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

Computational Engineering
Master's degree in Space and Aeronautical Engineering
Universitat Politècnica de Catalunya

Assignment 1 (CFD)

Non-viscous potential flows

González Martínez, Víctor

Contents

1 Problem Specification

Understanding how a fluid moves around a solid body is one of the central problems in aerodynamics. Among all possible configurations, the case of a circular cylinder has become a classic benchmark for studying potential flow. Even though it represents a simplified and ideal scenario, it allows us to explore in detail the main aerodynamic principles that explain how lift and circulation are generated when a body interacts with a moving fluid.

The purpose of this project is to simulate the two-dimensional, steady and inviscid flow around a circular cylinder using a numerical model based on the **streamfunction formulation**. This representation is especially useful because it automatically satisfies the mass conservation condition and provides a clear view of the flow pattern through its isolines. The flow is considered **incompressible and irrotational**, so viscous and turbulent effects are neglected, focusing solely on the pressure-driven behaviour of the fluid. Under these assumptions, the governing equations reduce to a Laplace-type system that can be solved iteratively to obtain the potential flow field.

The computational domain is designed as a rectangular channel with a circular cylinder located at its centre. A uniform velocity is imposed at the inlet, representing the freestream conditions, while the outlet boundary applies a zero-gradient condition to allow the flow to leave the domain smoothly. The upper and lower walls maintain a constant streamfunction value to ensure continuity of the global flow rate. On the cylinder surface, the streamfunction is adjusted to represent either a stationary or a rotating boundary, depending on the case under study.

Three configurations are examined in order to understand the influence of rotation on the flow field and on the aerodynamic forces acting on the cylinder:

1. **Static cylinder**, used as a reference case, where no rotation or circulation is introduced and no lift is expected, as predicted by d'Alembert's paradox.
2. **Clockwise rotation**, in which the induced circulation alters the symmetry of the flow and creates a downward lift due to pressure differences between the upper and lower sides.
3. **counter-clockwise rotation**, producing an opposite effect: the pressure distribution reverses, leading to an upward lift force.

By comparing these three cases, the relationship between the **circulation** around the body and the resulting **lift coefficient** can be evaluated, and the numerical solution can be validated against theoretical models. The simulation provides information on several key aspects of the flow, including the **streamfunction**, **pressure**, and **temperature fields**, as well as the **aerodynamic coefficients** derived from them.

Ultimately, this analysis aims to show how even a simple inviscid model can reproduce the fundamental aerodynamic phenomena associated with circulation, offering a solid foundation for understanding more complex flow situations, such as those counter-clockwised around airfoils or rotating machinery.

2 Numerical Methodology

The analysis of potential flow around a circular cylinder has been carried out through a numerical model developed in **MATLAB**, based on the **streamfunction formulation**. This approach is particularly suitable for representing two-dimensional, steady, incompressible, and inviscid flows, since it inherently satisfies the mass conservation equation and provides a direct visualization of the flow field through its streamlines.

The inviscid assumption implies that viscous effects are neglected and that the motion of the fluid is governed exclusively by pressure gradients. Under these conditions, the velocity field can be directly derived from the streamfunction, and pressure is obtained from the energy and isentropic relations.

The computational procedure is divided into several stages: mesh generation, discretization of the flow field, imposition of boundary conditions, iterative solution of the streamfunction, and post-processing to obtain the pressure, temperature, circulation, and aerodynamic forces.

2.1 Discretization and Solution of the Streamfunction Equation

The streamfunction field ψ is obtained by solving an elliptic equation of Laplace type, which describes the equilibrium of the potential flow in the absence of sources or sinks. The discrete form used in the code is:

$$a_P \psi_P = a_E \psi_E + a_W \psi_W + a_N \psi_N + a_S \psi_S + b_P \tag{2.1.1}$$

where the subscripts E, W, N, S refer to the neighboring control volumes in the east, west, north, and south directions. The influence coefficients are computed as:

$$a_N = \rho_N \frac{\Delta x}{\Delta y_N}, \quad a_S = \rho_S \frac{\Delta x}{\Delta y_S}, \quad a_E = \rho_E \frac{\Delta y}{\Delta x_E}, \quad a_W = \rho_W \frac{\Delta y}{\Delta x_W}, \tag{2.1.2}$$

and the central coefficient is given by:

$$a_P = a_E + a_W + a_N + a_S \tag{2.1.3}$$

The term b_P represents a possible source term, which in this case is set to zero since the flow is purely potential. To improve numerical stability, the densities at the cell faces $(\rho_N, \rho_S, \rho_E, \rho_W)$ are evaluated using **harmonic means** between adjacent cells, preventing spurious oscillations and ensuring smooth transitions.

The resulting algebraic system is solved iteratively using the **Gauss–Seidel method**, which updates the streamfunction values successively until convergence is achieved. This iterative scheme is particularly effective for steady-state elliptic equations and provides a stable and monotonic convergence pattern under well-defined boundary conditions.

2.2 Boundary Conditions

The computational domain consists of a rectangular channel containing a circular cylinder located at its center. The boundaries are defined as follows:

- **Inlet:** A uniform velocity profile is imposed through a linear streamfunction:

$$\psi = V_\infty y \tag{2.2.1}$$

  representing a constant horizontal inflow and ensuring a uniform mass flow rate across the channel height.

- **Outlet:** A zero-gradient condition ($\partial\psi/\partial x = 0$) is applied to allow the flow to exit the domain smoothly, avoiding artificial reflections.

- **Upper and lower walls:** Constant streamfunction values are imposed, preserving the total mass flow and preventing fluid penetration through the solid boundaries.

