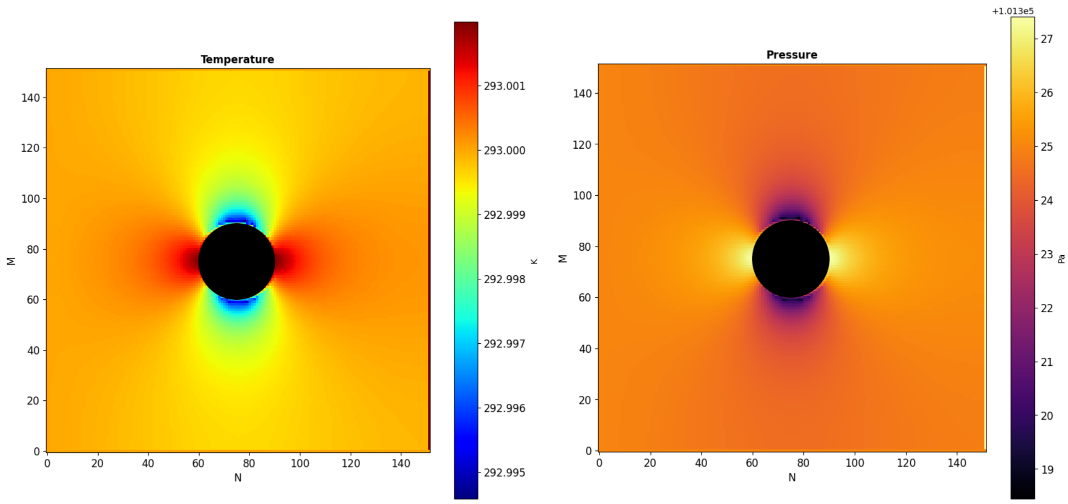


Figure 5.1: Map of velocity gradient

As we can see, the velocity is symmetric respect to the cylinder, reaching the maximum values at the top and the bottom of it as we expected. We also observe at the lateral points that the velocity dicreases to zero, and that would explain the shape of the streamlines around the body of the object.

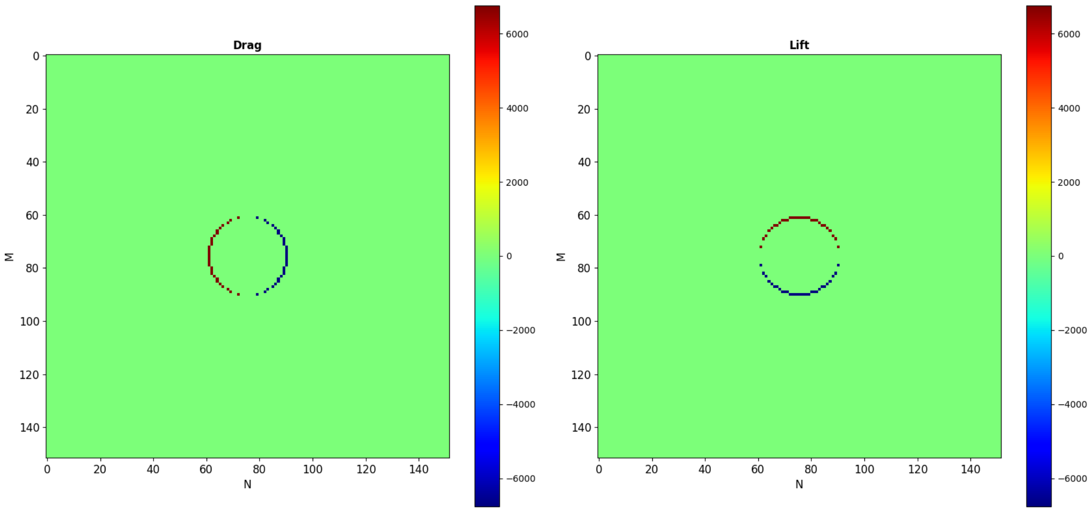


(a) Temperature (b) Pressure

Figure 5.2: Maps of Temperature and Pressure gradients

The same occurs to the temperature and the pressure, they are symmetric, but in this case, both reach the maximum at the lateral points.

