UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Master's In Space And Aeronautical Engineering

Computational Engineering Assignment 1

Non-Viscous Potential Flows

October, 2023

Author: Iñaki Fernandez Tena

Index

If it is in the reader's interest, further discussion of the code can be found in the following repository: **Code**

1 Introduction

In this work we will consider a non-viscous and irrotational fluid flow over an static and rotating cylinder. In addition, at some point of the report we will also assume that the fluid flow is incomprensible. The problem will be solved in two dimension. Under these context it is worth to use the stream function formulation.

For steady and 2-D flows, the velocities over X and Y directions, v_x and v_y, can be written in terms of the stream function ψ as it follows:

$$v_x = \frac{\rho_{ref}}{\rho}\frac{\partial \psi}{\partial y}; \qquad v_y = \frac{\rho_{ref}}{\rho}\frac{\partial \psi}{\partial x} \tag{1}$$

where ρ_{ref} is the density at references conditions. Since the non-viscous region is determined by the Euler equations it is worth to remark that the stream function fulfils the mass conservation equation.

In this case, since we are in 2D, it is of interest to take a close look into the vorticity, $\mathbf{w}$, which is expressed as:

$$\mathbf{w} = \nabla \times \mathbf{v}. \tag{2}$$

Using Eq. (1) the vorticity can be expressed as:

$$\frac{\partial}{\partial x}\left(\frac{\rho_{ref}}{\rho}\frac{\partial \psi}{\partial y}\right) + \frac{\partial}{\partial y}\left(\frac{\rho_{ref}}{\rho}\frac{\partial \psi}{\partial y}\right) = -w_z. \tag{3}$$

As it was mentioned above, we are considering an irrotational flow so that the vorticity is zero by definition:

$$\frac{\partial}{\partial x}\left(\frac{\rho_{ref}}{\rho}\frac{\partial \psi}{\partial y}\right) + \frac{\partial}{\partial y}\left(\frac{\rho_{ref}}{\rho}\frac{\partial \psi}{\partial y}\right) = 0. \tag{4}$$

Therefore, if we consider that the flow is incomprensible ($\rho_{ref}/\rho = 1$) the equation for the stream function becomes into Laplace's equation:

$$\nabla^2 \psi = 0. \tag{5}$$

The solutions of the Laplace's equation will give us the analytical solution for the flow over a cylinder, which will be discussed in Section **4.2**. This problem is also known as potential flow.

In this work we are interested in the numerical solution of a non-viscous potential flow. In Computational Fluid Dynamics (CFD) the numerical solutions are based in the discretization of the governing equations. Instead of discretazing the Laplace's equation (Eq. 5) we will start from the circulation. Using the Stoke's theorem and considering an irrotational

flow, the circulation can be written as:

$$\Gamma = \int_S (\nabla \times \mathbf{v}) dS = \oint_C \mathbf{v} d\mathbf{l} = 0. \tag{6}$$

Now, let's delve into the finite volume method. This approach involves breaking down our computational domain into smaller Control Volumes (CVs). In Figure 1, we can see a representation of a CV along with its neighboring CVs.

Each CV is enclosed by four walls, each of which contains a node: one to the east (e), one to the north (n), one to the west (w), and one to the south (s). At the center of the CV, we find a node denoted as P. Surrounding this CV, there are four neighboring CVs, each with its central node labeled as East (E), North (N), West (W), and South (S). To specify the dimensions of the CV, we refer to its length in the X direction as Δxp and its height in the Y direction as Δyp.

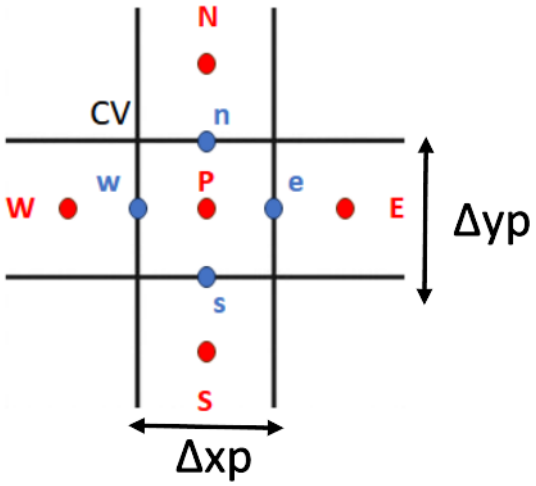


Figure 1: *A scheme of a CV and it's neighbours. In red the centred nodes: P, S, N, W and E. In blue the nodes centred in the walls of the CV: s, n, w and e.*

Once the discretization is defined, the circulation Γ, Eq. 6, for the internal node P can be written (up to second order approximation in the line integrals) as:

$$\Gamma \sim v_{ye}\Delta_{yP} - v_{xn}\Delta_{xP} - v_{yw}\Delta_{yP} + v_{xs}\Delta_{xP} = 0. \tag{7}$$

We introduce the stream function definition (Eq. 1):

$$-\frac{\rho_{ref}}{\rho}\left(\frac{\partial\psi}{\partial x}\right)_e \Delta_{yP} - \frac{\rho_{ref}}{\rho}\left(\frac{\partial\psi}{\partial y}\right)_n \Delta_{xP} + \frac{\rho_{ref}}{\rho}\left(\frac{\partial\psi}{\partial x}\right)_w \Delta_{yP} + \frac{\rho_{ref}}{\rho}\left(\frac{\partial\psi}{\partial y}\right)_s \Delta_{xP} = 0, \tag{8}$$

and approximating the differentials (up to second-order approximation in the derivatives):

$$-\frac{\rho_{ref}}{\rho}\frac{\psi_E-\psi_P}{d_{PE}}\Delta_{yP}-\frac{\rho_{ref}}{\rho}\frac{\psi_N-\psi_P}{d_{PN}}\Delta_{xP}+\frac{\rho_{ref}}{\rho}\frac{\psi_P-\psi_W}{d_{PW}}\Delta_{yP}+\frac{\rho_{ref}}{\rho}\frac{\psi_P-\psi_S}{d_{PS}}\Delta_{xP}=0. \quad (9)$$

We have to remark here that Eq. (9) is only for the central node P. So generalising it we obtain for all the internal nodes:

$$a_P\psi_P = a_E\psi_E + a_W\psi_W + a_N\psi_N + a_S\psi_S + b_p, \quad (10)$$

where a_i, ψ_i and b_p will be matrices of dimension $[N+2, M+2]$, being N and M the discretization mesh quantity in X direction and Y direction, respectively. This implies that Eq. (10) will have $(N+2)\times(M+2)$ equations and unknowns. The values of $a_i \quad i \in [P, E, W, N, S]$ and b_P can be expressed as:

$$a_E = \frac{\rho_{ref}}{\rho}\frac{\Delta_{yP}}{d_{PE}}\ ; \quad a_W = \frac{\rho_{ref}}{\rho}\frac{\Delta_{yP}}{d_{PW}} \quad (11)$$

$$a_N = \frac{\rho_{ref}}{\rho}\frac{\Delta_{xP}}{d_{PN}}\ ; \quad a_S = \frac{\rho_{ref}}{\rho}\frac{\Delta_{xP}}{d_{PS}} \quad (12)$$

$$a_P = a_E + a_W + a_N + a_S\ ; \quad b_P = 0 \quad (13)$$

With Eq. (10), one is able to solve the stream function ψ for all the CVs. Once the stream function is known te velocities can be obtained using Eq. (1).

We don't have to forget that the Boundary Conditions (BC) play a key role in numerical calculations. In this work, the non-viscous potential flow will be solved in a channel. This implies that we will have two solid walls (top and bottom), one inlet wall and one outlet wall, as it is pictured in Figure 2. A more detailed description around BC will be given in Section **2**.

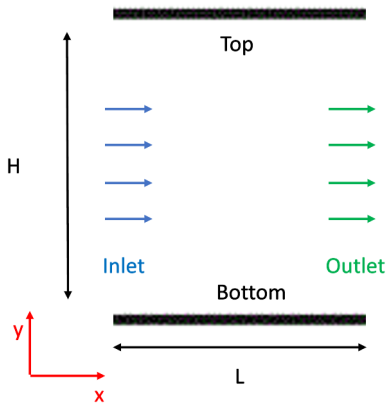


Figure 2: *A scheme of the channel.*

2 Gauss-Seidel Method

Our objective is to solve Eq. (10) and to add the BCs, so that the hole problem for the stream function is solved.

To initiate the analysis, we define the values of parameters at different boundaries as follows (Fig. 2): at the bottom boundary, we set $a_E = a_W = a_N = a_S = 0$, $a_P = 1$, and establish b_P and ψ_B as 0; at the top boundary, similar conditions are applied with b_P and ψ_T determined as $v_{in}H$, where v_{in} signifies the initial velocity, and H represents the cavity height; at the inlet boundary, we maintain $a_E = a_W = a_N = a_S = 0$, $a_P = 1$, and set b_P as $\psi = v_{in}y_{pj}$; for the outlet boundary, we specify $(\partial\psi/\partial x = 0)$, so that $\psi_P \sim \psi_W$, while $a_E = a_N = a_S = 0$, $a_P = 1$, $a_W = 1$, and b_P is set to 0.

To calculate the remaining internal nodes, we employ the Gauss-Seidel Method, an iterative numerical approach adapted for solving systems of linear equations, particularly useful for tackling extensive matrices and systems. In this context, we apply it to solve Eq. (10) for ψ_P. After each iteration, once all node computations are completed, we browse whether the difference between the current ψ_P values and their counterparts from the previous iteration, denoted as ψ_P^*, falls below a specified threshold ε. If this condition is met, we consider the Gauss-Seidel method as having converged, and the final solution is given by ψ_P. If not, we iterate once more across all nodes, continually observing whether $|\psi_P - \psi_P^*| < \varepsilon$ holds true.

When we are dealing with the cylinder, it becomes necessary to distinguish between CVs situated inside the cylinder (referred to as solid CVs) and those outside (known as fluid CVs). To achieve this differentiation, we introduce a matrix named I_{body}, where we denote as one those CVs residing inside the cylinder and zero to denote CVs outside the cylinder.