- **Cylinder surface:** The solid body is represented through a logical mask that cancels the density values within the solid, thus preventing flow inside the cylinder. For the static case, the streamfunction remains constant along the cylinder surface. In the rotating cases, an internal correction factor is applied to the streamfunction to simulate the rotational motion:

$$\psi_{\text{body}} = \left(\frac{\psi_{\text{bottom}} + \psi_{\text{top}}}{2}\right) \cdot \text{factor} \tag{2.2.2}$$

  where the factor is 0.5 for clockwise rotation and 1.5 for counter-clockwise rotation. This adjustment introduces circulation in the corresponding direction, breaking the flow symmetry and reproducing the rotational influence.

2.3 Velocity Field Calculation (Mass flow)

Once the streamfunction field has converged, the velocity components are obtained from spatial derivatives of ψ. In the code, the velocities are computed at the faces of each control volume using centered finite differences:

$$u_N = \rho_N \frac{\psi_{i,j+1} - \psi_{i,j}}{\Delta y_N}, \quad u_S = \rho_S \frac{\psi_{i,j} - \psi_{i,j-1}}{\Delta y_S} \tag{2.3.1}$$

$$v_E = \rho_E \frac{\psi_{i,j} - \psi_{i+1,j}}{\Delta x_E}, \quad v_W = \rho_W \frac{\psi_{i-1,j} - \psi_{i,j}}{\Delta x_W} \tag{2.3.2}$$

The cell-centered velocities are computed by averaging the opposite face values:

$$u_P = \frac{u_N + u_S}{2}, \qquad v_P = \frac{v_E + v_W}{2} \tag{2.3.3}$$

and the velocity magnitude is given by:

$$V_P = \sqrt{u_P^2 + v_P^2} \tag{2.3.4}$$

This procedure allows reconstructing the complete velocity field, enabling the analysis of flow patterns and the influence of cylinder rotation on the overall distribution.

2.4 Pressure and Temperature Field Calculation

The pressure field is derived from the energy balance of the flow. Under the inviscid and isentropic assumptions, the local temperature is computed from a simplified Bernoulli relation:

$$T_P = T_0 + \frac{V_\infty^2 - V_P^2}{2c_p} \tag{2.4.1}$$

and the pressure is obtained using the isentropic relation between pressure and temperature:

$$P_P = P_0 \left(\frac{T_P}{T_0}\right)^{\frac{\gamma}{\gamma-1}} \tag{2.4.2}$$

These expressions allow the estimation of pressure and temperature distributions without solving the full energy equation, capturing the variations caused by the rotational motion and the corresponding circulation around the body.

2.5 Aerodynamic Forces and Circulation Calculation

The aerodynamic forces on the cylinder are evaluated by integrating the pressure distribution around the solid surface. In the code, the drag and lift forces per unit span are computed as:

$$D' = \sum(-P_{i+1,j} + P_{i-1,j})\,\Delta y, \qquad L' = \sum(-P_{i,j+1} + P_{i,j-1})\,\Delta x \tag{2.5.1}$$

Thus, the drag and the lift are obtained directly from the pressure differences along the streamwise direction.

The total circulation around the cylinder is obtained by summing the tangential velocity contributions along the cells adjacent to the solid boundary:

$$\Gamma = \sum(\tau_N + \tau_S + \tau_E + \tau_W) \tag{2.5.2}$$

where each τ term represents the tangential flux through a given face. This quantity is essential for quantifying the effect of rotation on the lift force and validating the consistency of the numerical model.

In summary, the implemented methodology provides an accurate representation of the potential flow behavior around a rotating cylinder.

3 Code Structure

The numerical code developed for this project was entirely implemented in **MATLAB**, following a modular and well-documented structure that facilitates both interpretation and modification. It is designed to simulate the **two-dimensional potential flow** around a circular cylinder under both static and rotational conditions, using a numerical formulation based on the *streamfunction*.

The program is divided into five main blocks:

1. Initialization of physical and numerical parameters,
2. Computation of flow properties and discrete coefficients,
3. Iterative resolution using the Gauss–Seidel method,
4. Post-processing of physical magnitudes,
5. Graphical representation and output of results.

Each block performs a specific task while maintaining consistency between the physical, geometric, and numerical variables, ensuring a logical computational flow and a clear interpretation of the results.

3.1 Initialization and Grid Generation

In the first section, the **physical parameters** of the problem are defined, including the channel dimensions (L, H), cylinder diameter (D), and flow properties $(V_\infty, P_\infty, T_\infty, \rho_\infty, c_p, \gamma)$. The **numerical parameters** are also set, such as the number of control volumes (N, M) and the corresponding cell size $(\Delta x, \Delta y)$.

Matrices are initialized to store the main field variables: the streamfunction ψ, pressure P, temperature T, density ρ, and velocity components u and v. To properly handle boundary conditions, all arrays include two layers of **ghost cells**, which simplify the application of Dirichlet and Neumann conditions without affecting interior nodes.

The geometry of the cylinder is represented using a **logical mask** that identifies solid cells within the computational domain. Each node satisfying $(x - L/2)^2 + (y - H/2)^2 \leq (D/2)^2$ is marked as solid, and its density is set to zero. This approach avoids complex body-fitted meshes and maintains a simple Cartesian grid structure.

3.2 Computation of Discrete Coefficients

Once the grid is defined, the code computes the **discretization coefficients** of the streamfunction equation. The coefficients a_E, a_W, a_N, and a_S are obtained from the face densities and the geometric distances between nodes, while the central coefficient is given by:

$$a_P = a_E + a_W + a_N + a_S$$

The face densities $(\rho_E, \rho_W, \rho_N, \rho_S)$ are computed using the **harmonic mean** between adjacent cells, improving numerical stability and ensuring smooth transitions. In solid regions, all density values are set to zero to prevent any mass flux through the body surface. Coefficients are also updated in ghost layers to ensure consistent boundary conditions throughout the computational domain.