Finally, we are going to compute the values of drag and lift and we expect to obtain zero in both due to the fact that the cylinder is static and the pressure has a completely symmetric distribution.



(a) Drag (b) Lift

Figure 5.3: Drag and Lift

We can perfectly observe that in both cases, drag and lift, the distribution is completely symmetrical, giving rise to values of

$$D = -0.024\,N/m \qquad L = -0.033\,N/m$$

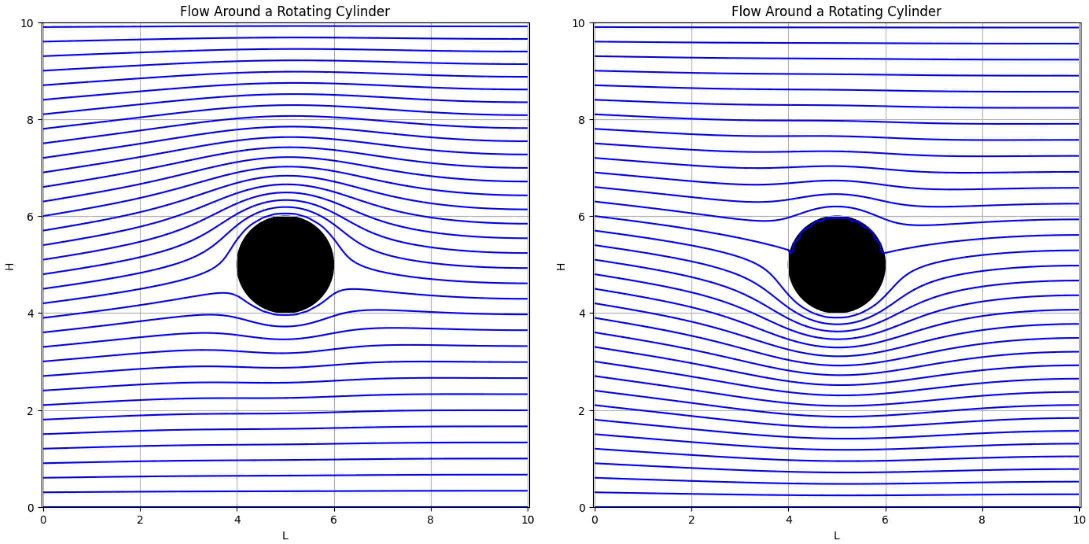
These values are not exactly zero since the solver does not give us the theoretical values but the approximate in a great way.

Furthermore, due to the fact that the flow has not viscosity and there is simmetry respect to the vertical axis, it was expected that the drag would be zero, this phenomena is known as d'Alembert's paradox.

5.2 Rotating Cylinder

The last case is the rotating cylinder. In order to obtain it, we will change the stream function value into the cylinder. In our case, for the cylinder rotating clockwise we multiply the unrotated value by 0.8 and for counterclockwise by 1.2.

