Numerical Analysis for Non - Viscous Potential Flows

Student name: Altadill Llasat, Miquel

Professor name: Perez Segarra, Carlos David

Department: Centre Tecnològic de Transferència de Calor (CTTC)

Document: Final Report

Delivery Date: 23/03/2020

MSc in Aeronautic Engineering - ESEIAAT
Universitat Politècnica de Catalunya (UPC)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This report shows to the reader the application of the Potential Flow theory for two simple case of study. The objective is the understanding of the theory and how it is applied for developing a numerical analysis on this cases. In order to present the contents in a ordered and fluid way the report is divided in three main chapters.

Chapter 1 shows the basis of the applied theory and how it is needed to work with the govern equations in order to develop a suitable numerical analysis. Since the objective is not to deepen into the numerical study the work is focused on the mathematical modeling and physical phenomena study of the problems presented.

The following chapters are directly focused on the application of the aforementioned theory to two different cases in order to give a better understanding of the Potential Theory. This chapters keep the same structure, talking about the problem definition and how it has been discretized. Furthermore, it shows the coding structure followed for the development of the problem and the results and conclusions obtained.

1.1 Potential Flow

All the sciences of engineering need a compromise between reality and the needed simplifications for the mathematical calculations. The potential flow is a highly idealized theory used for modeling flows that match with the following assumptions:

- *Incompressibility.* The density and specific weight of the fluid is considered constant.
- *Irrotationality.* It implies a non-viscous fluid which particles move without rotation.

$$rot\overrightarrow{V} = 0 \tag{1.1}$$

- *Permanent Flow.* Which means that all the fluid properties and parameters are independent from time.(1)

This theory only allows to solve regions of the flow subjected to the non-viscous potential conditions, further away from the boundaries. Which means that it would not be possible to model the effects produced by the interaction of the fluid with the boundaries, leaving the boundary layer interaction study to a second plane. The analysis of non-viscous regions is determined by the Euler Eq. 1.2.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \tag{1.2a}$$

$$\frac{\partial \rho \vec{v}}{\partial t} + \nabla \cdot (\rho \vec{v} \vec{v}) = -\nabla p \tag{1.2b}$$

$$\frac{\partial E}{\partial t} + \nabla \cdot (E \vec{v}) = -\nabla \cdot (p \vec{v}) \tag{1.2c}$$

When it comes to solving the Euler equations, where the non-viscous regions can be considered irrational, two approaches will be considered:

- *Mathematical formulation based on the stream function definition:* Valid for steady 2D flows. The irrotational condition is not compulsory.
- *Mathematical formulation based on the velocity potential formulation:* Only valid for irrotational flows. However it might be used on unsteady and/or 3D flows. (4)

For developing this project is has been chosen the first approach. The stream function is defined by Eq. 1.3 for a steady 2D case which satisfies the mass conservation equation $\nabla \cdot (\rho \vec{v}) = 0$.

$$v_x = \frac{\rho_0}{\rho} \frac{\partial \phi}{\partial y} \tag{1.3a}$$

$$v_y = -\frac{\rho_0}{\rho} \frac{\partial \phi}{\partial x} \tag{1.3b}$$

The stream function equation is obtained the vorticity $\overrightarrow{w} = \nabla \overrightarrow{v}$, giving:

$$\frac{\partial}{\partial x}\left(\frac{\rho_0}{\rho}\frac{\partial \phi}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{\rho_0}{\rho}\frac{\partial \phi}{\partial y}\right) = w_z \tag{1.4}$$

Which for irrotational flows results in

$$\frac{\partial}{\partial x}\left(\frac{\rho_0}{\rho}\frac{\partial \phi}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{\rho_0}{\rho}\frac{\partial \phi}{\partial y}\right) = 0 \tag{1.5}$$

Equation 1.5 is the one which defines the mathematical and physical principles involved in the resolution of the problem. Once reached these equation its time to develop its discretization for the application of the Numerical Analysis.

1.2 Numerical Analysis

1.2.1 Control-Volume Formulation

The basic idea of the control-volume formulation lends itself to direct physical interpretation. The calculation domain is divided into a number of nonoverlapping control volumes so that there is one control volume surrounding each grid point. Then it is needed to integrate the differential equation over each control volume. Piece-wise profile seen in Fig. 1.1 shows the variation of ϕ between their neighbour grid points used for evaluating the govern equations integrals resulting in a discretization equation that will contain the values of ϕ for a group of grid points.

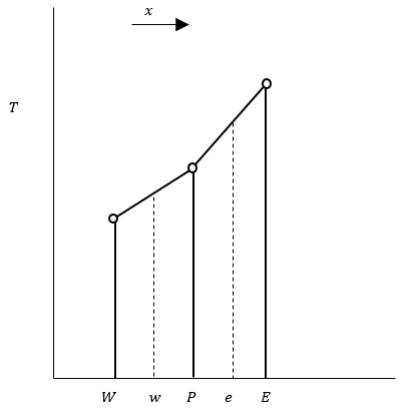

Figure 1.1: Piecewise - linear profile

The conservation equation obtained from the finite control volume study expresses its conservation principles for ϕ, just as the differential equation expresses it for an infinitesimal control volume. One of the most desirable features of this formulation is that the resulting solution would imply the integral conservation quantities such as mass, momentum and energy is exactly satisfied over any group of control volumes and over the whole calculation domain.

When it is needed to solve the discretization equation to obtain the grid-point values of ϕ, which are only needed to constitute the solution, without any explicit reference as to how ϕ varies between the grid points. The solution should only be dependent on the grid point values.