3.3 Iterative Resolution (Gauss–Seidel Solver)

The core of the program is an iterative loop that applies the **Gauss–Seidel method** to solve the discretized streamfunction equation. At each iteration, the boundary conditions are enforced, and ψ is updated in all fluid cells according to the finite difference expressions.

Cells inside the cylinder are assigned a fixed value of ψ based on a **rotation factor** that simulates solid-body motion. Three configurations are considered:

- Case 1: static cylinder (factor = 1),
- Case 2: clockwise rotation (factor = 0.5),
- Case 3: counter-clockwise rotation (factor = 1.5).

This approach introduces controlled circulation into the flow while maintaining the same mesh and discretization scheme. At each iteration, the face velocities are computed from the spatial derivatives of ψ, and the thermodynamic variables (pressure and temperature) are updated using Bernoulli and isentropic relations. The iterative process continues until convergence or until the maximum number of iterations is reached.

3.4 Post-Processing and Physical Quantities

After convergence, the post-processing stage extracts the main physical quantities of interest. Velocity components are combined to obtain the magnitude field, and from this, the local pressure and temperature are derived.

The total **circulation** Γ is evaluated as the sum of tangential velocity contributions at the solid boundary. The aerodynamic forces acting on the cylinder are then calculated as:

$$D' = \sum(-P_{i+1,j} + P_{i-1,j})\,\Delta y, \qquad L' = \sum(-P_{i,j+1} + P_{i,j-1})\,\Delta x$$

The **drag** and the **lift** are obtained by pressure integration.

3.5 Visualization and Output

Finally, the code generates several **graphical outputs** that display:

- Streamlines,
- Pressure distribution,
- Temperature distribution.

Each figure shows the entire domain and the cylinder outline, using color maps and filled contour plots to clearly represent the flow behavior. Additionally, the MATLAB console outputs the final values of circulation, drag and lift providing a concise summary of the simulation results.

The modular design of the code, together with the clear separation between computation and visualization, makes it easily extensible to more complex geometries or flow conditions, while maintaining transparency and numerical stability.

4 Results and Code Verification

4.1 Streamlines

First, the streamlines obtained for the three studied cases are analyzed: the static cylinder (Fig. 1), the clockwise rotating cylinder (Fig. 2), and the counter-clockwise rotating cylinder (Fig. 3).

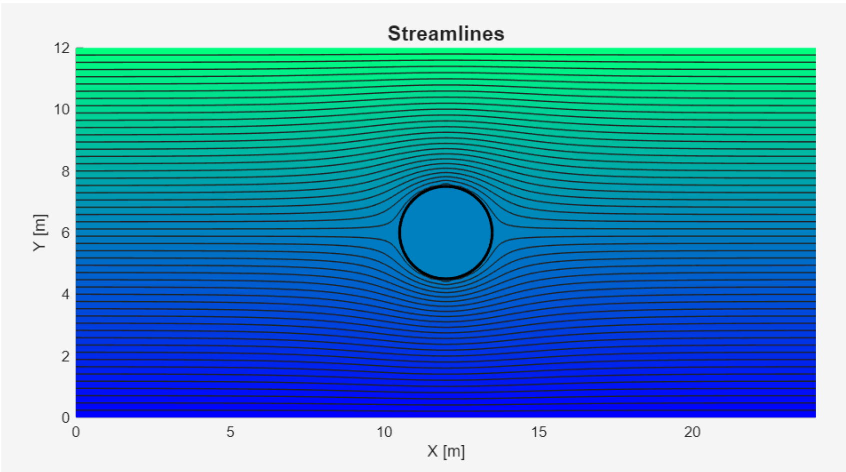

Figure 1: Streamlines static case.

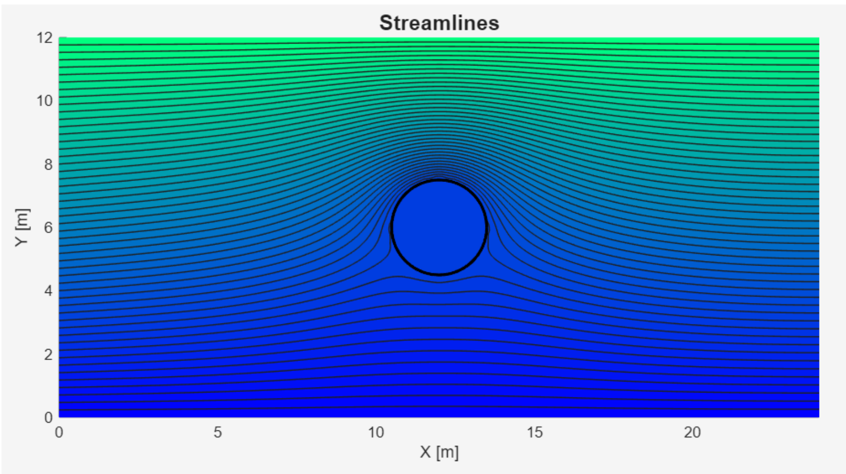

Figure 2: Streamlines clockwise case.