3 Code Structure

In this section we introduce the code structure. As it will be shown in Section 8, the code is written for python3 language. Further discussion on the code can be found in the repository linked above the index.

- Establish the input data, which is segregated into two distinct sections: one dedicated to the physical aspects of the problem encompassing parameters like the cavity's length and height (L and H), the input velocity v_{in}, and thermodynamic variables such as initial temperature, pressure, and density (T_{in}, P_{in}, and ρ_{in}). It's worth noting that the value of ρ_{in} holds negligible significance in our context, given that we are solving for non-viscous potential flow within an incompressible fluid. Although the code incorporates the fraction ρ_{ref}/ρ, it consistently evaluates to unity.
- Generate the mesh with $N \times M$ CVs over L and H.
- Define the cylinder with a diameter D. In the context of parallel flow, there is no mandatory requirement for its definition.
- Define all the matrices with dimension of $(N+2) \times (M+2)$.
- Generate the I_{body} matrix.
- Initialize the stream function and evaluate all the internal nodes $a_i \;\; i \in [P, E, W, N, S]$ and b_P. In addition to this we compute the BCs.
- Solve the Gauss-Seidel algorithm for all internal nodes.
- Compute the velocities v_P, the temperature T_P, the pressure P_P, the lift $\mathbf{L}$, the dragg $\mathbf{D}$ and the circulation Γ.
- Create all the plots and save them.

4 Code Verification and Validation

4.1 Verification: BCs and Mesh Refinement Studies

Before presenting the results obtained in this work, it is crucial to conduct an analysis to ensure the code's proper functionality. In the initial phase, it is essential to examine whether the boundary conditions (BCs) are accurately reflected in the solutions. To illustrate this, let's consider the numerical solution for the static cylinder case, as depicted in Figure 3.

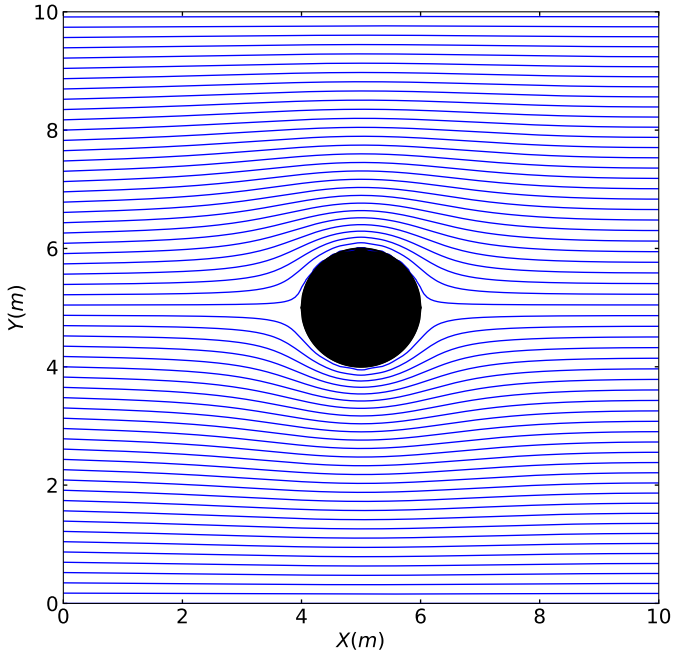
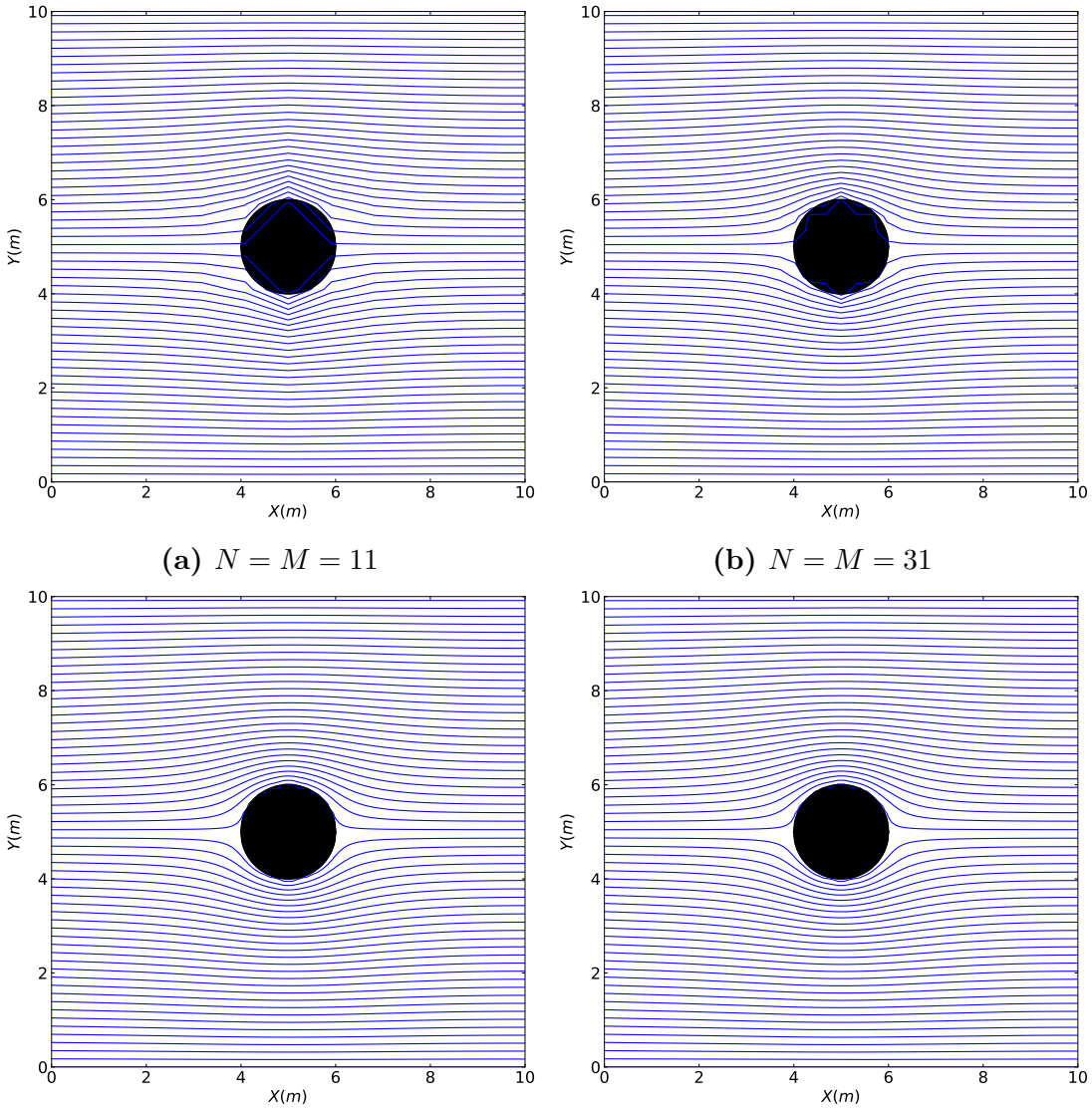


Figure 3: *Numerical solution of the stream function over a cylinder. The cavity dimensions are* $L = H = 10$ *m. The initial velocity is* $v_{in} = 1.5$ *m/s. The mesh is constructed as* $N = M = 171$*. The convergence criteria in the Gauss-Seidel algorithm is* $\varepsilon = 10^{-6}$*. And the diameter of the cylinder is* $D = 2$ *m.*

Upon closer examination, it becomes evident that at the inlet, all the streamlines run parallel to each other, aligning with our expectations (the same behavior can also be observed in the parallel flow case, as shown in Figure 7). Additionally, around the cylinder, the streamlines conform to the cylinder's shape, demonstrating that the blocking-off method is functioning as expected.

The blocking-off method will yield improved performance as we increase the mesh resolution. To examine this phenomenon, we have generated streamlines around a cylinder for various mesh configurations, as depicted in Figure 4. The visual representation clearly illustrates that the blocking-off technique works more effectively with finer meshes, aligning with our expectations. Consequently, larger meshes yield better outcomes.

However, larger meshes place higher demands on computational resources, resulting in



(a) $N = M = 11$ **(b)** $N = M = 31$

(c) $N = M = 101$ **(d)** $N = M = 131$

Figure 4: *Numerical solution of the stream function over a cylinder. The cavity dimensions are* $L = H = 10$ *m. The initial velocity is* $v_{in} = 1.5$ *m/s. The convergence criteria in the Gauss-Seidel algorithm is* $\varepsilon = 10^{-6}$*. And the diameter of the cylinder is* $D = 2$ *m.*

prolonged computation times. To illustrate this connection, we have depicted in Figure 5 the convergence steps required by the Gauss-Seidel method for symmetric mesh refinement in the context of a stationary cylinder. The graph shows that an augmentation in mesh size corresponds to an escalation in the number of convergence steps necessary for the Gauss-Seidel method, consequently extending the computational duration. It is remarkable that the growth of the curve exhibits a rate faster than linear. This same behaviour has been observed in both clockwise and anti-clockwise numerical solutions.