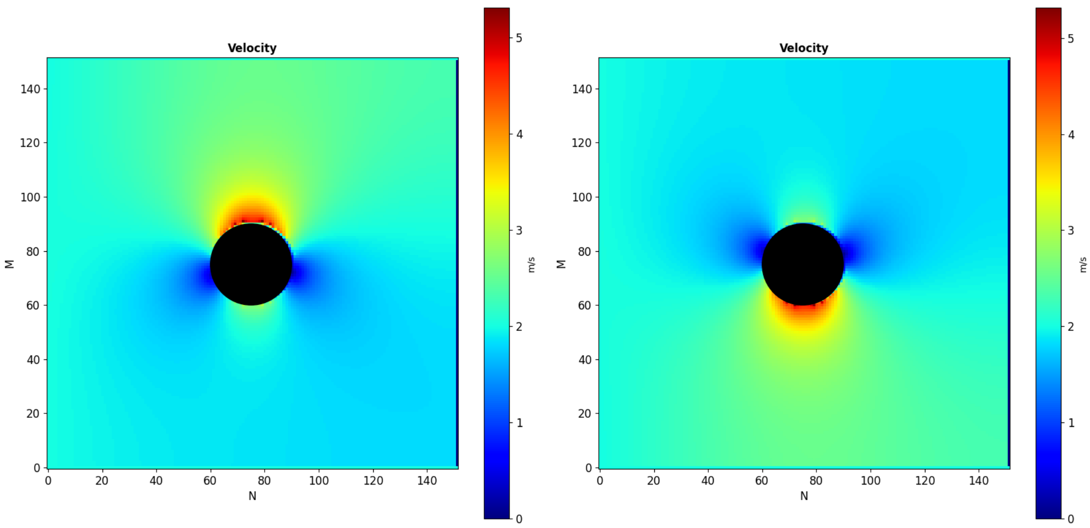
Then, we obtain the following results



(a) Clockwise (b) Counterclockwise

Figure 5.4: Potential flow around a rotating cylinder

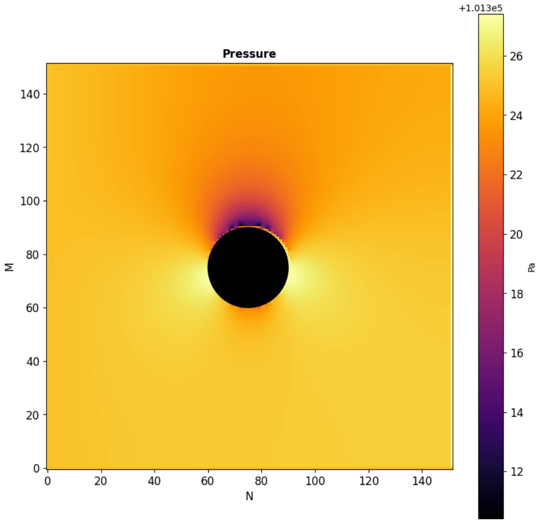
We can see how the movement of the flow has changed, producing an asymmetry with respect to the x-axis due to the rotation of the cylinder. In addition, it is observed how the stream lines are different depending on the direction of rotation.