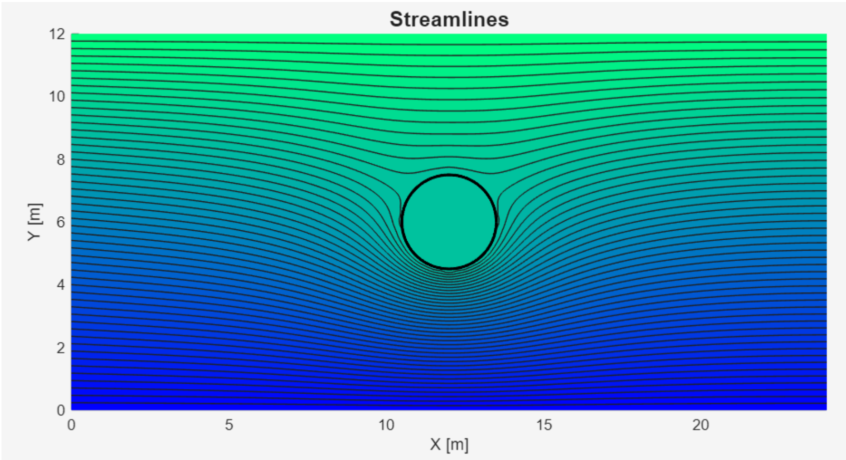

Figure 3: Streamlines counter-clockwise case.

In the **static case**, the streamlines show a perfectly symmetric distribution with respect to the horizontal axis passing through the cylinder's center. The flow splits evenly between the upper and lower sides, with no indication of circulation or net lift. This result is consistent with the theory of ideal potential flow, where a symmetric body without rotation experiences no lift or drag force (d'Alembert's paradox).

In the **clockwise case**, the streamlines are deflected towards the lower part of the cylinder, indicating higher velocity over the upper surface and lower velocity underneath. This asymmetry produces a pressure decrease in the upper region and consequently a downward lift force, as expected for positive circulation according to the adopted convention.

Conversely, in the **counter-clockwise case**, the streamlines rise above the upper side of the cylinder, showing an opposite deflection. Here, the pressure decreases below the cylinder and increases above it, producing a negative lift, consistent with the negative circulation generated by the opposite rotation.

The comparison of the three streamline patterns demonstrates that the code correctly reproduces the expected physical effects: symmetry for the static case and antisymmetric behavior when reversing the rotation direction. This provides visual confirmation of the validity of the numerical model and the correct implementation of the boundary conditions.

4.2 Aerodynamic Parameters

The numerical results collected in Table 4.2.1 reinforce the previous observations.

	Circulation	**Drag**	**Lift**
Static	0	0	0
Clockwise	65.45	-2.9	848.27
Counter-clockwise	-65.46	-2.75	-848.45

Table 4.2.1: Aerodynamic Parameters.

In the static configuration, the values of circulation, lift, and drag are practically zero, confirming the accurate resolution of the potential field.

In the clockwise case, a positive circulation and lift are obtained, consistent with the downward deflection of the streamlines and the pressure difference between the upper and lower sides of the cylinder.

In contrast, the counter-clockwise case produces negative circulation and also negative lift, confirming that the numerical model preserves the physical symmetry between both rotation directions. In all scenarios, the drag values remain small, as expected for an inviscid model.

These results confirm that the code solves the potential flow equations in a stable and coherent way, accurately reproducing the theoretical relationships between circulation, pressure, and aerodynamic forces.

4.3 Pressure and Temperature Distribution

The pressure distribution around the cylinder further supports the conclusions above:

- In the static case (Fig. 4), the pressure field is symmetric, with two low-pressure zones located at the upper and lower stagnation points.
- In the clockwise case (Fig. 5), the pressure significantly decreases over the upper region of the cylinder, generating a downward lift force.
- In the counter-clockwise case (Fig. 6), the lowest pressure occurs below the cylinder, reversing the lift direction compared to the previous configuration.

Regarding the temperature field, for both the clockwise (Fig. 8) and counter-clockwise (Fig. 9) cases, the region of minimum temperature spatially coincides with the area of minimum pressure, in accordance with the thermodynamic relationship between static pressure and the kinetic energy of the flow. For the static case (Fig. 7), the temperature field is symmetric with two low-temperature zones located at the upper and lower stagnation points.

Overall, both the visual and numerical results confirm that the developed code is correctly implemented and physically consistent, successfully reproducing the theoretical behavior of potential flow around a cylinder in both static and rotating conditions.

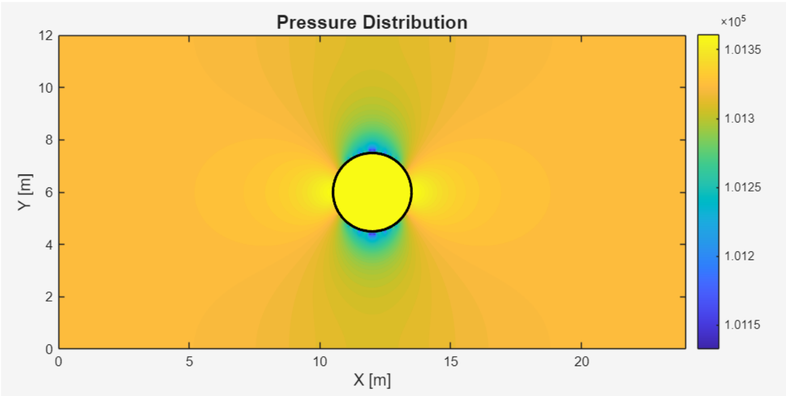

Figure 4: Pressure distribution static case.

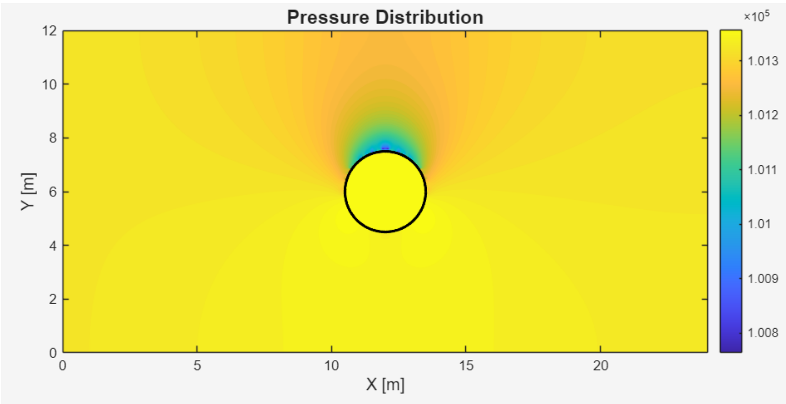

Figure 5: Pressure distribution clockwise case.