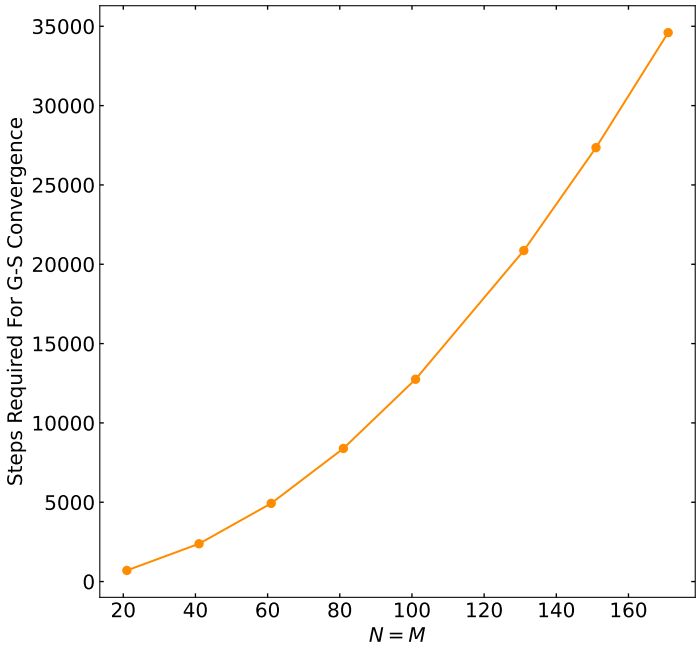


Figure 5: *Required steps for G-S algorithm to converge vs mesh ($N = M$). This curve have been done for the static case.*

Lastly, we conducted simulations with a more strict convergence criterion, specifically $\varepsilon < 10^{-6}$. The observed behavior depicted in Figure 4 remained consistent. This suggests that a convergence criterion of $\varepsilon = 10^{-6}$ is sufficient for the Gauss-Seidel algorithm.

4.2 Validation: Comparison with the Analytical Solution

As mentioned in Section **4.2**, non-viscous potential flow can be solved analytically. The solution for the stream function can be obtained by solving Eq. (5).

Before going into solving streamlines around a cylinder, it is beneficial to address the problem of a simple parallel flow. This can be accomplished by solving Laplace's equation for a uniform stream ψ_U. The solution can be expressed as (in cylindrical coordinates)[1]:

$$\psi_U = V_\infty r \sin\theta \tag{14}$$

Here, V_∞ represents the free stream velocity. Figure 6 illustrates the streamlines for $L = 10$ and $H = 10$.

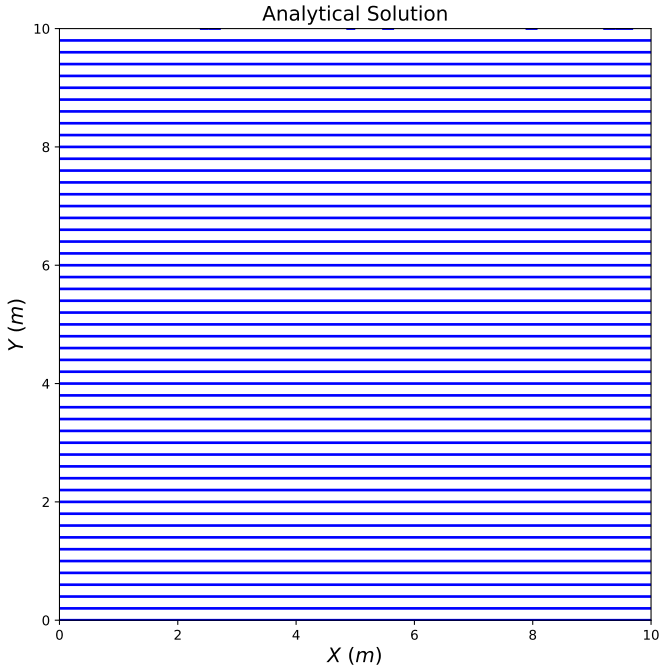
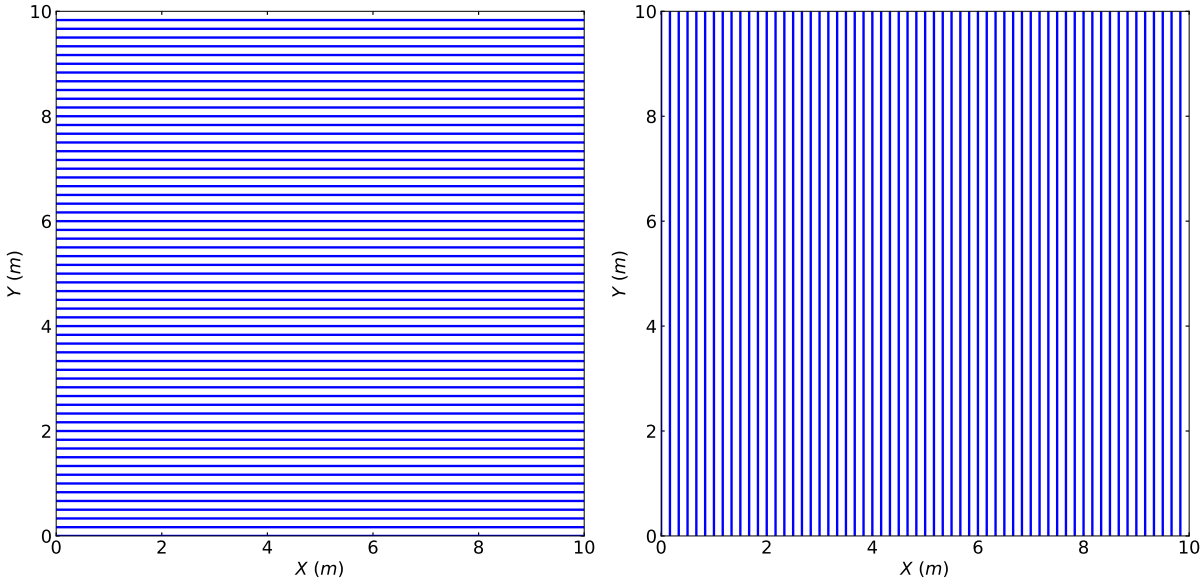


Figure 6: *Analytical solution of the Laplace's equation for a parallel flow. The cavity dimensions are* $L = 10$ *and* $H = 10$.

We can compare this result with the numerical solution, as depicted in Figure 7a. Here, we maintain identical dimensions with $N = M = 41$.

To further validate our numerical solution for the parallel flow, we have also plotted the solution for a vertical case in Figure 7b, employing the same numerical parameters. The similarities between the analytical and numerical solutions is readily apparent.

Before writing the analytical solution of the stream function over a static cylinder we have to remember that if ψ_1 and ψ_2 are independent solutions of Eq. (5), $\psi_3 = \psi_1 + \psi_2$ will also be a solution of the equation. In this case, a combination of the solution of an uniform stream, ψ_U, + the solution of a doublet (a combination of a source and a sink), ψ_D, yields



(a) *Horizontal parallel stream lines.* **(b)** *Vertical parallel stream lines.*

Figure 7: *The cavity dimension are $L = H = 10\ m$. The initial velocity is $v_{in} = 0.15\ m/s$. The mesh is constructed as $N = M = 41$. The convergence criteria in the Gauss-Seidel algorithm is $\varepsilon = 10^{-6}$.*

to a flow over a cylinder, ψ_C. The solution in cylindrical coordinates can be expressed as[1]:

$$\psi_C = V_\infty r \sin\theta \left(1 - \frac{R^2}{r^2}\right) \tag{15}$$

where R is the radius of the cylinder. The analytical solution for a cylinder of a diameter $D = 2$ have been plotted in Figure 8.

We can compare this result with the numerical solution, as shown in Figure 3. In general, the analytical and numerical solutions exhibit a high degree of similarity. However, there is a slight disparity in the streamlines located at the center of the cavity along the Y direction.

In the analytical solution, two distinct points are evident where the velocity is zero; these points are referred to as stagnation points. However, in the numerical solution, we observe that these two stagnation points are not precisely reproduced. Instead, the streamlines deviate slightly upward in the Y direction, following the boundary of the cylinder. For smaller meshes, this characteristic streamline actually penetrates the cylinder, taking an unconventional path, as depicted in Figure 9, passing from the left side to the right side. As we increase the mesh resolution, this particular streamline begins to more closely conform to the contour of the cylinder.

This discrepancy in the central streamline may be a direct consequence of the blocking-off method, which, for smaller meshes, does not accurately simulate the cylinder. With an

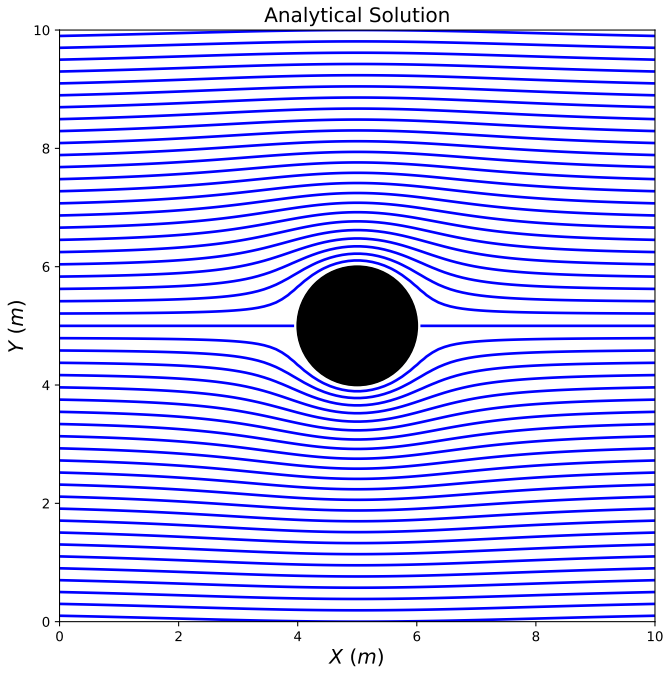


Figure 8: *Analytical solution of the Laplace's equation for non-viscous potential flow over a cylinder. The cavity dimensions are* $L = H = 10$*. The diamater of the cylinder is* $D = 2$*.*

increase in mesh size, the cylinder becomes better represented in the numerical solution.