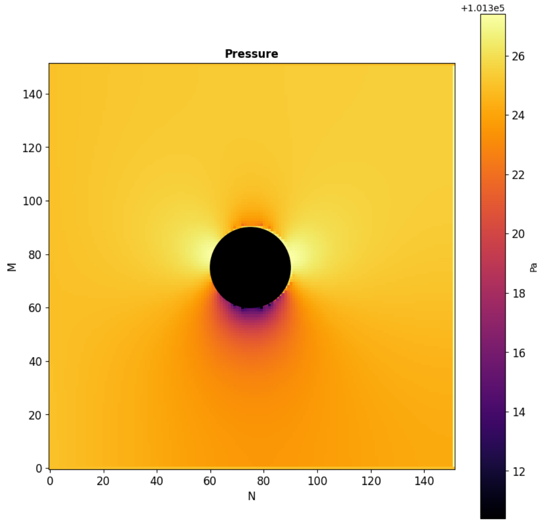
(a) Clockwise (b) Counterclockwise

Figure 5.5: Velocity gradient

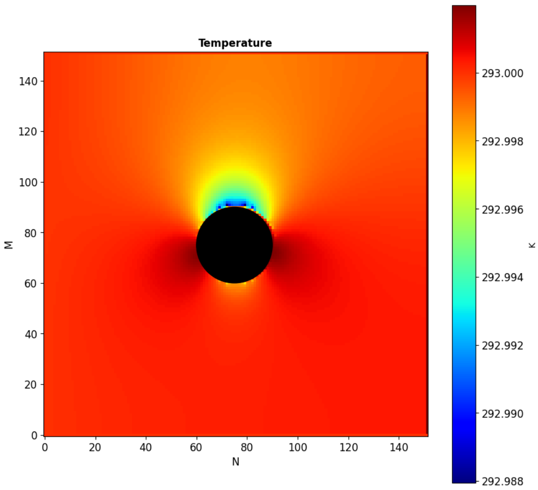
With respect to the velocity, we can see how it changes as the cylinder rotates. This is because, depending on the direction of rotation, the flow that goes in favor of this will be accelerated, while the one that goes against the rotation will be slowed down, which produces this variation in the velocity distribution.



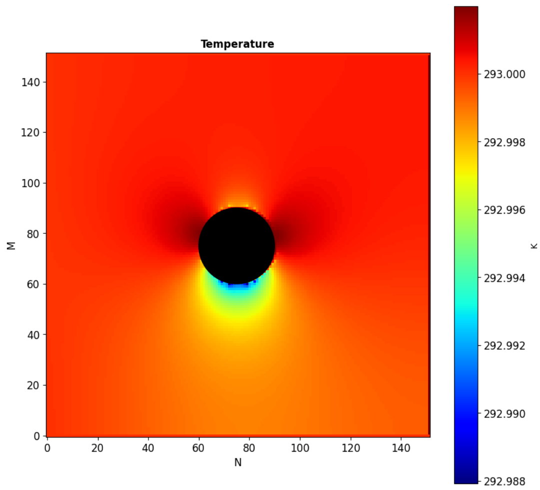
(a) P. Clockwise



(b) P. Counterclockwise



(c) T. Clockwise



(d) T. Counterclockwise

The same happens with the pressure and temperature around the cylinder, depending on the direction of rotation we will find areas with higher pressure and temperature above or below.

Finally, using the equation 2.6 we are able to compute the circulation in the cylinder and its contour, obtaining the following result:

$$\Gamma = -10.76\ m^2/s \qquad \Gamma = 10.82\ m^2/s$$

These are the circulation values for the cylinder rotating clockwise and counterclockwise respectively. We see how the sign of the circulation changes with the direction of rotation of the cylinder, indicating that the code is performing the calculations correctly.

Then, we can calculate the lift by using the Kutta-Joukowski theorem, in which

$$L = -\rho_\infty V_\infty \Gamma \tag{5.1}$$

Therefore, according to the theorem

$$L_{clock} = 25.91\ N/m \qquad L_{count} = -26.05\ N/m$$

And these values should be similar to those obtained using the pressures

$$L_{clock} = 14.94\ N/m \qquad L_{count} = -15.00\ N/m$$

We can appreciate certain differences between these values, this is possibly due to the accumulated error at the time of calculating the velocities and pressures, since the solver is only an approximation.

Finally, as in the previous case (non-rotating), due to the d'Alembert's paradox the drag should be zero