Grid-point scheme shown in the Fig.1.2 is going to be employed to derive the discretization equation.. This scheme can be extended to two and three-dimensional situations, that is why it is only needed to describe it in this section. Grid point P has grid points W and E as neighbours, E denotes east $(+x)$ side and W west side $(-x)$. The dashed lines show the faces of the control volume denoted by letters e and w following the previous criterion.

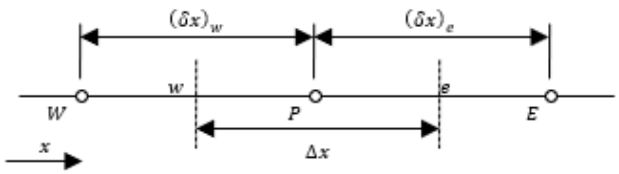

Figure 1.2: Grid-point scheme for one-dimensional problem

1.2.2 Equation Discretization

As it has been said, when the govern equation is obtained is is needed to discretize it. The discretization concepts lays in the idea of dividing the domain of study into different small interconnected pieces, each one of them governed by the discretized equation. Since each of the problems needed to be solved are considered irrotational, Stokes Theorem (Eq. 1.6)shows the path for obtaining the discretized equation.

$$\Gamma = \oint_C \overrightarrow{v} \cdot d\overrightarrow{l} \tag{1.6}$$

Due to the irrotationality the Circulation (Γ) is assumed to be zero. For a 2D case the integration of the velocity over the control volume faces gives the following expression:

$$\Gamma = v_{ye}\Delta y_P - v_{xn}\Delta x_P - v_{yw}\Delta y_P + v_{xs}\Delta x_P = 0 \tag{1.7}$$

Introducing the stream function definition the differentials in x and y directions are expressed as:

$$-\frac{\rho_0}{\rho_e}\frac{\psi_E - \psi_P}{d_{PE}}\Delta y_P - \frac{\rho_0}{\rho_n}\frac{\psi_N - \psi_P}{d_{PN}}\Delta x_P + \frac{\rho_0}{\rho_w}\frac{\psi_P - \psi_W}{d_{PW}}\Delta y_P + \frac{\rho_0}{\rho_s}\frac{\psi_P - \psi_S}{d_{PS}}\Delta x_P = 0 \tag{1.8}$$

Rearranging Eq. 1.8 terms is is obtained the coefficient form of the final discretization equation needed to solve the Potential Flow problem in the interior nodes volumes shown in Fig. 1.4.

$$a_P\psi_P = a_E\psi_E + a_W\psi_W + a_N\psi_N + a_S\psi_S + b_P \tag{1.9}$$

With coefficients,

$$a_E = \frac{\rho_0}{\rho_e}\frac{\Delta y_P}{d_{PE}} \tag{1.10a}$$

$$a_W = \frac{\rho_0}{\rho_w}\frac{\Delta y_P}{d_{PW}} \tag{1.10b}$$

$$a_N = \frac{\rho_0}{\rho_n}\frac{\Delta x_P}{d_{PN}} \tag{1.10c}$$

$$a_S = \frac{\rho_0}{\rho_s}\frac{\Delta x_P}{d_{PS}} \tag{1.10d}$$

$$a_P = a_E + a_W + a_N + a_S \tag{1.10e}$$

$$b_P = 0 \tag{1.10f}$$

1.3 Geometry Discretization

There are many possible ways for locating the control-volume but for the aim of this project only two different grids are explained. The description of each one will refer to a two-dimensional situation, although the concepts involved are applicable to one and three-dimensional situations.

1.3.1 Grid A: Faces located midway between the grid points

One of the most intuitive practises to construct the control volume is to place their faces *midway* between neighbouring grid points as seen in Fig.1.3. For building this grid to a 2-D plate grid points should be placed on its boundaries. Another observation is that the grid is nonuniform; the consequence of which is the grid point P does not lie at the geometric centre of the control volume.

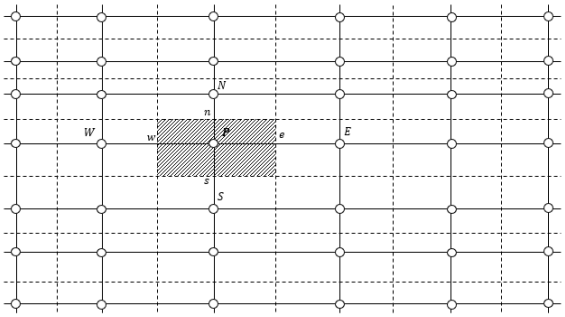

Figure 1.3: Location of control-volume faces - Grid A

1.3.2 Grid B: Grid points placed at the centres of the control-volumes

Another way to draw the grid is to draw the control-volume boundaries first and then place the grid point at the geometric centre of each control-volume. As it is seen in Fig.1.4 when the control volume sizes are non-uniform, their faces does not lie midway between the grid points.

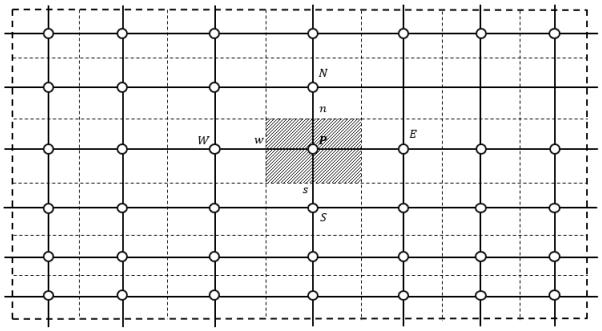

Figure 1.4: Location of control-volume faces - Grid B

The fact that the grid point P in Fig.1.3 may not be at the geometric centre of the control volume represents a disadvantage. In a heat conduction case, the temperature T_P cannot be observed as good representative value for the control volume in the calculation of the source term, the conductivity, and similar quantities(5). Grid A also presents objections in the calculation of the heat fluxes at the control volume faces, taking grid point e in Fig.1.3, for example, it is seen that it is

not at the centre of the control-volume face in which it lies. Thus, its assumed that the heat flux at e prevails over the entire face brings some inaccuracy.