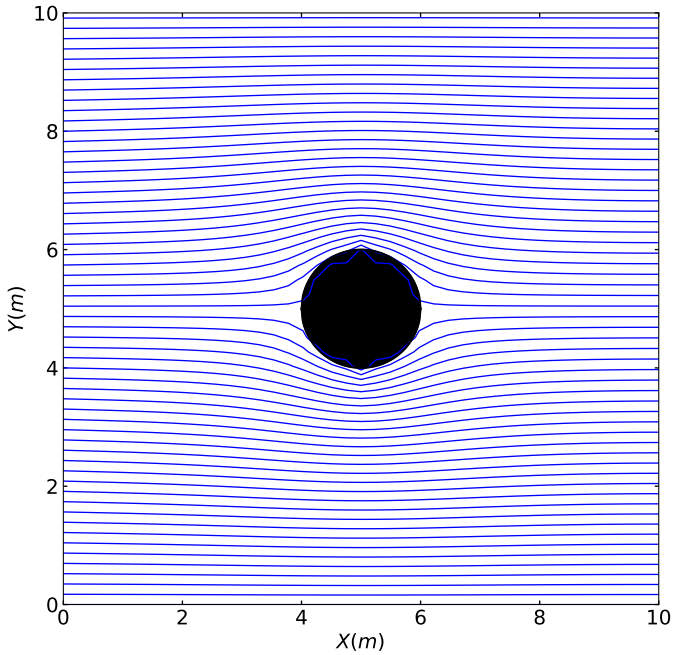


Figure 9: *Numerical solution of the stream function over a cylinder. The cavity dimensions are* $L = H = 10\ m$*. The initial velocity is* $v_{in} = 15\ m/s$*. The mesh is constructed as* $N = M = 41$*. The convergence criteria in the Gauss-Seidel algorithm is* $\varepsilon = 10^{-6}$*. And the diameter of the cylinder is* $D = 2\ m$*.*

Finally the rotating cylinder can be analytically constructed by adding a line vortex to

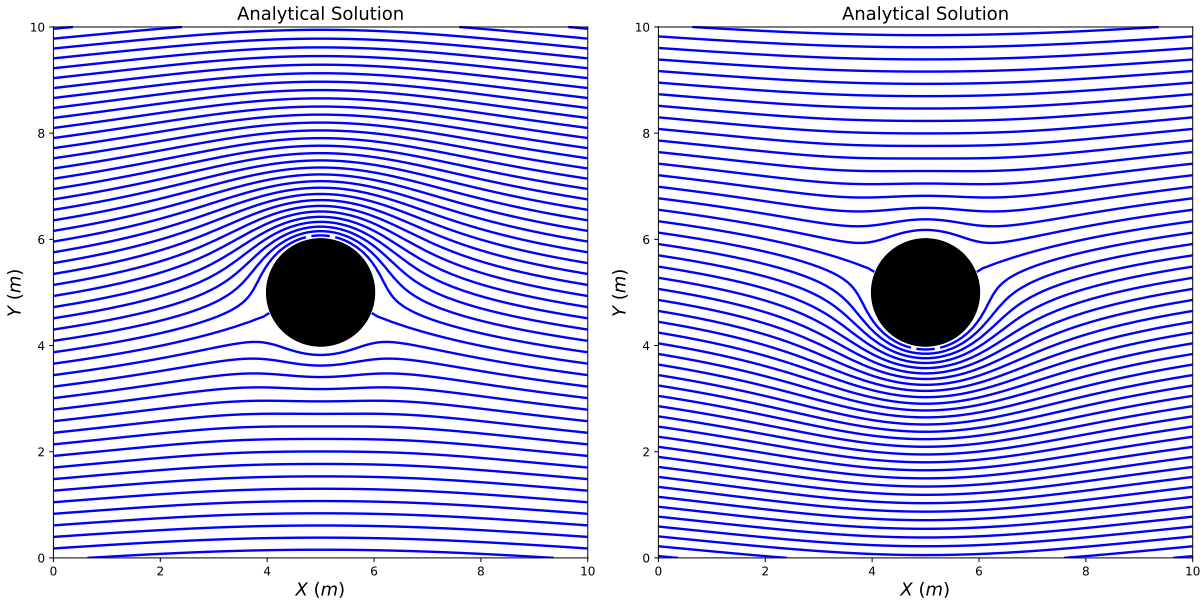
the solution [1]:

$$\psi_V = \frac{\Gamma}{2\pi} \ln\left(\frac{r}{R}\right) \tag{16}$$

So that the total stream function is expressed as:

$$\psi_C = V_\infty r \sin\theta \left(1 - \frac{R^2}{r^2}\right) \pm \frac{\Gamma}{2\pi} \ln\left(\frac{r}{R}\right). \tag{17}$$

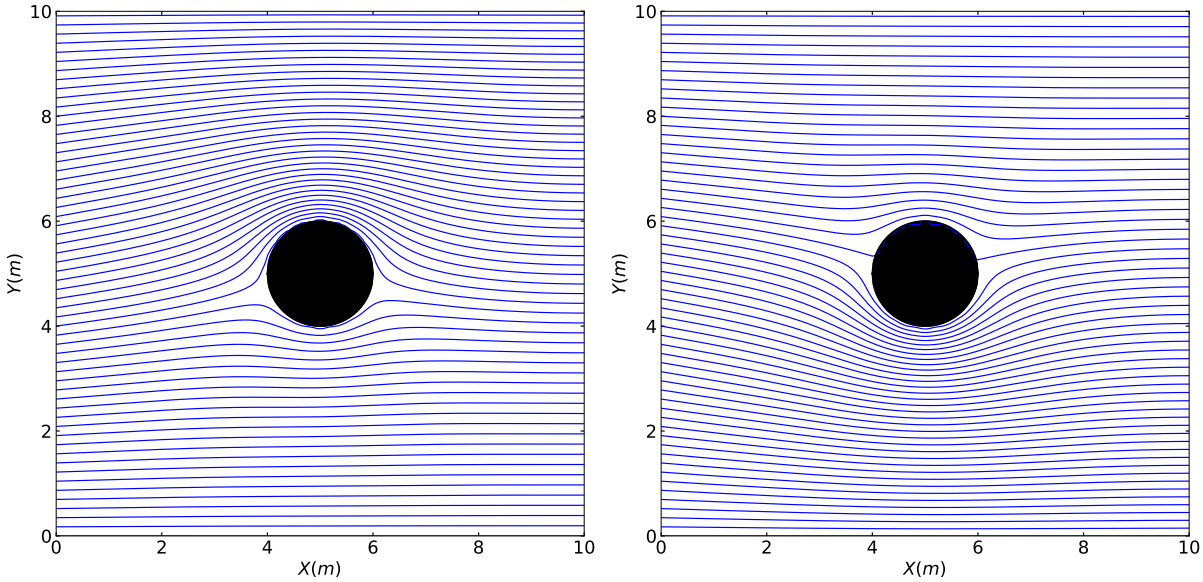
for the + case we get a clockwise rotation while for the - case we get an anti-clockwise rotation. The analytical clockwise and anti-cloc wise solutions and the numerical clockwise and anti-clockwise solutions of the stream function have been plotted in Figure 10a, 10b, 11a and 11b, respectively.



(a) *Clockwise rotation.* **(b)** *Anticlockwise rotation*

Figure 10: *Analytical solution for rotating cylinder. We have used a circulation of* $\Gamma = 0.5$.

We can note that in the rotation case the issue with the stagnation points in the numerical solution persists. This issue may indeed be attributed to the blocking-off method. Nonetheless, despite this discrepancy, we still obtain highly similar solutions between the computational and numerical approaches.



(a) *Clockwise rotation.* **(b)** *Anticlockwise rotation*

Figure 11: *Numerical solution of the stream function over a cylinder. The cavity dimensions are* $L = H = 10$ *m. The initial velocity is* $v_{in} = 1.5$ *m/s. The mesh is constructed as* $N = M = 171$*. The convergence criteria in the Gauss-Seidel algorithm is* $\varepsilon = 10^{-6}$*. And the diameter of the cylinder is* $D = 2$ *m. We have added/rested 1.5 to the body stream function.*

5 Physical Aspects

In this Section we present the physical aspects required to discuss the results. Firstly, the velocity can be obtained directly from stream function as it follows:

$$v_P = \sqrt{v_{xP}^2 + v_{yP}^2} \tag{18}$$

where,

$$v_{xP} = \frac{v_{xn} + v_{xs}}{2} \; ; \quad v_{yP} = \frac{v_{ye} + v_{yw}}{2} \tag{19}$$

and,

$$v_{xn} = \frac{\rho_{ref}}{\rho} \frac{\psi_N - \psi_P}{d_{PN}} \; ; \quad v_{xs} = -\frac{\rho_{ref}}{\rho} \frac{\psi_N - \psi_S}{d_{PS}} \tag{20}$$

$$v_{yw} = \frac{\rho_{ref}}{\rho} \frac{\psi_N - \psi_W}{d_{PW}} \; ; \quad v_{ye} = -\frac{\rho_{ref}}{\rho} \frac{\psi_E - \psi_P}{d_{PE}} \tag{21}$$

Secondly, the temperature can be obtained from the velocity:

$$T_P = T_{ref} + \frac{v_{ref}^2 - v_P^2}{2\overline{c_p}} \tag{22}$$

where $\overline{c_p}$ is the mean value of the specific heat at constant pressure. With the temperature obtained, the pressure is straightforward:

$$P_P = P_{ref} \left(\frac{T_P}{T_{ref}} \right)^{\frac{\gamma}{\gamma-1}} . \tag{23}$$

And with the pressure defined, the lift, L and the drag D can be obtained as it follows:

$$L = \sum_i^n (P_i \Delta_{xp})_N - (P_i \Delta_{xp})_S \tag{24}$$

$$D = \sum_i^n (P_i \Delta_{yp})_W - (P_i \Delta_{yp})_E \tag{25}$$

where n goes from $i = 1$ to all the CVs around the cylinder.

6 Results

The results will be shown for the static, clockwise and anti-clockwise cylinder for the incomprensible case. The physical and numerical input data used for the numerical calculations have been summarized in Table 1. We have to remark here that for Eqs. (22) and (23) we make use of $v_{in} = v_{ref}$, $T_{in} = T_{ref}$, and $P_{in} = P_{ref}$. And for the hole simulation $\rho_{in} = \rho_{ref}$.