$$D_{clock} = -0.45\ N/m \qquad D_{count} = -0.43\ N/m$$

6 Code

```
import numpy as np
import matplotlib.pyplot as plt
import random

"""INPUT DATA"""

# Physical Data
L = 10.0
H = 10.0
V_in = 2
P_in = 101325
T_in = 293
gamma = 1.4
Rho_in = 1.204

# Numeral Data
N = 150
M = 150
delta = 0.000001
Psi_in = random.randint(1,10)
delta_N = L/N
delta_M = H/M

"""PREVIOUS CALCULATION"""

# Mesh Generation
X_cv = np.zeros(N+1)
Y_cv = np.zeros (M+1)

for i in range(1,N+1):
    X_cv[i] = X_cv[i-1] + delta_N

for j in range(1, M+1):
    Y_cv[j] = Y_cv[j-1] + delta_M

# Internal Nodes
X_p = np.zeros(N)
Y_p = np.zeros(M)

for i in range(N):
    X_p[i] = (X_cv[i+1] + X_cv[i])/2

for j in range(M):
    Y_p[j] = (Y_cv[j+1] + Y_cv[j])/2

# Mesh Generation
X_P = np.zeros(N+2)
Y_P = np.zeros (M+2)

X_P[0] = X_cv[0]
X_P[-1] = X_cv[-1]
Y_P[0] = Y_cv[0]
Y_P[-1] = Y_cv[-1]
for i in range (1, N+1):
    X_P[i] = X_p[i-1]
for j in range (1, M+1):
    Y_P[j] = Y_p[j-1]

# Cylinder
D = 2.0
x0 = L/2.0
```

```
y0 = H/2.0

# Define Metrics
Psi = np.zeros((M+2,N+2))
Psi_aux = np.zeros((M+2,N+2))
Rho = np.zeros((M+2,N+2))
a_E = np.zeros((M+2,N+2))
a_W = np.zeros((M+2,N+2))
a_S = np.zeros((M+2,N+2))
a_N = np.zeros((M+2,N+2))
b_P = np.zeros((M+2,N+2))
a_P = np.zeros((M+2,N+2))

"""INITIAL MAP"""

# Outlet nodes
for i in range(0,N+2):
    Psi[0][i] = Psi_in
    Psi[-1][i] = Psi_in
    Psi_aux[0][i] = Psi_in
    Psi_aux[-1][i] = Psi_in

# Inlet nodes
for i in range(0,N+2):
    for j in range(0,M+2):
        Psi[j][i] = Psi_in
        Psi_aux[j][i] = Psi_in
        Rho[j][i] = Rho_in

# Cylinder Body
I_cyl = np.zeros((M+2, N+2))
for i in range (N+2):
    for j in range (M+2):
        distance = ((X_P[i]-x0)**2 + (Y_P[j]-y0)**2)**(1/2)
        if distance <= (D/2):
            I_cyl[j][i] = 1
        else:
            I_cyl[j][i] = 0

"""EVALUATION DISCRETIZATION COEFFICIENTS"""

# Bottom nodes
for i in range(0,N+2):
    a_E[0][i] = 0
    a_W[0][i] = 0
    a_N[0][i] = 0
    a_S[0][i] = 0
    a_P[0][i] = 1
    b_P[0][i] = 0
    Psi[0][i] = b_P[0][i]

# Top Nodes
for i in range(0,N+2):
    a_E[-1][i] = 0
    a_W[-1][i] = 0
    a_N[-1][i] = 0
    a_S[-1][i] = 0
    a_P[-1][i] = 1
    b_P[-1][i] = V_in*H
    Psi[-1][i] = b_P[-1][i]

# Inlet nodes
for i in range(0,N+2):
    for j in range(1,M+1):
```

```
        Psi[j][i] = V_in * Y_p[j-1]

# Cylinder Nodes
for i in range (N+2):
    for j in range (M+2):
        if I_cyl[j][i] == 1:
            Psi[j][i] = 1.2*V_in*H/2

"""GAUSS-SEIDEL METHOD"""

r = 1.0

for i in range (0, N+2):
    for j in range (0, M+2):
        if I_cyl[j][i]==1:
            Psi[j][i] = 1.2 * V_in * H/2
        else:
            Psi[j][i] = Psi_aux[j][i]
        a_E[j][i] = 0
        a_W[j][i] = 0
        a_N[j][i] = 0
        a_S[j][i] = 0
        a_P[j][i] = 0
        b_P[j][i] = 0

### TOP & BOTTOM
# Top
for i in range (0, N+2):
    a_E[-1][i] = 0
    a_W[-1][i] = 0
    a_N[-1][i] = 0
    a_S[-1][i] = 0
    a_P[-1][i] = 1
    b_P[-1][i] = V_in * H
    Psi[-1][i] = b_P[-1][i]

# Bottom
for i in range (0, N+2):