Grid B does not have these problems because the point P lies at the centre of the control volume by definition and points such as e lies at the centre of their respective faces. One of the decisive advantages of Grid B is that the control volume turns out to be the basic unit of the discretization method, it is more convenient drawing the control-volume boundaries first and let the grid-point locations follow as consequence.

There are some advantages of Grid A over Grid B but the aforementioned advantages that Grid B represents over grid A makes to consider that the election of Grid B for the problem formulation is going to be the most suitable. It is needed to make additional considerations for the control volume near the boundaries of the domain. In the chosen case (Grid B) it is convenient to completely fill the calculation domain with regular control volumes and to place the boundary grid points on the faces of the near-boundary control volume faces. Figure 1.5 shows this arrangement of the Grid B where a typical boundary face i is located not between the boundary point B and the internal point I, it actually passes through the boundary point.

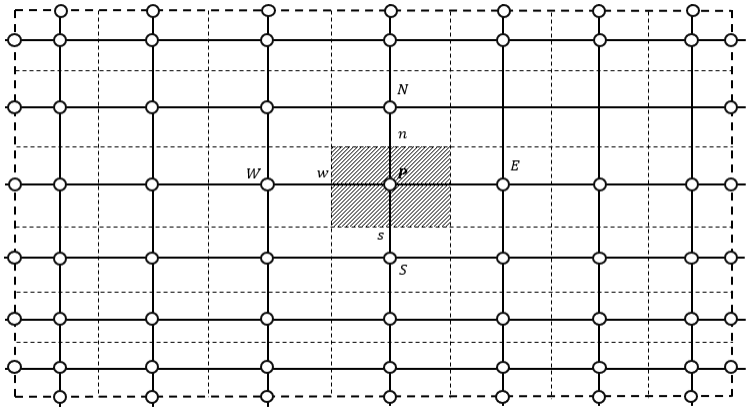

Figure 1.5: Boundary control volumes in Practice B

Chapter 2

Case 1: Flow in a Chanel

2.1 Problem Definition

The first case of study consists in solving a simple flow in a channel. Figure 3.1 shows a graphical representation of the real problem, which would consists in two flat plates with an air flow in between them. As it is one of the most simple cases for studying the Potential Flow this case is a suitable first approach for modelling the fluid field. Table 2.1 and 2.2 show the geometrical parameters and physical properties needed for solving the problem.

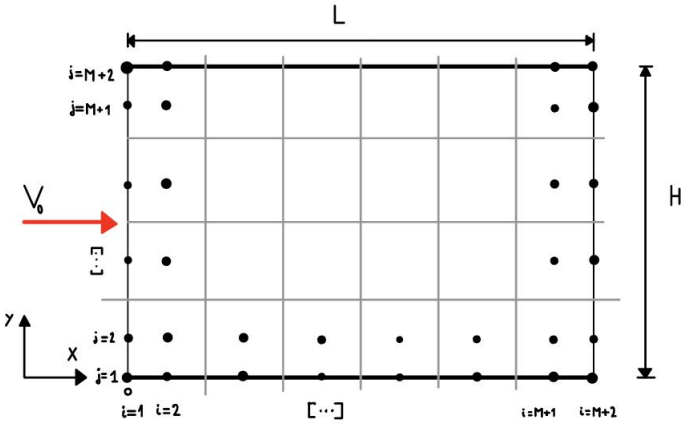

Figure 2.1: Problem definition and geometry discretization for Case 1

Geometrical Property	L	H	N	M
Value	1	1	50	50

Table 2.1: Domain geometry discretization for Case 1

Physical Property	$T(K)$	$P(Pa)$	$\rho(kg/m^3)$	γ	$R(J/kgK)$	$c_p(J/kgK)$	ψ_0	$v_0(m/s)$
Value	300	100000	1.0595	1.394	287	$5/2 \cdot R$	3	4

Table 2.2: Fluid of study (Air) physical properties for Case 1

2.1.1 Boundary Conditions

For the numerical analysis and development of each case of study it is necessary to establish the meaning of boundary conditions. Those conditions are the ones applied to the nodes that draw the limits of our domain. There are different conditions to be applied in each case but for the scope of this project the ones described below are enough to model any of the phenomenons of study.

2.1.2 Dirichlet Boundary Condition

When solving ordinary or partial differential equations in the presence of a boundary, there needs to be a boundary condition on the solution. Dirichlet boundary conditions are a specification of the value that the solution takes itself on the boundaries of the domain. This type of boundary condition is also called a fixed boundary condition. For this study the condition is applied setting a variable ϕ with a prescribed value at the boundary nodes.

$$\phi = Val \tag{2.1}$$

No-Slip Boundary Condition

When studying the movement of fluids such as in the convection-diffusion phenomenon it is often necessary to apply the no-slip condition. It tells us that the fluid velocity at all fluid–solid boundaries is equal to that of the solid boundary:

$$\phi = v; \quad Val = v_{wall}; \quad \rightarrow \quad v = v_{wall} \tag{2.2}$$

The use of the no-slip condition illustrates well the use of scientific models and idealizations but more importantly, that condition gives us a realistic macroscopic approach to a realistic model.(6)

2.1.3 Neumann Boundary Condition

This condition when imposed on an ordinary or a partial differential equation specifies the derivative values of a solution over the boundary domain:

$$\frac{d\phi}{dn} = Val \tag{2.3}$$

One possible application is to describe the heat flux over a surface. For example, if an adiabatic boundary is desired it would be necessary to set this derivative to zero. A possible interpretation of this condition is that the variable ϕ remains constant over the boundary if the set value is zero. Imposing this condition to the outlet would represent that the flow exists the boundary. Table 2.3 shows the conditions defined for solving this case of study.