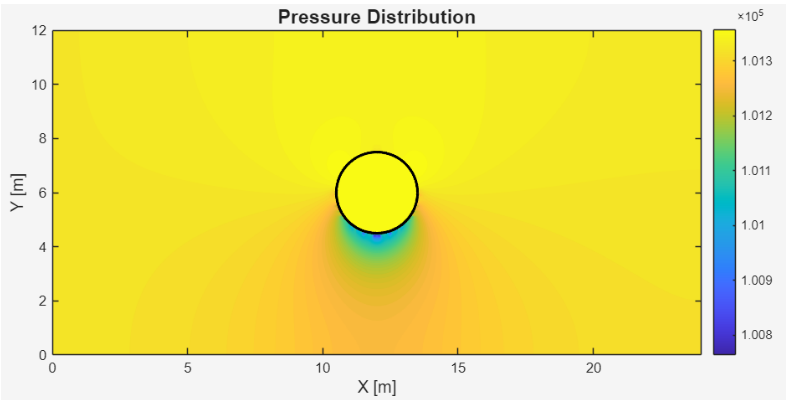

Figure 6: Pressure distribution counter-clockwise case.

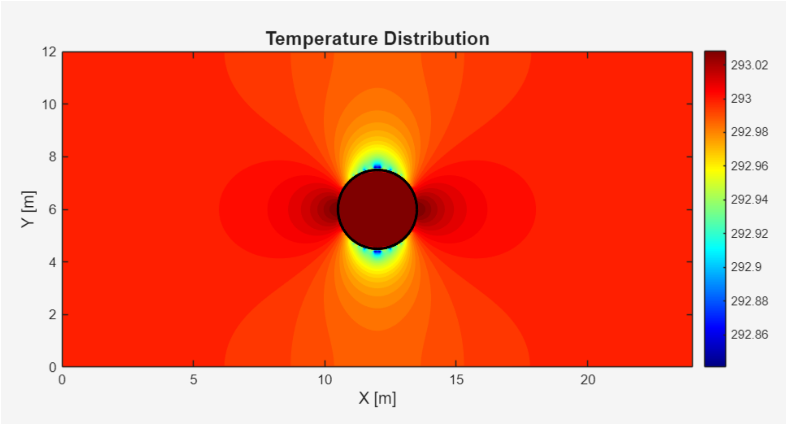

Figure 7: Temperature distribution static case.

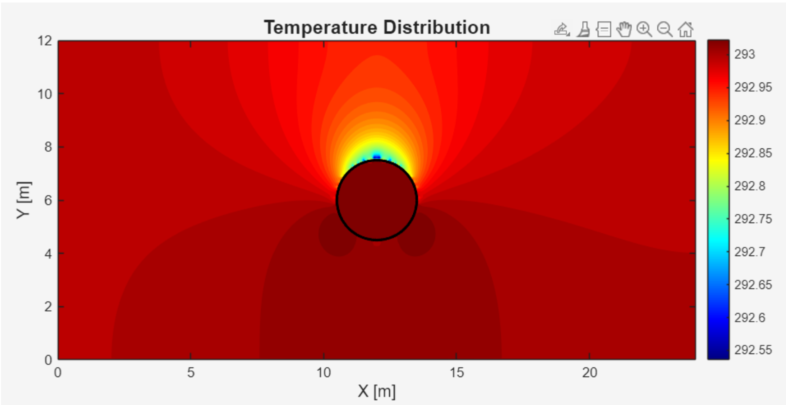

Figure 8: Temperature distribution clockwise case.

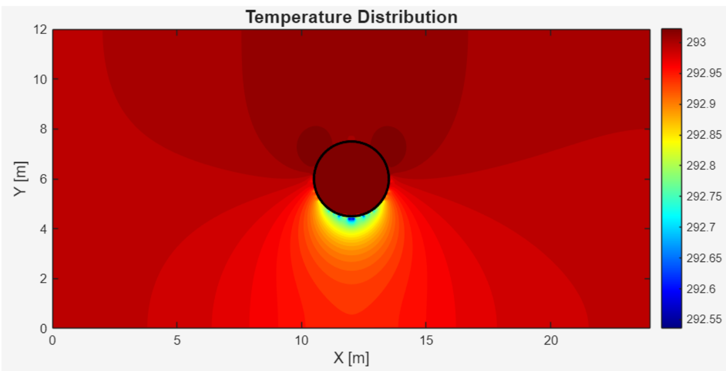

Figure 9: Temperature distribution counter-clockwise case.

A Appendix

Matlab code used:

```
clc;
clear;
close all;

%% ASSIGNMENT 1: Víctor González Martínez
%% ESEIAAT MSc in Space and Aeronautical Engineering

% Problem geometry and reference flow properties
L =    24;                % Channel length [m]
H =    12;                 % Channel height [m]
D =    3;               % Cylinder diameter [m]
V_in = 8;                % Inlet (free-stream) velocity [m/s]
P_in = 101325;            % Reference pressure (used as initial/isentropic base) [Pa]
T_in = 293;               % Reference temperature (initial/static) [K]
rho_in = 1.20;            % Reference density [kg/m^3]
cp = 1006;                % Specific heat at constant pressure [J/kg·K]
R = 287;                  % Gas constant for air [J/kg·K]
gammaAir  = 1.4;          % Heat capacity ratio (air)

% Grid resolution and solver controls
N = 241;                  % Number of control volumes in X
M = 121;                  % Number of control volumes in Y
n_it = 20000;             % Number of Gauss{Seidel iterations for psi
dx = L/N;                 % Cell size in X
dy = H/M;                 % Cell size in Y

% Case selector for cylinder state (affects psi inside solid)
case_selector = 1; % 1 (Static); 2 (Clockwise rotation); 3 (Counter clockwise rotation)

% Utility arrays: preallocation prevents dynamic resizing inside loops
mat0_NM   = zeros(N,   M);     % Zero matrix for interior-coefficient storage
mat0_NM2 = zeros(N+2, M+2);   % Zero matrix with 1-cell ghost layer
mat1_NM2 = ones (N+2, M+2);   % Ones matrix with ghost layer
vect0_N2 = zeros(N+2, 1);     % Zero column vector (N+2)
vect0_M2 = zeros(1,   M+2);   % Zero row vector    (M+2)