    a_E[0][i] = 0
    a_W[0][i] = 0
    a_N[0][i] = 0
    a_S[0][i] = 0
    a_P[0][i] = 1
    b_P[0][i] = 0
    Psi[0][i] = 0

### INLET & OUTLET FLOWS
# Inlet
for j in range (1, M+1):
    a_E[j][0] = 0
    a_W[j][0] = 0
    a_N[j][0] = 0
    a_S[j][0] = 0
    a_P[j][0] = 1
    b_P[j][0] = V_in * Y_p[j-1]
    Psi[j][0] = b_P[j][0]

# Outlet
for j in range (1, M+1):
    a_E[j][-1] = 0
    a_W[j][-1] = 1
    a_N[j][-1] = 0
    a_S[j][-1] = 0
    a_P[j][-1] = 1
```

```
    b_P[j][-1] = 0
    Psi[j][-1] = Psi[j][-2]

while r > delta:
    suma = 0.0
    suma_aux = 0.0

    # Internal Nodes
    for i in range (1, N+1):
        for j in range (1, M+1):
          if I_cyl[j][i] == 0:
            a_E[j][i] = (Rho_in/Rho[j][i+1])*(delta_M/(np.abs(X_P[i+1]-X_P[i])))
            a_W[j][i] = (Rho_in/Rho[j][i-1])*(delta_M/(np.abs(X_P[i]-X_P[i-1])))
            a_N[j][i] = (Rho_in/Rho[j+1][i])*(delta_N/(np.abs(Y_P[j+1]-Y_P[j])))
            a_S[j][i] = (Rho_in/Rho[j-1][i])*(delta_N/(np.abs(Y_P[j]-Y_P[j-1])))
            a_P[j][i] = a_E[j][i] + a_W[j][i] + a_N[j][i] + a_S[j][i]
            b_P[j][i] = 0
            Psi[j][i] = (a_E[j][i]*Psi[j][i+1] + a_W[j][i]*Psi[j][i-1] //
            + a_N[j][i]*Psi[j+1][i] + a_S[j][i]*Psi[j-1][i])/a_P[j][i]
            Psi[j][-1] = Psi[j][-2]

    for i in range (0, N+2):
        for j in range (0, M+2):
            suma = suma + Psi[j][i]
            suma_aux = suma_aux + Psi_aux[j][i]

    average = suma / (N*M)
    average_aux = suma_aux /(N*M)
    r = np.abs(average-average_aux)

    if r < delta:
        break

    for i in range (N+2):
        for j in range (M+2):
            Psi_aux[j][i] = Psi[j][i]

"""PLOT"""

plt.figure(figsize=(8, 8))
plt.contour(X_P, Y_P, Psi, levels = 38, colors='b', linestyles='solid')
circle = plt.Circle((L/2, H/2), D/2, color='black', fill=True)
plt.gca().add_patch(circle)
plt.axis('equal')
plt.title('Flow Around a Rotating Cylinder')
plt.xlabel('L')
plt.ylabel('H')
plt.grid()
plt.show()

"""FINAL CALCULATIONS"""

# Velocities
V_E = np.zeros((M+2,N+2))
V_W = np.zeros((M+2,N+2))
V_N = np.zeros((M+2,N+2))
V_S = np.zeros((M+2,N+2))
V_X = np.zeros((M+2,N+2))
V_Y = np.zeros((M+2,N+2))
V_P = np.zeros((M+2,N+2))

### TOP & BOTTOM
# Top
for i in range (0, N+2):
```

```
    V_E[-1][i] = 0
    V_W[-1][i] = 0
    V_N[-1][i] = 0
    V_S[-1][i] = 0
    V_X[-1][i] = V_in
    V_Y[-1][i] = 0
    V_P[-1][i] = np.sqrt((V_X[-1][i])**2 + (V_Y[-1][i])**2)

# Bottom
for i in range (0, N+2):
    V_E[0][i] = 0
    V_W[0][i] = 0
    V_N[0][i] = 0
    V_S[0][i] = 0
    V_X[0][i] = V_in
    V_Y[0][i] = 0
    V_P[0][i] = np.sqrt((V_X[0][i])**2 + (V_Y[0][i])**2)

### INLET & OUTLET FLOWS
# Inlet
for j in range (1, M+1):
    V_E[j][0] = 0
    V_W[j][0] = 0
    V_N[j][0] = 0
    V_S[j][0] = 0
    V_X[j][0] = V_in
    V_Y[j][0] = 0
    V_P[j][0] = np.sqrt((V_X[j][0])**2 + (V_Y[j][0])**2)

# Outlet
for j in range (1, M+1):
    V_E[j][-1] = 0
    V_W[j][-1] = 0
    V_N[j][-1] = 0
    V_S[j][-1] = 0
    V_X[j][-1] = V_X[j][-2]
    V_Y[j][-1] = 0