Boundary	a_N	a_S	a_E	a_W	a_P	b_P	ψ_P
Top	0	0	0	0	1	0	$v_o \cdot H$
Bottom	0	0	0	0	1	0	0
Inlet	0	0	0	0	1	0	$v_0 \cdot y_P$
Outlet	0	0	0	1	a_W	$v_0 \cdot y_P$	ψ_W

Table 2.3: Boundary conditions for Case 1

2.2 Algorithm Structure

The developed algorithm followed for solving the first case of study follows a very simple and intuitive programming structure. As it is seen in Figure 3.1 the domain is defined by N and M for each direction, this parameters define the number of computational nodes in the domain of study. It needs to be noted that two extra nodes are fixed for each one of the boundaries in X and Y directions. Since the stream function needs to be computed at the centre of each CV the grid formulation developed in Section 1.3.2 is the most suitable one. For the nodal velocity computing additional considerations need to be taken into account because it need to perform an average between the CV faces velocity value. When developing the code some errors may be introduced due to the fact that the number of faces for each direction is $M + 1$ and $N + 1$ which is a different number from the nodal points. This consideration is very important for Gauss-Seidel based solvers.

Once told the main idea about how the parameters matrices dimensions would be, its time to focus on how each function of the code works and what its purpose is. The code is divided into different functions called in a *main.m* file, we can find each function in the Appendix B.

InputData.m

This function contains all the parameters needed to run all the functions in the main file. It contains the domain geometry, it can work with negative values of the coordinate system. It also has information about the mesh size and physical constants of the fluid of study. The reference values and the initial field values are also introduced in this file. If any numerical parameter is required for the solver, it is also introduced in this file, such as the error constrain or the maximum number of iterations.

UniformMesh.m

This function is the one in charge of generating the mesh for the domain discretization. It works reading the matrix with the introduced domain points and the number of nodes for each direction, which is saved in a vector for simplifying the function inputs. Inside this function works using another inside function that firstly place the control volume faces and then sets each node in between each face. If any other mesh type is required this function would always be in charge of generating it.

InitializeField.m

For computing and solving the problem it is needed to save into the internal memory some matrices that would represent the required field such as the Temperature or Stream. It counts the number of nodes of the generated domain to correctly define the array bounds, after that it saves the introduced value in the InputData corresponding to each field. It also computes the initial velocity field needed to solve the problem.

InteriorCoefficients.m

Since this case is a non time dependent and incompressible case the coefficients would only need to be computed at the beginning of the code. This function computes the interior nodes coefficients and save them into its corresponding array. If a direct solver is implemented it is always needed to compute and save all the coefficients before the solver implementation but for iterative solvers it

would be a good consideration to compute each needed coefficient when visiting each node. By this method it is possible to save a large amount of RAM memory but for this case is more efficient to compute them all at the beginning and accessing each one when needed since they remain constant along each iteration.

BoundaryConditions.m

The previous function only filled the interior coefficients, now its time to fill the boundaries with the defined boundary conditions. This function saves the required BC for the stream function, the coefficients and the velocity field. Once saved all the information required for the solving the discretized equations is ready.

GSsolver.m

The solver is the one in charge of solving the discretized equations for obtaining the problem results. In this case it has been implemented an iterative Gauss-Seidel solver. It firstly solves the stream function for the latter computing of the pressure, temperature and velocity field. It also introduces the Neumann BC because it does not iterate all the domain, it goes from i=2 to i=N+1, wich means that the boundaries are not computed so this condition has to be forced by the use of the *neumannbc.m* function. If a direct solver was implemented the use of this function would not be necessary because it would be implemented when reading the coefficients at the boundaries.

PostProcess.m

This generic function is the one that contains all the data visualization and post-processing tools required for the solution of the problem. For example, in our case it would be necessary to rotate the solution arrays in order to have the same orientation as the problem scheme.

2.3 Results

Once the code has been cleaned for errors and gives an apparently physically response it needs to be validated. For this it is necessary to test the results in different conditions and see if the symmetry is maintained and if the results match with experimental or computational data or analytical solutions.

2.3.1 Code Verification

In order to verify the results validity it is needed to compare the obtained results with an already verified problem solution. As Figure 2.2 shows, the expected results consists in a uniform distribution of the potential field value along the horizontal direction of the flow. Each one of the streamline has a constant value of $v_0 \cdot y$.

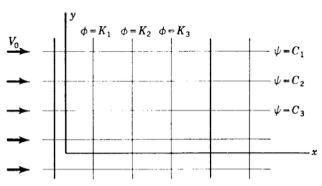

Figure 2.2: Flow net for uniform potential flow (1)

The following figure shows the behaviour of the developed code. It is seen that the the Stream Function streamlines correspond to the expected ones which would give the code the needed validation for the further computing of the velocity, pressure, temperature and density field.