Name	Symbol	Value	Units
X Direction Mesh	N	171	#
Y Direction Mesh	M	171	#
G-S Convergence Criteria	ε	10^{-6}	#
Cavity Length	L	10	m
Cavity Height	H	10	m
Cylinder Diameter	D	2	m
Input Temperature	T_{in}	298	K
Input Pressure	P_{in}	1.013×10^5	N / m^2
Input Velocity	V_{in}	1.15	m / s
Input Density	ρ_{in}	1.1225	kg / m^3
Specific Heat Ratio	γ	1.4	#
Specific Heat at Cons. Press.	$\overline{c_p}$	1005	J / (kg K)

Table 1: *Used numerical and physical input data for simulations.*

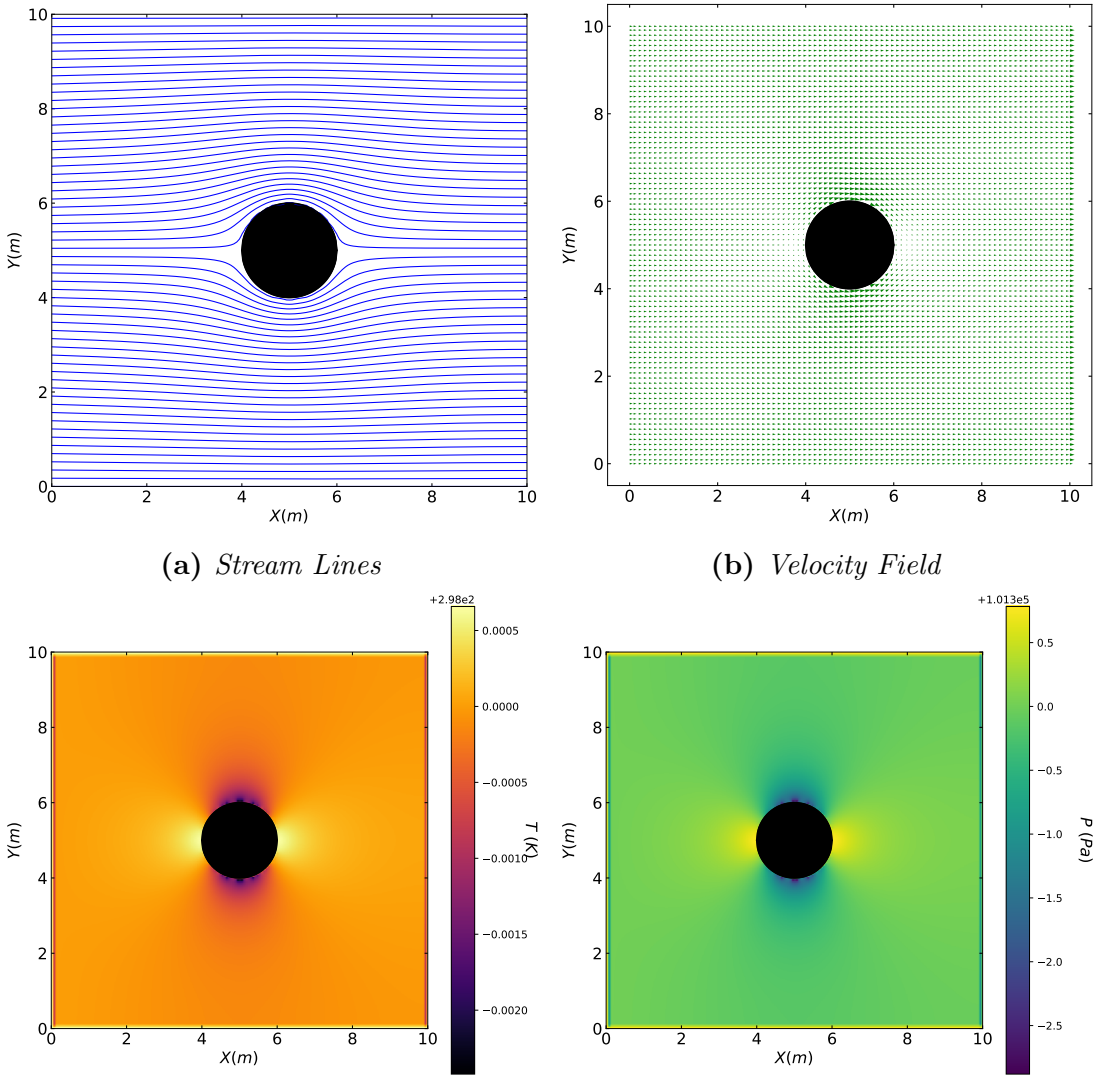
The results will be presented as it follows:

- First the static solution followed by the clockwise and anti-clockwise solutions.
- In each case a figure with the solution of the stream lines, velocity field, temperature field and pressure field will be presented. Followed by a brief discussion.
- In each case the values of lift, drag and circulation will be presented. Followed by a brief discussion.

6.1 Static Cylinder

In Figure 12, we present the results of the static cylinder. Specifically, we show the streamlines, the velocity field represented using arrows, the temperature field, and the pressure field in Figures 12a, 12b, 12c, and 12d, respectively.

It is evident that we obtain nearly symmetrical solutions in both the X and Y directions, as one would expect. Notably, the velocity exhibits higher magnitudes at the upper and lower regions of the cylinder. This phenomenon implies, in accordance with Eq. (22), that the temperature will be lower at the upper and lower portions of the cylinder. Given the proportional relationship between temperature and pressure (due to $\gamma > 1$), the pressure



(a) *Stream Lines* **(b)** *Velocity Field*

(c) *Temperature Field* **(d)** *Pressure Field*

Figure 12: *Results for the non-viscous potential flow around the static cylinder.*

will consequently assume lower values at the top and bottom, as illustrated in Figure 12d.

In terms of the forces, the obtained lift and drag, and circulation are:

$$L = 8.542 \times 10^{-7}\ N\ ; \quad D = 7.863 \times 10^{-4}\ N\ ; \quad \Gamma = 6.172 \times 10^{-8}\ \frac{m^2}{s} \tag{26}$$

The obtained solutions align closely with the expected outcomes. Firstly, the negligible lift observed is consistent with the symmetric pressure distribution along the cylinder, implying an absence of forces in the Y direction and, consequently, no lift. Secondly, the minimal drag obtained is in line with the assumption of a non-viscous flow, which should theoretically yield zero drag. Finally, the small circulation observed aligns with the consideration of an irrotational flow, where no circulation should appear. The presence of very small, non-zero values can be attributed to numerical errors.

6.2 Rotating Cylinder: Clockwise

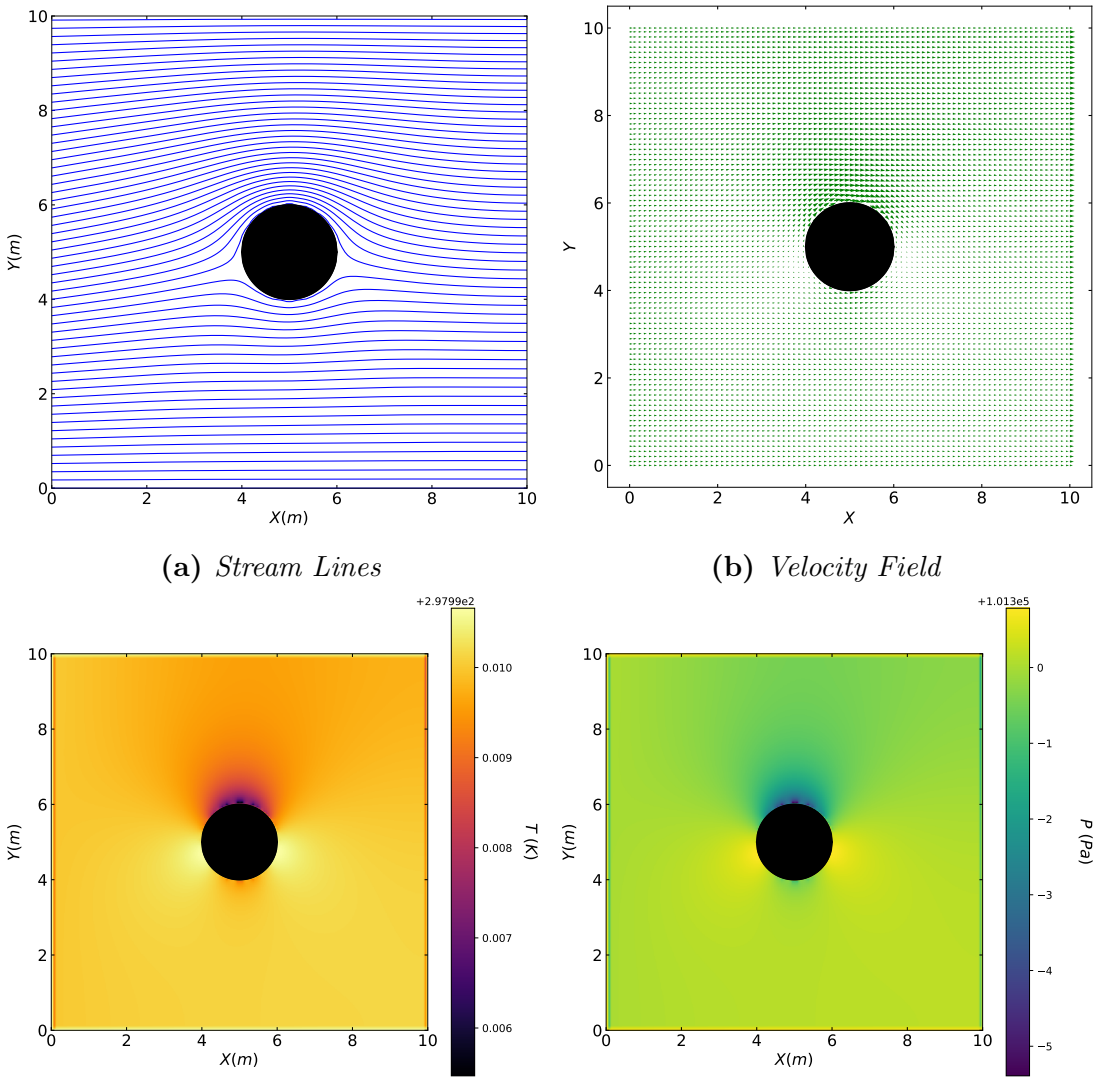
In Figure 13, we present the results of the clockwise rotating cylinder. Specifically, we show the streamlines, the velocity field represented using arrows, the temperature field, and the pressure field in Figures 13a, 13b, 13c, and 13d, respectively.