```

```matlab
% Primary fields and coefficient storage (on extended grid with ghosts)
psi    = mat0_NM2;            % Streamfunction unknown (solved by Gauss{Seidel)
a_p    = mat0_NM2;            % Central coefficient of discretized psi-equation
b_p    = mat0_NM2;            % Source term (here initialized to zero)
psi_0 = mat0_NM2;             % Previous-iteration psi (initial guess = zeros)
T_p    = mat0_NM2;            % Temperature field (post-processed from velocity magnitude)
P_p    = mat0_NM2;            % Pressure field (from isentropic relation using T)
celltype = mat1_NM2;          % Domain mask: 1 = fluid cell, 0 = solid (cylinder)

[a_w, a_s, a_e, a_n] = deal(mat0_NM); % Discrete stencil coefficients W/S/E/N (interior size)

% Half distances (center-to-face) used in harmonic averaging
[d_Pw, d_Pe] = deal(vect0_N2); % Distances in X to West/East faces per i
[d_Pn, d_Ps] = deal(vect0_M2); % Distances in Y to North/South faces per j

% Initial thermodynamic fields (uniform)
rho = mat1_NM2 * rho_in;     % Density field (uniform initialization)
T   = mat1_NM2 * T_in;       % Temperature field (uniform)
P   = mat1_NM2 * P_in;       % Pressure field (uniform)
T_0 = T;                     % Save base T for post-processing (Bernoulli-like)
P_0 = P;                     % Save base P for isentropic relation

% Face densities (to be filled via harmonic mean)
[rho_w, rho_n, rho_s, rho_e] = deal(mat1_NM2); % W/N/S/E face densities

% Face mass-flux velocities from psi (u and v on faces)
[uN, vE, uS, vW] = deal(mat0_NM2);      % u on N/S, v on E/W faces

% Circulation contributions on faces adjacent to the body
[tauN, tauE, tauS, tauW] = deal(mat0_NM2);


% GRID GENERATION (uniform)
i = linspace(0, N-1, N); % Cell-center index array in X
j = linspace(0, M-1, M); % Cell-center index array in Y

xC = (i + 0.5) * dx;      % Cell-center coordinates in X
yC = (j + 0.5) * dy;      % Cell-center coordinates in Y

xEdge = [0, xC, L];       % Cell-edge coordinates in X (including domain edges)
yEdge = [0, yC, H];       % Cell-edge coordinates in Y (including domain edges)

% CYLINDER MASK (type_cell)
% Marks as solid (0) those indices whose (x,y) fall inside the cylinder.
for i = 1:N+2
```

```
    for j = 1:M+2
        x = xEdge(1,i);             % Using edge-based coordinates to build mask
        y = yEdge(1,j);
        if (x - L/2)^2 + (y - H/2)^2 <= (D/2)^2
            celltype(i,j) = 0;       % 0 => solid cell
            rho(i,j) = 0;            % Remove density in solid (no flow)
        end
    end
end

psi(:,:) = psi_0; % Initialize psi with initial guess (zeros)

% GEOMETRIC HALF-DISTANCES (center-to-face)
% These distances are used to compute harmonic-mean face properties.
for i = 1:N+2
    for j = 1:M+2
        % Y-direction distances (North/South)
        if j <= M+1
            d_Pn(1,j) = yEdge(1,j+1) - yEdge(1,j);   % full spacing to the North edge
            d_Pn(1,j) = d_Pn(1,j) / 2;               % half distance (center -> N face)
            d_Nn(1,j) = d_Pn(1,j);                   % alias for clarity
        end
        if j >= 2
            d_Ps(1,j) = yEdge(1,j) - yEdge(1,j-1);   % full spacing to the South edge
            d_Ps(1,j) = d_Ps(1,j) / 2;               % half distance (center -> S face)
            d_Ss(1,j) = d_Ps(1,j);
        end
        % X-direction distances (East/West)
        if i <= N+1
            d_Pe(i,1) = xEdge(1,i+1) - xEdge(1,i);   % full spacing to the East edge
            d_Pe(i,1) = d_Pe(i,1) / 2;               % half distance (center -> E face)
            d_Ee(i,1) = d_Pe(i,1);
        end
        if i >= 2
            d_Pw(i,1) = xEdge(1,i) - xEdge(1,i-1);   % full spacing to the West edge
            d_Pw(i,1) = d_Pw(i,1) / 2;               % half distance (center -> W face)
            d_Ww(i,1) = d_Pw(i,1);
        end
    end
end

% HARMONIC-MEAN FACE DENSITIES AND COEFFICIENTS
% Use harmonic averaging for face densities (robust for diffusive operators).
```

```
for i = 2:N+1
    for j = 2:M+1
        if (celltype(i,j) == 1) % fluid cell
            % Face densities via harmonic mean in each direction
            rho_n(i,j) = d_Pn(1,j) / ( rho(i,j+1)*d_Nn(1,j)/rho_in + rho(i,j)*d_Pn(1,j)/rho_in );
            rho_s(i,j) = d_Ps(1,j) / ( rho(i,j-1)*d_Ss(1,j)/rho_in + rho(i,j)*d_Ps(1,j)/rho_in );
            rho_e(i,j) = d_Pe(i,1) / ( rho(i+1,j)*d_Ee(i,1)/rho_in + rho(i,j)*d_Pe(i,1)/rho_in );