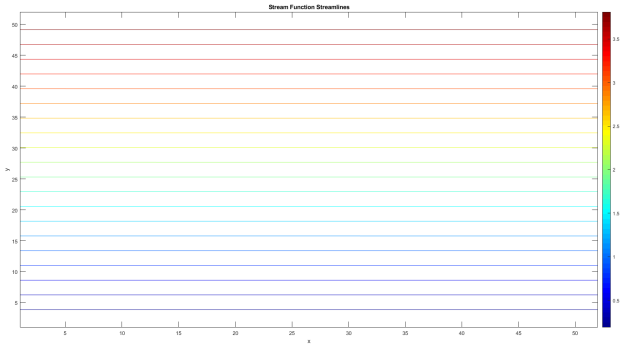

Figure 2.3: Stream Function streamlines for a 50x50 mesh in Case 1

2.3.2 Case of Study

Before commenting the results obtained in the case of study one needs to note that the plots are presented in a inverse way from the problem definition scheme shown in Figure 3.1 because of how the matrices are treated when computing the solution. The only consideration is that the bottom boundary corresponds with the top side of the plot and the top boundary with the bottom of the plot.

Figure 2.4 shows how the stream function values varies from zero to $v_0 \cdot H$ as it is expected. Once it is obtained the stream function value it could be computed the temperature, density, pressure and velocity field.

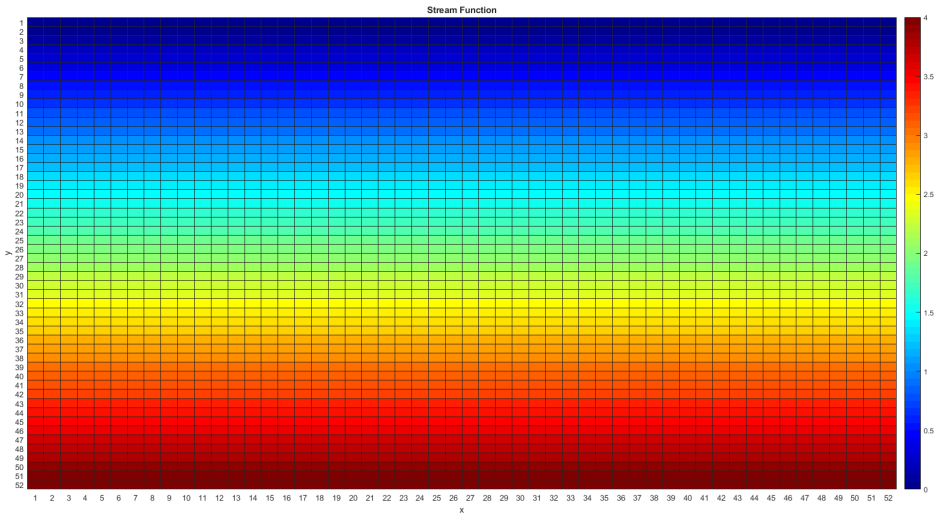

Figure 2.4: Stream Function for a 50x50 mesh in Case 1

Since the fluid of study is considered non-viscous as it was expected the velocity along the channel does not vary, taking as value the one introduced at the inlet condition. Figure 2.5 also shows the no-slip boundary condition placed at the solid boundaries of the domain of study.

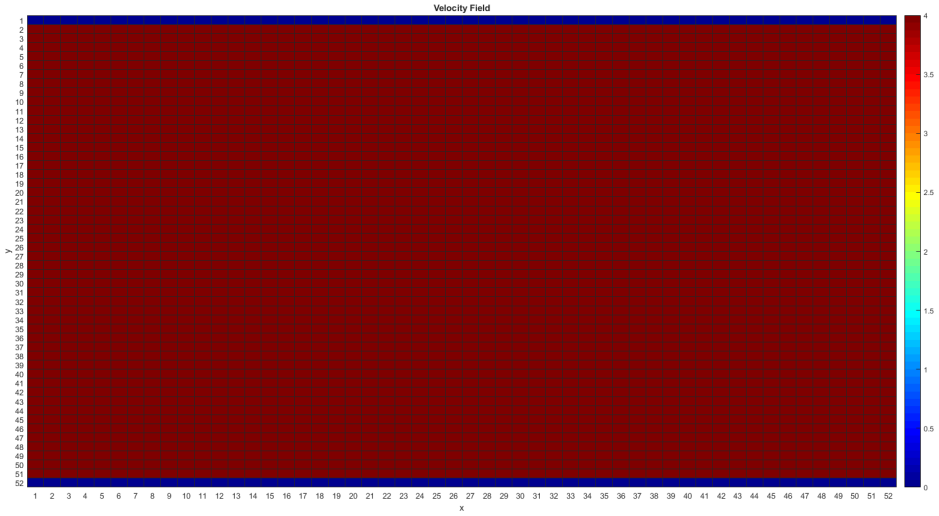

Figure 2.5: Velocity field for a 50x50 mesh in Case 1

The following Figures (Fig. 2.6 - 2.7 - 2.8) show the expected solution for the temperature, density and pressure field. Although it seems that it changes along the domain points it does not, this effect is produced by the representation method chosen. If one focus on the scale shown at the right side of the plots, it could be noted that the variation is almost zero, which is a suitable result. Since the boundary nodes are not computed the boundaries keep the fixed value.