It is evident that we obtain non-symmetrical solutions in Y directions, as one would expect. Notably, the velocity exhibits higher magnitudes at the upper regions of the cylinder. This phenomenon implies, in accordance with Equation (22), that the temperature will be lower at the upper of the cylinder. Given the proportional relationship between temperature and pressure (due to $\gamma > 1$), the pressure will consequently assume lower values at the top, as illustrated in Figure 12d. All this non-symmetric velocity, temperature and pressure fields, implies that the cylinder will rotate clockwise.

In terms of the forces, the obtained lift and drag, and circulation are:

$$L = 5.274\ N\ ; \quad D = -0.167\ N\ ; \quad \Gamma = -0.165\ \frac{m^2}{s} \tag{27}$$

The obtained solutions align closely with the expected outcomes. Firstly, the pressure difference along the cylinder creates a lift, which is consistent with the non-symmetric pressure distribution along the cylinder. Secondly, a non-negligible drag appears. This occurs because the velocity field is not exactly symmetrical due to the rotation of the cylinder. Finally, the circulation observed along the cylinder aligns with the solution of a



(a) *Stream Lines* **(b)** *Velocity Field*

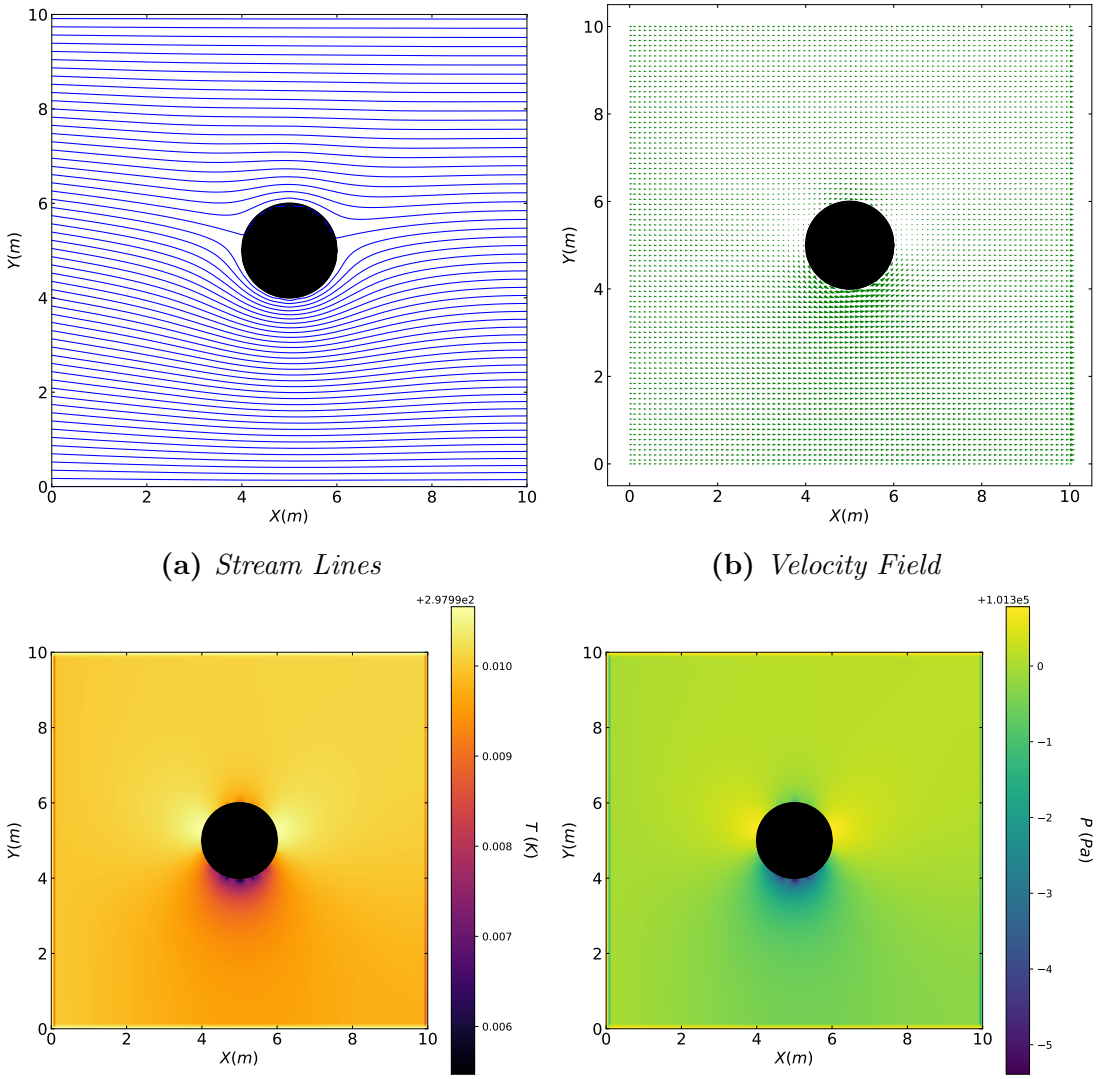
(c) *Temperature Field* **(d)** *Pressure Field*

Figure 13: *Results for the non-viscous potential flow around the clockwise rotating cylinder.*

rotating cylinder.

6.3 Rotating Cylinder: Anti-Clockwise

In Figure 14, we present the results of the anti-clockwise rotating cylinder. Specifically, we show the streamlines, the velocity field represented using arrows, the temperature field, and the pressure field in Figures 14a, 14b, 14c, and 14d, respectively.



(a) *Stream Lines* **(b)** *Velocity Field*

(c) *Temperature Field* **(d)** *Pressure Field*

Figure 14: *Results for the non-viscous potential flow around the anti-clockwise rotating cylinder.*

It is evident that we obtain non-symmetrical solutions in Y directions, as one would expect. In addition to this, this non-symmetrical distribution is contrary to the one obtained for

the clockwise case. Notably, the velocity exhibits higher magnitudes at the lower regions of the cylinder. This phenomenon implies, in accordance with Equation (22), that the temperature will be lower at the bottom of the cylinder. Given the proportional relationship between temperature and pressure (due to $\gamma > 1$), the pressure will consequently assume lower values at the bottom , as illustrated in Figure 12d. All this non-symmetric velocity, temperature and pressure fields, implies that the cylinder will rotate anti-clockwise.

In terms of the forces, the obtained lift and drag, and circulation are:

$$L = -5.274\ N\ ; \quad D = -0.167\ N\ ; \quad \Gamma = 0.165\ \frac{m^2}{s} \tag{28}$$

The obtained solutions align closely with the expected outcomes. Firstly, the pressure difference along the cylinder creates a lift, which is consistent with the non-symmetric pressure distribution along the cylinder. Secondly, a non-negligible drag appears. This occurs because the velocity field is not exactly symmetrical due to the rotation of the cylinder. Finally, the circulation observed along the cylinder aligns with the solution of a rotating cylinder.

It's worth noting that the lift and circulation values obtained in this subsection have equal magnitudes but opposite signs when compared to the clockwise rotation case. This outcome is expected, since the only difference between clockwise and anti-clockwise numerical simulations is the input that changes the direction of rotation.

7 Conclusions

In conclusion, in this work the numerical solution of a non-visocus potential flow over a static, clockwise rotating and anti-clockwise rotation cylinder was obtained.

Firstly, a validation process was carried out with an examination of boundary conditions, demonstrating that the code accurately represents flow behaviours, such as streamlines aligning with expectations at the inlet and conforming to the cylinder's shape around the object. Moreover, a discussion of the effect of mesh refinement on the code's was performed. It was observed that finer mesh resolutions improved the effectiveness of the blocking-off technique, obtaining better numerical solutions. However, it's important to note that larger meshes also increased computational demands and extended computation times, showing a non-linear relationship between mesh size and convergence steps. Additionally, by applying a stricter convergence criterion, the code's robustness was confirmed, affirming that a convergence criterion of $\varepsilon = 10^6$ is sufficient for the Gauss-Seidel algorithm.

The analytical solutions for simple flow cases, including uniform stream and flow around a static cylinder, were compared with numerical results. Generally, there was a high agreement between the two approaches, with slight disparities near the centre of the cylinder in the numerical solution. This discrepancy was attributed to the blocking-off method, which improved with increased mesh resolution. In addition to this, the code was validated for flow around a rotating cylinder, where clock-wise and anti-clockwise rotations were analyzed. Despite persistent issues with stagnation points in the numerical solution, likely due to the blocking-off method, the code and analytical solutions exhibited strong similarities.

In terms of the physical aspects of the results, for the static cylinder case (Figure 12), we observed nearly symmetrical solutions in both the X and Y directions. This symmetry is reflected in the velocity, temperature, and pressure fields. The forces calculated, including lift, drag, and circulation, align closely with theoretical expectations. The negligible lift is consistent with the symmetry of the solution and drag and circulation are consistent with the non-viscous, irrotational flow assumption, while the small values in the three magnitudes can be attributed to numerical errors.

Moving on to the clockwise rotating cylinder (Figure 13), the solutions become non-symmetrical in the Y direction, as anticipated. The velocity, temperature, and pressure fields exhibit non-symmetric characteristics due to the cylinder's rotation. Consequently, the cylinder rotates in a clockwise direction. The forces, including lift, drag, and circulation, are in line with the expectations for a rotating cylinder, further validating the code's performance. Similarly, for the anti-clockwise rotating cylinder (Figure 14), we again observe non-symmetrical solutions in the Y direction, but in this case, they are opposite to the clockwise rotation case. The cylinder rotates anti-clockwise, and the forces obtained, such as lift, drag, and circulation, align with expectations, mirroring the results from the clockwise case.