            rho_w(i,j) = d_Pw(i,1) / ( rho(i-1,j)*d_Ww(i,1)/rho_in + rho(i,j)*d_Pw(i,1)/rho_in );
        else
            % In solids, set face densities to zero to suppress updates/fluxes
            rho_n(i,j) = 0;
            rho_s(i,j) = 0;
            rho_e(i,j) = 0;
            rho_w(i,j) = 0;
        end

        % Discrete coefficients for psi (Poisson/Laplace-like operator)
        % a_* ~ (face property) * (face area) / (distance to neighbor)
        a_n(i-1,j-1) = rho_n(i,j) * (dx) / d_Pn(1,j);
        a_s(i-1,j-1) = rho_s(i,j) * (dx) / d_Ps(1,j);
        a_e(i-1,j-1) = rho_e(i,j) * (dy) / d_Pe(i,1);
        a_w(i-1,j-1) = rho_w(i,j) * (dy) / d_Pw(i,1);

        % Central coefficient as sum of neighbor contributions (no source term here)
        a_p(i,j) = a_w(i-1,j-1) + a_s(i-1,j-1) + a_e(i-1,j-1) + a_n(i-1,j-1);
    end
end

% Boundary/ghost layers: set a_p = 1 to stabilize/anchor Dirichlet-like psi BCs
a_p([1, N+2], :) = 1;
a_p(:, [1, M+2]) = 1;

% Helper to mirror/copy face densities into ghost layers (keeps stencils consistent)
updateDensity = @(rho, idx, jdx) rho(idx, jdx);

% West/East ghost updates for rho_w
for j = 1:M+2
    rho_w(1,j)   = updateDensity(rho_w,   2, j);  % West ghost <- interior
    rho_w(N+2,j) = updateDensity(rho_w, N+1, j);  % East ghost <- interior
end

% South/North ghost updates for rho_s
for i = 1:N+2
    rho_s(i,1)   = updateDensity(rho_s, i,   2);      % South ghost <- interior
```

```
    rho_s(i,M+2) = updateDensity(rho_s, i, M+1);      % North ghost <- interior
end

% Ghost updates for rho_n
for j = 1:M+2
    rho_n(1,j)   = updateDensity(rho_n,   2, j);
    rho_n(N+2,j) = updateDensity(rho_n, N+1, j);
end

% Ghost updates for rho_e
for i = 1:N+2
    rho_e(i,1)   = updateDensity(rho_e, i,   2);
    rho_e(i,M+2) = updateDensity(rho_e, i, M+1);
end

% GAUSS{SEIDEL SOLVER (psi)
iteration = 1;

while iteration <= n_it
    % Apply streamfunction boundary conditions on ghost layers
    j = 2:M+1;
    psi(:, 1)   = 0;                  % Left boundary reference (psi = 0)
    psi(N+2, :) = psi(N+1, :);        % Right boundary: zero-gradient (Neumann)
    psi(1, j)   = V_in * yEdge(1,j); % Bottom: uniform inflow, psi = V * y
    psi(:, M+2) = V_in * H;           % Top: psi constant at V*H

    % Gauss{Seidel sweep over interior cells
    for i = 2:N+1
        for j = 2:M+1
            if celltype(i, j) == 1
                % Discrete equation: sum(a_nb * psi_nb) + b_p = a_p * psi_P
                psi(i, j) = ( a_e(i-1, j-1) * psi(i+1, j) + ...
                              a_w(i-1, j-1) * psi(i-1, j) + ...
                              a_n(i-1, j-1) * psi(i, j+1) + ...
                              a_s(i-1, j-1) * psi(i, j-1) + ...
                              b_p(i, j) ) / a_p(i, j);
            else
                % Inside the solid: prescribe psi using a 'factor' to emulate rotation
                if      case_selector == 1
                    factor = 1;   % Static
                elseif case_selector == 2
                    factor = 0.5; % Clockwise rotation
                elseif case_selector == 3
                    factor = 1.5; % Counter-clockwise rotation
                end
                % Here psi is set based on average of bottom/top boundaries, scaled by factor
                psi(i, j) = ( (psi(i, 1) + psi(i, M+2)) / 2 ) * factor;
            end
        end
```

```
    end

    % FACE VELOCITIES FROM STREAMFUNCTION (u,v)
    % Using u ~ d(psi)/dy and v ~ -d(psi)/dx (scaled by face densities here).
    for i = 1:N+2
        for j = 1:M+2
            % North face u-component (forward difference in y)
            if j < M+2
                uN(i, j) = rho_n(i, j) * (psi(i, j+1) - psi(i, j)) / d_Pn(1, j);
            end
            % South face u-component (backward difference in y)
            if j > 1
                uS(i, j) = rho_s(i, j) * (psi(i, j) - psi(i, j-1)) / d_Ps(1, j);
            end
            % East face v-component (backward difference in x with sign)
            if i < N+2
                vE(i, j) = rho_e(i, j) * (psi(i, j) - psi(i+1, j)) / d_Pe(i, 1);
            end
            % West face v-component (forward difference in x with sign)
            if i > 1
                vW(i, j) = rho_w(i, j) * (psi(i-1, j) - psi(i, j)) / d_Pw(i, 1);
            end
        end
    end

    % CELL-CENTER VELOCITIES, TEMPERATURE AND PRESSURE (post-proc)
    for i = 1:N+2