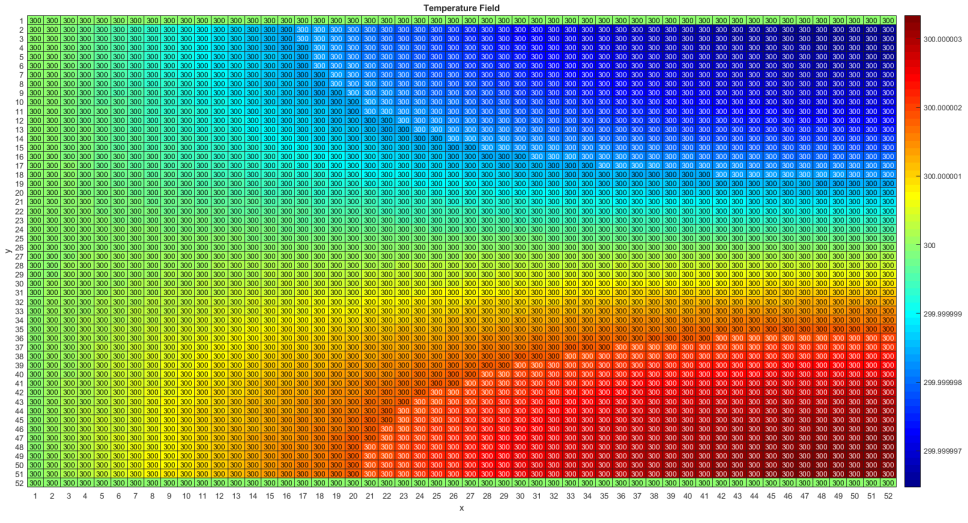

Figure 2.6: Temperature field for a mesh in Case 1

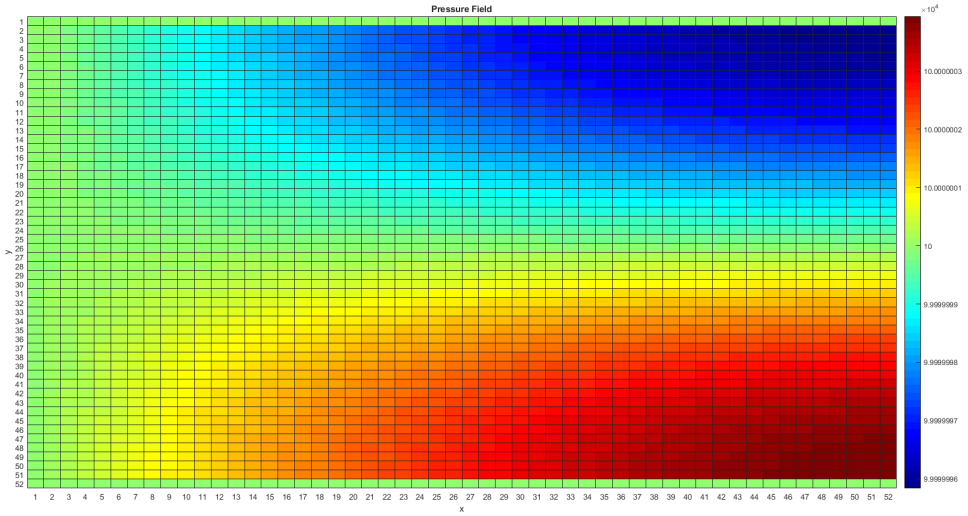

Figure 2.7: Pressure field for a 50x50 mesh in Case 1

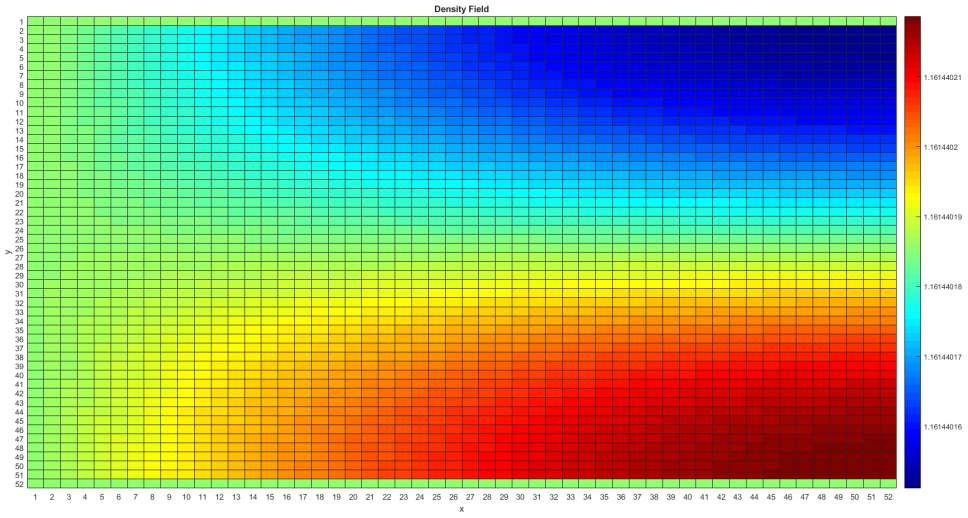

Figure 2.8: Density field for a 50x50 mesh in Case 1

2.3.3 Future Improvements

The development of this study has led to see the large size of the Numeric Analysis field and the number of applications it has. The fact of being working in a specific field always leads to new ideas on how you could improve since you have many possibilities to explore.This section talks about improvements and extensions of the project that have been coming up throughout its development.

Numerical Solver

It would be great to compare the performance of different Solvers applied to the developed cases of study. This would give us information for making a comparative study between each method talking about its advantages or disadvantages against the others for each case.

C Programming & HPC

Unless its complexity, C language gives us a better computational performance than Matlab and also has the option of working directly with the processors. That leaves the possibility to develop a paralleled Numerical Solver which would give a much better performance, this would allow us to run larger simulations in less time.

For further stages of this work, an application to C language and High Performance Computing of the developed codes would be an advantage for studying the results of a paralleled against the old ones.

Computational Resolution

For improving the performance of the code it would be necessary to study how each of the solver parameters would affect to the computational costs of the code. Some example parameters would be the mesh size and nodal distribution.

- **Error Analysis:** It is necessary to have all the sources of error inside your solver located in order to avoid deviation of the final results obtained.