    V_P[j][-1] = np.sqrt((V_X[j][-1])**2 + (V_Y[j][-1])**2)

# Internal Nodes
for i in range (1, N+1):
    for j in range (1, M+1):
        V_E[j][i] = -(Psi[j][i+1]-Psi[j][i]) / np.abs(X_P[i+1]-X_P[i])
        V_W[j][i] = (Psi[j][i-1]-Psi[j][i]) / np.abs(X_P[i]-X_P[i-1])
        V_N[j][i] = -(Psi[j+1][i]-Psi[j][i]) / np.abs(Y_P[j+1]-Y_P[j])
        V_S[j][i] = (Psi[j-1][i]-Psi[j][i]) / np.abs(Y_P[j-1]-Y_P[j])
        V_X[j][i] = (V_N[j][i] + V_S[j][i])/2
        V_Y[j][i] = (V_E[j][i] + V_W[j][i])/2
        V_P[j][i] = np.sqrt((V_X[j][i])**2 + (V_Y[j][i])**2)

# Specific Heat
c_P = 1034.09 - 2.849*(10**(-1))*T_in+7.817*(10**(-4))*T_in**2-4.971*(10**(-7))*T_in**3

# Temperature, Pressure & Density
T = np.zeros((M+2, N+2))
P = np.zeros((M+2, N+2))

for i in range (0, N+2):
    for j in range (0, M+2):
        T[j][i] = T_in + (V_in**2 - V_P[j][i]**2)/(2*c_P)
        P[j][i] = P_in * (T[j][i]/T_in)**(gamma/(gamma-1))
        Rho[j][i] = P[j][i]/(287.1*T[j][i])

# Temperature Plot
```

```python
plt.figure(figsize=(10, 10))
circle = plt.Circle((N/2, M/2), (D*N)/(2*L), color='black', fill=True)
plt.gca().add_patch(circle)
plt.imshow(T, cmap='jet', origin = 'lower')
plt.colorbar(label='K', format = '%.3f').ax.tick_params(labelsize=12)
plt.title('Temperature', fontweight='bold')
plt.gca().tick_params(axis='both', labelsize=12)
plt.xlabel('N', fontsize = 12)
plt.ylabel('M', fontsize = 12)
plt.show()

# Pressure Plot
plt.figure(figsize=(10, 10))
circle = plt.Circle((N/2, M/2), (D*N)/(2*L), color='black', fill=True)
plt.gca().add_patch(circle)
plt.imshow(P, cmap='inferno', origin = 'lower')
plt.colorbar(label='Pa').ax.tick_params(labelsize=12)
plt.title('Pressure', fontweight='bold')
plt.gca().tick_params(axis='both', labelsize=12)
plt.xlabel('N', fontsize = 12)
plt.ylabel('M', fontsize = 12)
plt.show()

# Velocity Plot
plt.figure(figsize=(10, 10))
circle = plt.Circle((N/2, M/2), (D*N)/(2*L), color='black', fill=True)
plt.gca().add_patch(circle)
plt.imshow(V_P, cmap='jet', origin='lower')
plt.colorbar(label='m/s').ax.tick_params(labelsize=12)
plt.title('Velocity', fontweight='bold')
plt.gca().tick_params(axis='both', labelsize=12)
plt.xlabel('N', fontsize = 12)
plt.ylabel('M', fontsize = 12)
plt.show()

"""DRAG AND LIFT"""

drag = np.zeros((M+2, N+2))
lift = np.zeros((M+2, N+2))
drag_tot = 0.0
lift_tot = 0.0

# Drag
for i in range (N+2):
    for j in range (M+2):
        if I_cyl[j][i] == 1 and I_cyl[j][i-1] == 0 :
            #lift[j][i] = P[j+1][i]*delta_M - P[j-1][i]*delta_M
            drag[j][i] = P[j][i-1]*delta_M
        if I_cyl[j][i] == 1 and I_cyl[j][i+1] == 0 :
            drag[j][i] = -P[j][i+1]*delta_M

        drag_tot = drag[j][i] + drag_tot

# Lift
for i in range (N+2):
    for j in range (M+2):
        if I_cyl[j][i] == 1 and I_cyl[j-1][i] == 0 :
            #lift[j][i] = P[j+1][i]*delta_M - P[j-1][i]*delta_M
            lift[j][i] = P[j-1][i]*delta_N
        if I_cyl[j][i] == 1 and I_cyl[j+1][i] == 0 :
            lift[j][i] = -P[j+1][i]*delta_N

        lift_tot = lift[j][i] + lift_tot