Notably, the comparison between clockwise and anti-clockwise rotations reveals that the

lift and circulation values have equal magnitudes but opposite signs. This observation is consistent with the expected behavior, as the only difference between these simulations is the direction of rotation input.

References

[1] I. H. Shames, "Mechanics of fluids," 1982.

8 Appendix: Python3 Code

```
#=======================================
# Import modules
#=======================================

import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

#=======================================
# Physical input data
#=======================================

L    = 10.0                                 # Channel lenght.
H    = 10.0                                 # Channel height.
rho  = 1.225                                # Air density at sea level in kg/m^3.
    Incompresible = constant density at all mesh points.
T_in = 298.0                                # Normal air temperature in K.
P_in = 1.013E05                             # Air pressure at sea level in N/m^2.
V_in = 1.15                                 # Air velocity in m/s.
R    = 8.31                                 # Ideal gas constant in J/mol*K


#=======================================
# Numerical input data
#=======================================

N         = 41                              # Control volumes in x direction. Odd number!
M         = 41                              # Control volumes in y direction. Odd number!
delta_X   = L / N                           # Control volume lenght in x direction.
delta_Y   = H / M                           # Control volume height in y direction.
eps       = 1.0E-08                         # Gauss-Seidel method convergence parameter.
psi_B     = 0.0                             # Stream function at the bottom of the channel
    .
psi_T     = V_in * H                        # Stream function at the top of the channel.
psi_s     = (psi_B + psi_T) / 2.0           # Stream function start value.
t_max     = 1000000                         # G-S loop steps.
```

```
psi_body  = psi_T / 2.0                    # Stream function inside the body. Add +- 1.5
    for clockwise and anti-clockwise rotation.

# Mesh generation

x_cv = np.linspace(0, L, N+1)              # Generate x points with the same spacing
y_cv = np.linspace(0, H, M+1)              # Generate y points with the same spacing
x_p  = np.zeros(N+2)                       # Vectors for the centered control volumes
    with N+2 elements
y_p  = np.zeros(M+2)                       # Vectors for the centered control volumes
    with M+2 elements

# Fill x_p and y_p

for i in range(1, N+1):
    x_p[i] = (x_cv[i] + x_cv[i-1]) / 2.0

for j in range(1, M+1):
    y_p[j] = (y_cv[j] + y_cv[j-1]) / 2.0


# Set boundary points at the ends of the domain
x_p[0]  = x_cv[0]                          # Left boundary
x_p[-1] = x_cv[-1]                         # Right boundary
y_p[0]  = y_cv[0]                          # Bottom boundary
y_p[-1] = y_cv[-1]                         # Top boundary

#=======================================
# Define the cylinder
#=======================================

# Cylinder definition for the plot

D = 2.0                                    # Diameter
x0 = L / 2.0                               # X position of the center of the cylinder
y0 = H / 2.0                               # Y position of the center of the cylinder

# Cylinder definition

points = 100
X = np.linspace (-D / 2.0, D / 2.0, points)
Y = np.zeros(points)
for i in range(points):
    Y[ i ] = np.sqrt((D / 2.0)**2 - X[ i ] ** 2)


#=======================================
# Define matrixes
#=======================================

psi       = np.zeros((N+2,M+2))            # Stream function matrix
psi_start = np.zeros((N+2,M+2))            # Start stream function
psi_ax    = np.zeros((N+2,M+2))                # Auxiliary matrix for G-S solver
rho       = np.zeros((N+2,M+2))            # Density matrix
a_P       = np.zeros((N+2,M+2))            # Auxiliar matrix at point p
a_E       = np.zeros((N+2,M+2))            # Auxiliar matrix at point east
a_S       = np.zeros((N+2,M+2))            # Auxiliar matrix at point south
a_W       = np.zeros((N+2,M+2))            # Auxiliar matrix at point west
a_N       = np.zeros((N+2,M+2))            # Auxiliar matrix at point north
b_P       = np.zeros((N+2,M+2))            # Generation term
```

```
conver    = np.zeros((N+2,M+2))           # Matrix that will be used in G-S algorithm to observe the convergence
I_body    = np.zeros((N+2,M+2))           # Solid or fluid indentificator matrix

#=======================================
# Compute I_body matrix
#=======================================

for i in range(N+2):
    for j in range(M+2):
        x_aux = x_p[i]                                          # Compute the p cell x position
        y_aux = y_p[j]                                          # Compute the p cell y position
        distance = np.sqrt((x_aux - x0)**2 + (y_aux - y0)**2)  # Compute the p cell r position
        if distance <= (D / 2.0):  # Is the r vector inside the cylinder?
            I_body[i,j] = 1        # yes ==> I_body = 1
        else:
            I_body[i,j] = 0        # no ==> I_body = 0


#=======================================
# Initialize stream function and
# density matrixes
#=======================================
rho_in = P_in / R * T_in
for i in range(N+2):
    for j in range(M+2):
        psi[i,j]       = psi_s
        psi_start[i,j] = psi_s
        rho[i,j]       = rho_in



#=======================================
# Evaluate internal nodes
#=======================================

rho_ref = rho_in                        # Set the reference value as the input (rho_ref/rho=1)
for i in range(1,N+1):
    for j in range(1,M+1):
        a_E[i,j] = (rho_ref / rho[i + 1,j]) * (delta_Y / np.abs(x_p[i] - x_p[i + 1]))
        a_W[i,j] = (rho_ref / rho[i - 1,j]) * (delta_Y / np.abs(x_p[i] - x_p[i - 1]))
        a_N[i,j] = (rho_ref / rho[i,j + 1]) * (delta_X / np.abs(y_p[j] - y_p[j + 1]))
        a_S[i,j] = (rho_ref / rho[i,j - 1]) * (delta_X / np.abs(y_p[j] - y_p[j - 1]))
        a_P[i,j] = a_E[i,j] + a_W[i,j] + a_N[i,j] + a_S[i,j]

for i in range(N+2):
    for j in range(M+2):
        psi[0,j]  = 0.0                 # Bottom nodes
        psi[-1,j] = V_in * H            # Top nodes
for i in range(N+2):
    for j in range(M+2):
        a_E[0,j]  = 0.0                 # Bottom nodes
        a_W[0,j]  = 0.0                 # Bottom nodes
        a_N[0,j]  = 0.0                 # Bottom nodes
        a_S[0,j]  = 0.0                 # Bottom nodes
        a_P[0,j]  = 1.0                 # Bottom nodes
```

```
        b_P[0,j]  = psi_B                # Bottom nodes
        a_E[-1,j] = 0.0                  # Top nodes
        a_W[-1,j] = 0.0                  # Top nodes
        a_N[-1,j] = 0.0                  # Top nodes
        a_S[-1,j] = 0.0                  # Top nodes
        a_P[-1,j] = 1.0                  # Top nodes
        b_P[-1,j] = psi_T                # Top nodes

#=======================================
# Inlet Flow
#=======================================

for j in range(1, M+1):
    psi[j,0] = V_in * y_p[j]
    b_P[j,0] = V_in * y_p[j]
    a_P[j,0] = 1.0
#=======================================
# Here we start the G-S algorithm
# loop since the outlet of psi must
# be also solved interatively, so that
# the stream function converge.
#=======================================

print("========================")
print("Starting G-S algorithm")
print("========================")
psi_ax = psi                             # Set the initial value of the axuliary stream
    function
for t in range(t_max):
    #=======================================
    # Outlet Flow
    #=======================================
    for j in range(1, M+1):
        psi[j,-1] = psi[j,-2]
        a_W[j,-1] = 1.0
        a_P[j,-1] = 1.0

    #=======================================
    # Compute G-S for non boundary points
    #=======================================
    r = np.sum(psi_ax)                       # Sum the values of the psi auxiliary matrix
    for i in range(1, N+1):
        for j in range(1, M+1):
            if I_body[i,j] == 1:             # Is the p cell inside the body?
                psi[i,j] = psi_body          # yes, psi=psi_body
            else:                            # no, psi= G-S alsogirthm
                psi[i, j] = (a_E[i, j] * psi_ax[i + 1, j] + a_W[i, j] * psi_ax[i - 1, j]
                    + a_N[i, j] * psi_ax[i, j + 1] + a_S[i,j] * psi_ax[i, j - 1] + b_P[i,
                     j]) / a_P[i, j]

    sum = np.sum(psi)                        # Sum the values of psi matrix
    if np.abs(sum-r) <= eps:                 # Watch if the |psi - psi_aux| < precision
        print("========================")
        print("G-S algorithm converged")
        print("The requiered steps has been:" +  " " + str(t))
        print("========================")
        break                                # If corveges just break the main G-S loop