Chapter 3

Case 2: Flow Over a Cylinder

3.1 Problem Definition

This case of study consists in solving a simple flow in a channel for the study of how it interacts with simple objects such as a cylinder. Figure 3.1 shows a graphical representation of the real problem, which would consists in two flat plates with an air flow in between them with a cylinder placed in between. This case is an extension from the first one and it only presents few modifications. Table 3.1 and 3.2 show the geometrical parameters and physical properties needed for solving the problem.

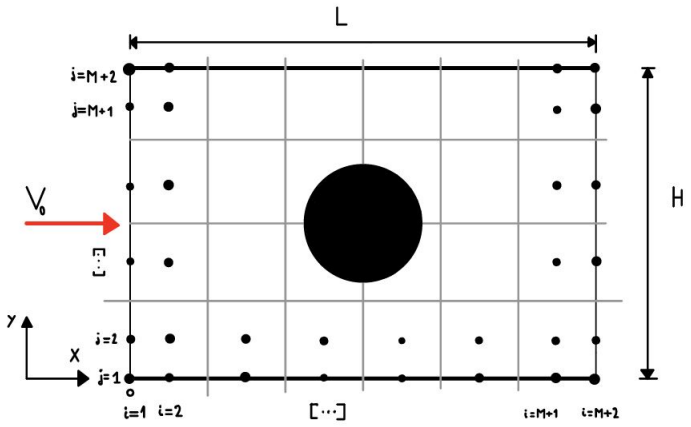

Figure 3.1: Problem definition and geometry discretization for case 2

Geometrical Property	L	H	R	N	M
Value	1	1	$H/10$	101	101

Table 3.1: Domain geometry discretization for Case 2

Physical Property	$T(K)$	$P(Pa)$	$\rho(kg/m^3)$	γ	$R(J/kgK)$	$c_p(J/kgK)$	ψ_0	$v_0(m/s)$
Value	300	100000	1.0595	1.394	287	$5/2 \cdot R$	3	4

Table 3.2: Fluid of study (Air) physical properties for Case 1

3.1.1 Boundary Conditions

Table 3.3 shows the conditions defined for solving this case of study.

Boundary	a_N	a_S	a_E	a_W	a_P	b_P	ψ_P
Top	0	0	0	0	1	0	$v_o \cdot H$
Bottom	0	0	0	0	1	0	0
Inlet	0	0	0	0	1	0	$v_0 \cdot y_P$
Outlet	0	0	0	1	a_W	$v_0 \cdot y_P$	ψ_W

Table 3.3: Boundary conditions for Case 2

3.1.2 Blocking - Off

The use of this methods is only needed when the analysis of the fluid field involves the interaction of solid-fluid regions. First of all it is needed to define where the solid boundaries are placed in the domain geometry and when the discretization is applied the code has to distinguish from solid or fluid Control Volumes. The constrain for setting its nature is based on where the centroid of the CV lays, if it is inside the solid region, it is considered as a solid and the same for fluid ones.

The treatment of the fluid CVs surrounding are the main point of this criteria. Taking the East node as an example it is known that its circulation is computed as

$$\Gamma_e \approx v_{ye}\Delta y_P = -\frac{\rho_0}{\rho_e}\frac{\partial \psi}{\partial x}\Delta y_P \tag{3.1}$$

In the case of solid-fluid interface the continuity of the ψ derivative is not clear, however it could be approximated as follows

$$v_{ye} = v_{ye}^- = v_{ye}^+ \tag{3.2}$$

$$\begin{aligned} v_{ye} &= -\frac{\rho_0}{\rho_P}\frac{\psi_e - \psi_P}{d_{Pe}} = \\ &= -\frac{\rho_0}{\rho_E}\frac{\psi_E - \psi_e}{d_{Ee}} \end{aligned} \tag{3.3}$$

$$v_{ye} = -\frac{\rho_0}{\rho_e}\frac{\psi_E - \psi_P}{d_{PE}} \tag{3.4}$$

For computing ρ_e it is needed to be evaluated at the face of the solid-fluid boundary. For evaluating this term it is needed to compute the harmonic mean between them, Appendix A shows the interface conductivity concept for computing the harmonic mean applied to a Heat Conduction case, which is transferable to our case of study.

3.2 Algorithm Structure

The developed algorithm followed for solving the first case of study follows a very simple and intuitive programming structure. As it is seen in Figure 3.1 the domain is defined by N and M for each direction, this parameters define the number of computational nodes in the domain of study.

It needs to be noted that two extra nodes are fixed for each one of the boundaries in X and Y directions. Since the stream function needs to be computed at the centre of each CV the grid formulation developed in Section 1.3.2 is the most suitable one. For the nodal velocity computing additional considerations need to be taken into account because it need to perform an average between the CV faces velocity value. When developing the code some errors may be introduced due to the fact that the number of faces for each direction is $M + 1$ and $N + 1$ which is a different number from the nodal points. This consideration is very important for Gauss-Seidel based solvers.

Once told the main idea about how the parameters matrices dimensions would be, its time to focus on how each function of the code works and what its purpose is. The code is divided into different functions called in a *main.m* file, we can find each function in the Appendix C. This function has almost the same function as the first one but with some additional considerations, this considerations are added in an extra paragraph to the following function explanation points. Only an extra function is added which is *material.m*.

InputData.m

This function contains all the parameters needed to run all the functions in the main file. It contains the domain geometry, it can work with negative values of the coordinate system. It also has information about the mesh size and physical constants of the fluid of study. The reference values and the initial field values are also introduced in this file. If any numerical parameter is required for the solver, it is also introduced in this file, such as the error constrain or the maximum number of iterations.