```

```
# Drag Plot
plt.figure(figsize=(10, 10))
plt.imshow(drag, cmap = 'jet')
plt.colorbar()
plt.title('Drag', fontweight='bold')
plt.gca().tick_params(axis='both', labelsize=12)
plt.xlabel('N', fontsize = 12)
plt.ylabel('M', fontsize = 12)
plt.show()

# Lift Plot
plt.figure(figsize=(10, 10))
plt.imshow(lift, cmap = 'jet')
plt.colorbar()
plt.title('Lift', fontweight='bold')
plt.gca().tick_params(axis='both', labelsize=12)
plt.xlabel('N', fontsize = 12)
plt.ylabel('M', fontsize = 12)
plt.show()

"""CIRCULATION"""

circ = np.zeros((M+2, N+2))
circ_tot = 0.0
for i in range (N+2):
  for j in range (M+2):
    if I_cyl[j][i] == 1 and I_cyl[j][i-1] == 0 :
      circ[j][i] = V_E[j][i]*delta_M + V_N[j][i]*delta_N //
      - V_W[j][i]*delta_M - V_S[j][i]*delta_N
      circ_tot = circ_tot + circ[j][i]
    if I_cyl[j][i] == 1 and I_cyl[j][i+1] == 0 :
      circ[j][i] = V_E[j][i]*delta_M + V_N[j][i]*delta_N //
      - V_W[j][i]*delta_M - V_S[j][i]*delta_N
      circ_tot = circ_tot + circ[j][i]
    if I_cyl[j][i] == 1 and I_cyl[j-1][i] == 0 :
      circ[j][i] = V_E[j][i]*delta_M + V_N[j][i]*delta_N //
      - V_W[j][i]*delta_M - V_S[j][i]*delta_N
      circ_tot = circ_tot + circ[j][i]
    if I_cyl[j][i] == 1 and I_cyl[j+1][i] == 0 :
      circ[j][i] = V_E[j][i]*delta_M + V_N[j][i]*delta_N //
      - V_W[j][i]*delta_M - V_S[j][i]*delta_N
      circ_tot = circ_tot + circ[j][i]
```