    else:
        psi_ax = psi                         # If it does not converge, reset the valu of
             psi_aux and start again the loop
```

```
205
206
207 #=======================================
208 # Compute the Velocity
209 #=======================================
210
211 # Define the requiered velocity matrix
212 v_xn = np.zeros((N+2,M+2))
213 v_xs = np.zeros((N+2,M+2))
214 v_ye = np.zeros((N+2,M+2))
215 v_yw = np.zeros((N+2,M+2))
216 v_xP = np.zeros((N+2,M+2))
217 v_yP = np.zeros((N+2,M+2))
218 v_P  = np.zeros((N+2,M+2))
219
220 for i in range(N+2):
221     for j in range(M+2):
222         v_xP[0,j]  = V_in             # Bottom nodes
223         v_yP[0,j]  = 0.0              # Bottom nodes
224         v_xP[-1,j] = V_in             # Top nodes
225         v_yP[-1,j] = 0.0              # Top nodes
226
227 # Inlet flow
228 for j in range(1,M+1):
229     v_xP[j,0] = V_in
230     v_yP[j,0] = 0.0
231
232 # Fill the velocities
233 for i in range(1,N+1):
234     for j in range(1,M+1):
235         if I_body[i,j] == 1:
236                 v_xP[i,j] = 0.0
237                 v_yP[i,j] = 0.0
238         else:
239             v_xn[i,j] = (rho_ref / rho[i,j + 1]) * ((psi[i + 1,j] - psi[i,j]) / np.abs(
                    y_p[j] - y_p[j + 1]))
240             v_xs[i,j] = -(rho_ref / rho[i,j - 1]) * ((psi[i - 1,j] - psi[i,j]) / np.abs(
                    y_p[j] - y_p[j - 1]))
241             v_ye[i,j] = -(rho_ref / rho[i + 1,j]) * ((psi[i,j +1] - psi[i,j]) / np.abs(
                    x_p[i] - x_p[i + 1]))
242             v_yw[i,j] = (rho_ref / rho[i - 1,j]) * ((psi[i,j -1] - psi[i,j]) / np.abs(x_p
                    [i] - x_p[i - 1]))
243             v_xP[i,j] = (v_xn[i,j] + v_xs[i,j]) / 2.0
244             v_yP[i,j] = (v_ye[i,j] + v_yw[i,j]) / 2.0
245             v_P[i,j]  = np.sqrt(v_xP[i,j]**2 + v_yP[i,j]**2)
246
247 # Outlet flow
248 for j in range(1, M+1):
249         v_xP[j,-1] = v_xP[j,-2]
250         v_yP[j,-1] = v_yP[j,-2]
251
252 #=======================================
253 # Compute the Temperature
254 #=======================================
255
256 # Input data
257 T_ref = T_in
258 V_ref = V_in
259 cp    = 1005 # J / Kg K
260
```

```
# Define temperature vector
T_P   = np.zeros((N+2, M+2))

# Compute temperature values
for i in range(N+2):
    for j in range(M+2):
        T_P[i,j] = T_ref + (V_ref**2 - v_P[i,j]**2) / (2.0 * cp)


#=======================================
# Compute the Pressure
#=======================================

# Physical values
P_ref = P_in
gamma = 1.4
P_P   = np.zeros((N+2, M+2))

for i in range(N+2):
    for j in range(M+2):
        P_P[i,j] = P_ref * (T_P[i,j] / T_ref) ** (gamma / (gamma - 1))


#=======================================
# Compute Lift and Dragg
#=======================================

L_plus = 0.0
L_minus = 0.0
D_plus = 0.0
D_minus=0.0

# Compute Lift
for i in range(1, N+1):
    for j in range(1, M+1):
        if I_body[i,j] == 1 and I_body[i + 1,j] == 0:
            L_plus = L_plus - P_P[i + 1,j] * delta_X

        elif I_body[i,j] ==1 and I_body[i - 1, j] == 0:
            L_minus = L_minus + P_P[i - 1,j] * delta_X


# Compute Lift
for i in range(1, N+1):
    for j in range(1, M+1):
        if I_body[i,j] == 1 and I_body[i,j + 1] == 0:
            D_plus = D_plus - P_P[i,j + 1] * delta_Y
        elif I_body[i,j] ==1 and I_body[i, j - 1] == 0:
            D_minus = D_minus + P_P[i,j - 1] * delta_Y

L = np.abs(L_minus + L_plus)
D = np.abs(D_minus + D_plus)

# Write the results forces
forces_file = f"Results/Static/ForcesS_{N}x{M}.txt"
with open(forces_file, 'w') as file:
    file.write("=================\n")
    file.write(f' The total lift over the cylinder is: L = {L} N\n')
    file.write(" \n")
    file.write("=================\n")
```

```
    file.write(f' The total dragg over the cylinder is: D = {D} N\n')

#=======================================
# Compute The Circulation
#=======================================

circ = 0.0

# Compute Circulation
for i in range(1, N+1):
    for j in range(1, M+1):
        if I_body[i,j] == 1 and I_body[i + 1,j] == 0:
            circ = circ - v_xs[i + 1,j] * delta_X

        elif I_body[i,j] ==1 and I_body[i - 1, j] == 0:
            circ = circ + v_xn[i - 1,j] * delta_X


# Compute Circulation
for i in range(1, N+1):
    for j in range(1, M+1):
        if I_body[i,j] == 1 and I_body[i,j + 1] == 0:
            circ = circ - v_ye[i,j + 1] * delta_Y
        elif I_body[i,j] ==1 and I_body[i, j - 1] == 0:
            circ = circ + v_yw[i,j - 1] * delta_Y

circ_abs = np.abs(circ)

# Write the results forces
circulation_file = f"Results/Static/CirculationS_{N}x{M}.txt"
with open(circulation_file, 'w') as file:
    file.write("=================\n")
    file.write(f' The circulation over the cylinder is {circ_abs} m^2 / s')

#=======================================
# Generate the plot
#=======================================
output_plot1 = f"Results/Static/Static_SL_{N}x{M}.pdf"
output_plot2 = f"Results/Static/Static_Vel_{N}x{M}.pdf"
output_plot3 = f"Results/Static/Static_Temp_{N}x{M}.pdf"
output_plot4 = f"Results/Static/Static_Pres_{N}x{M}.pdf"
print(" ")
print("=================")
print("Starting the plot")
print(" ")
print("It might take a few seconds")
print("===========================")

# Figure specifications
fontsize=15

# Start first plot
plt.figure(1)
plt.figure(figsize = (8,8))
plt.tick_params(axis='both', which='both',length=3, width=1.0,
labelsize=15, right=True, top=True, direction='in') # For ticks in borders

# Figure labels
plt.xlabel(r"$X(m)$", fontsize=fontsize)
plt.ylabel(r"$Y(m)$", fontsize=fontsize)
```

```
381
382 # Plot
383 plt.contour(x_p, y_p, psi, levels=60, colors='blue', linewidths=1.0)
384
385 # Cylinder plot
386 plt.plot(X + x0, Y + y0, color = "black")
387 plt.plot(X + x0,-Y + y0, color = "black")
388 plt.fill_between(X + x0, Y + y0, -Y + y0, color="black")
389
390 # Save figure
391 plt.savefig(output_plot1,bbox_inches='tight')
392
393 # Start second plot
394 plt.figure(2)
395 plt.figure(figsize = (8,8))
396 plt.tick_params(axis='both', which='both',length=3, width=1.0,
397 labelsize=15, right=True, top=True, direction='in') # For ticks in borders
398
399 # Figure labels
400 plt.xlabel(r"$X(m)$", fontsize=fontsize)
401 plt.ylabel(r"$Y(m)$", fontsize=fontsize)
402
403 # Plot
404 P,Z = np.meshgrid(x_p, y_p)      # We generate the mesh points (for plotting, not
        computing)
405 plt.quiver(P[::2, ::2], Z[::2, ::2], v_xP[::2, ::2], v_yP[::2, ::2], color="green")
406
407 # Cylinder plot
408 plt.plot(X + x0, Y + y0, color = "black")
409 plt.plot(X + x0,-Y + y0, color = "black")
410 plt.fill_between(X + x0, Y + y0, -Y + y0, color="black")
411
412 # Save figure
413 plt.savefig(output_plot2,bbox_inches='tight')
414
415 # Start third plot
416 plt.figure(3)
417 plt.figure(figsize = (8,8))
418 plt.tick_params(axis='both', which='both',length=3, width=1.0,
419 labelsize=15, right=True, top=True, direction='in') # For ticks in borders
420
421 # Figure labels
422 plt.xlabel(r"$X(m)$", fontsize=fontsize)
423 plt.ylabel(r"$Y(m)$", fontsize=fontsize)
424
425 # Plot
426 plt.imshow(T_P, cmap= 'inferno', extent=(x_p.min(), x_p.max(), y_p.min(), y_p.max()),
        origin='lower')
427 plt.colorbar()
428
429 # Get the current axis
430 ax = plt.gca()
431
432 # Add text next to the color bar
433 text_x = 1.25  # Adjust the x-coordinate as needed
434 text_y = 0.5   # Adjust the y-coordinate as needed
435 text = r"$T~(K)$"
436 ax.text(text_x, text_y, text, transform=ax.transAxes, rotation=270, va='center', fontsize
        =fontsize)
437
```

```
# Cylinder plot
plt.plot(X + x0, Y + y0, color = "black")
plt.plot(X + x0,-Y + y0, color = "black")
plt.fill_between(X + x0, Y + y0, -Y + y0, color="black")

# Save figure
plt.savefig(output_plot3,bbox_inches='tight')

plt.figure(4)
plt.figure(figsize = (8,8))
plt.tick_params(axis='both', which='both',length=3, width=1.0,
labelsize=15, right=True, top=True, direction='in') # For ticks in borders

# Figure labels
plt.xlabel(r"$X(m)$", fontsize=fontsize)
plt.ylabel(r"$Y(m)$", fontsize=fontsize)

# Plot
plt.imshow(P_P, cmap= 'viridis', extent=(x_p.min(), x_p.max(), y_p.min(), y_p.max()),
    origin='lower')
plt.colorbar()

# Get the current axis
ax = plt.gca()

# Add text next to the color bar
text_x = 1.25  # Adjust the x-coordinate as needed
text_y = 0.5   # Adjust the y-coordinate as needed
text = r"$P~(Pa)$"
ax.text(text_x, text_y, text, transform=ax.transAxes, rotation=270, va='center', fontsize
    =fontsize)

# Cylinder plot
plt.plot(X + x0, Y + y0, color = "black")
plt.plot(X + x0,-Y + y0, color = "black")
plt.fill_between(X + x0, Y + y0, -Y + y0, color="black")

# Save figure
plt.savefig(output_plot4,bbox_inches='tight')


print("Plots has been done sucesfully!")
print(" ")
print("You should find them on:" + " " + str(output_plot1) +  " " + "and" +  " " + str(
    output_plot2) + " " + "and" + " " + str(output_plot3) + " " + "and" + " " + str(
    output_plot4))
print("==============================")


plt.close(1)
plt.close(2)
plt.close(3)
plt.close(4)
```