        for j = 1:M+2
            % Approximate cell-center u and v by averaging face values
            Vx_P(i, j) = (uN(i, j) + uS(i, j)) / 2;
            Vx_P(i, 1)   = V_in;   % enforce inlet-like value on bottom ghost
            Vx_P(i, M+2) = V_in;   % enforce on top ghost
            Vy_P(i, j) = (vE(i, j) + vW(i, j)) / 2;

            V_P(i, j) = sqrt(Vx_P(i, j)^2 + Vy_P(i, j)^2); % speed magnitude

            % Temperature from a Bernoulli-like exchange: T + V^2/(2cp) = const
            T_p(i, j) = T_0(i, j) + (V_in^2 - V_P(i, j)^2) / (2 * cp);

            % Pressure from isentropic relation using T (valid at low Mach)
            P_p(i, j) = P_0(i, j) * ( T_p(i, j) / T_0(i, j) )^( gammaAir / (gammaAir - 1) );
        end
    end

    % Prepare for next iteration (psi already updated in-place)
```

```
    psi_0 = psi; % keep last psi as reference (not strictly needed in current scheme)
    T = T_p;      % update state copies (for clarity)
    P = P_p;
    iteration = iteration + 1;
end

% CIRCULATION (line integral)
% Accumulate contributions only for faces that are adjacent to the solid.
for i = 2:N+1
    for j = 2:M+1
        if      celltype(i,j-1) == 0
            % Solid just south: use north-face contribution (+)
            tauN(i,j) =  uN(i,j) * (xEdge(1,i+1) - xEdge(1,i));
        elseif celltype(i,j+1) == 0
            % Solid just north: use south-face contribution with (-)
            tauS(i,j) = -uS(i,j) * (xEdge(1,i)   - xEdge(1,i-1));
        elseif celltype(i-1,j) == 0
            % Solid just west:  use east-face (v) with (-)
            tauE(i,j) = -vE(i,j) * (yEdge(1,j+1) - yEdge(1,j));
        elseif celltype(i+1,j) == 0
            % Solid just east:  use west-face (v) with (+)
            tauW(i,j) =  vW(i,j) * (yEdge(1,j)   - yEdge(1,j-1));
        else
            % Not adjacent to solid: no contribution
            tauN(i,j) = 0;
            tauS(i,j) = 0;
            tauW(i,j) = 0;
            tauE(i,j) = 0;
        end
    end
end
i = 1:N+2;
j = 1:M+2;
Tau(i,j) = tauN(i,j) + tauS(i,j) + tauE(i,j) + tauW(i,j); % local sum of edge terms
Circulation = sum(Tau(:));                                 % total circulation

% DRAG
% Pressure integration proxy along body surface (finite differences in x).
drag = zeros(N+2, M+2);
for i = 1:N+2
    for j = 1:M+2
        if celltype(i,j) == 0
            % East-West pressure difference times face length (delta_Y)
            drag(i,j) = -P_p(i+1,j)*dy + P_p(i-1,j)*dy;
        end
    end
end
Drag = sum(drag(:)); % Net drag per unit span [N/m]

% LIFT
```

```
% Pressure integration proxy along body surface (finite differences in y).
lift = zeros(N+2, M+2);
for i = 1:N+2
    for j = 1:M+2
        if celltype(i,j) == 0
            % North-South pressure difference times face length (delta_X)
            lift(i,j) = -P_p(i,j+1)*dx + P_p(i,j-1)*dx;
        end
    end
end
Lift = sum(lift(:)); % Net lift per unit span [N/m]

% OUTPUTS
fprintf('RESULTS: ');
fprintf('Circulation: %.6e [m^2/s]\n', Circulation);
fprintf('Drag: %.6e [N/m]\n', Drag);
fprintf('Lift: %.6e [N/m]\n', Lift);

% FIGURES / VISUALIZATION
[a, b] = meshgrid(yEdge, xEdge);    % Note: 'b' -> X, 'a' -> Y (ordering matches fields)
alpha  = 0 : 0.01 : 2*pi;           % Angle parameter for cylinder outline
X_plot = (D/2) * cos(alpha) + L / 2;
Y_plot = (D/2) * sin(alpha) + H / 2;

% PRESSURE DISTRIBUTION
figure(1);
contourf(b, a, P_p, 80, 'LineColor', 'none'); % Filled pressure contours
colormap("parula");
colorbar;
hold on;
plot(X_plot, Y_plot, 'k', 'LineWidth', 2);     % Cylinder outline
axis equal;
xlim([0, L]);
ylim([0, H]);
xlabel('X [m]', 'FontSize', 12);
ylabel('Y [m]', 'FontSize', 12);
title('Pressure Distribution', 'FontSize', 14);
hold off;

% TEMPERATURE DISTRIBUTION
figure(2);
contourf(b, a, T_p, 50, 'LineColor', 'none'); % Filled temperature contours
colormap ("jet");
colorbar;
hold on;
plot(X_plot, Y_plot, 'k', 'LineWidth', 2);
axis equal;
xlim([0, L]);
ylim([0, H]);
xlabel('X [m]', 'FontSize', 12);
ylabel('Y [m]', 'FontSize', 12);
```

```
title('Temperature Distribution', 'FontSize', 14);
hold off;

% STREAMLINES
figure(3);
hold on;
contourf(b, a, psi, 50);            % Filled contours of streamfunction
colormap("winter");
xlabel ('X [m]');
ylabel ('Y [m]');
axis equal;
xlim([0,L]);
ylim([0,H]);
plot(X_plot, Y_plot, 'k', 'LineWidth', 2);
title('Streamlines', 'FontSize', 14);
hold off;

toc; % Elapsed time
```