The radius of the cylinder is computed as a function of the domain height in order to reduce any possible code error and give the algorithm a fully optimization concept.

UniformMesh.m

This function is the one in charge of generating the mesh for the domain discretization. It works reading the matrix with the introduced domain points and the number of nodes for each direction, which is saved in a vector for simplifying the function inputs. Inside this function works using another inside function that firstly place the control volume faces and then sets each node in between each face. If any other mesh type is required this function would always be in charge of generating it.

Material.m

This function is the one in charge of generating the cylinder in the center of the domain. It works counting the number of nodes for each direction and then places the center of the cylinder in the central node if the nodes are odd or almost in the center if it is pair. That is the reason why the number of nodes in the computational study is kept to an odd number, because this way we ensure that the cylinder is placed at the center. If any other object was needed this function would be in charge of generating it and would save the data in a material array for a better distinction between solids or solid-fluid media.

InitializeField.m

For computing and solving the problem it is needed to save into the internal memory some matrices that would represent the required field such as the Temperature or Stream. It counts the number of nodes of the generated domain to correctly define the array bounds, after that it saves the introduced value in the InputData corresponding to each field. It also computes the initial velocity field needed to solve the problem.

For this case it also initializes the field inside the cylinder using the material data provided by the previous function. In the case of a cylinder with rotation this function would be the one in charge of setting the desired value of the stream function. In our case it fills the density field with value zero, in order to have a suitable computing of the interior coefficients.

InteriorCoefficients.m

Since this case is a non time dependent and incompressible case the coefficients would only need to be computed at the beginning of the code. This function computes the interior nodes coefficients and save them into its corresponding array. If a direct solver is implemented it is always needed to compute and save all the coefficients before the solver implementation but for iterative solvers it would be a good consideration to compute each needed coefficient when visiting each node. By this method it is possible to save a large amount of RAM memory but for this case is more efficient to compute them all at the beginning and accessing each one when needed since they remain constant along each iteration.

It only computes the coefficients when the density is different to zero, which mean that we are computing a fluid node. With this constrain and using the interface conductivity concept it is easily computed the coefficients at each fluid node. Coefficients at the solid are set to zero value.

BoundaryConditions.m

The previous function only filled the interior coefficients, now its time to fill the boundaries with the defined boundary conditions. This function saves the required BC for the stream function, the coefficients and the velocity field. Once saved all the information required for the solving the discretized equations is ready.

GSsolver.m

The solver is the one in charge of solving the discretized equations for obtaining the problem results. In this case it has been implemented an iterative Gauss-Seidel solver. It firstly solves the stream function for the latter computing of the pressure, temperature and velocity field. It also introduces the Neumann BC because it does not iterate all the domain, it goes from i=2 to i=N+1, which means that the boundaries are not computed so this condition has to be forced by the use of the *neumannbc.m* function. If a direct solver was implemented the use of this function would not be necessary because it would be implemented when reading the coefficients at the boundaries.

It is needed to set the constrain that avoids the solver to compute the solid boundaries, eliminating any possible error source and giving the desired valid results. If this condition is not introduced infinite values may appear due to the zero value of density fixed inside the cylinder of study.

PostProcess.m

This generic function is the one that contains all the data visualization and post-processing tools required for the solution of the problem. For example, in our case it would be necessary to rotate the solution arrays in order to have the same orientation as the problem scheme.

3.3 Results

Once the code has been cleaned for errors and gives an apparently physically response it needs to be validated. For this it is necessary to test the results in different conditions and see if the symmetry is maintained and if the results match with experimental or computational data or analytical solutions.

3.3.1 Code Verification

Although previous code has been verified this case also needs from an extra verification. It is known that the streamlines around the cylinder would be closer between them because of the reduction of the available section where the fluid go through. Figure 3.5 shows this behaviour giving physical meaning to the obtained results. But for checking the consistency of the numerical results it is needed from an extra verification.

According to (2) it is seen that there are existing points of null velocity at the leading edge and trainling edge of the cylinder, which would correspond to position 0 and 180 degree of the cylinder geometry, this points are known as stagnation points. Figure 3.2 shows that the points of maximum velocity exist in the upper and lower points of the cylinder with a corresponding value equivalent to the double of the initial velocity. Figure 3.3 verifies this statements giving the developed code the required validation.

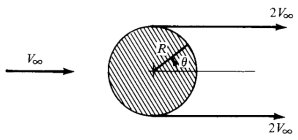

Figure 3.2: Maximum velocity in the flow over a cylinder (2)

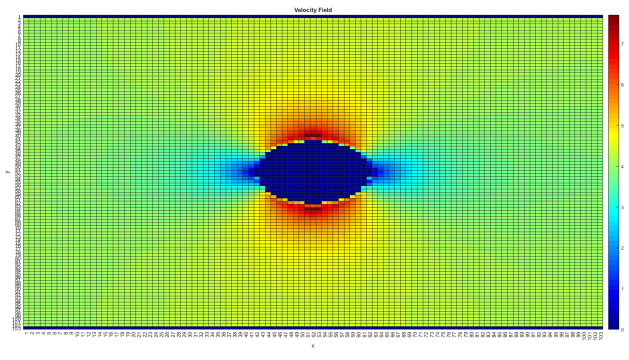

Figure 3.3: Velocity field for a 101x101 mesh in Case 2

3.3.2 Case of Study

Before commenting the results obtained in the case of study one needs to note that the plots are presented in a inverse way from the problem definition scheme shown in Figure 3.4 because of how the matrices are treated when computing the solution. The only consideration is that the bottom boundary corresponds with the top side of the plot and the top boundary with the bottom of the plot.

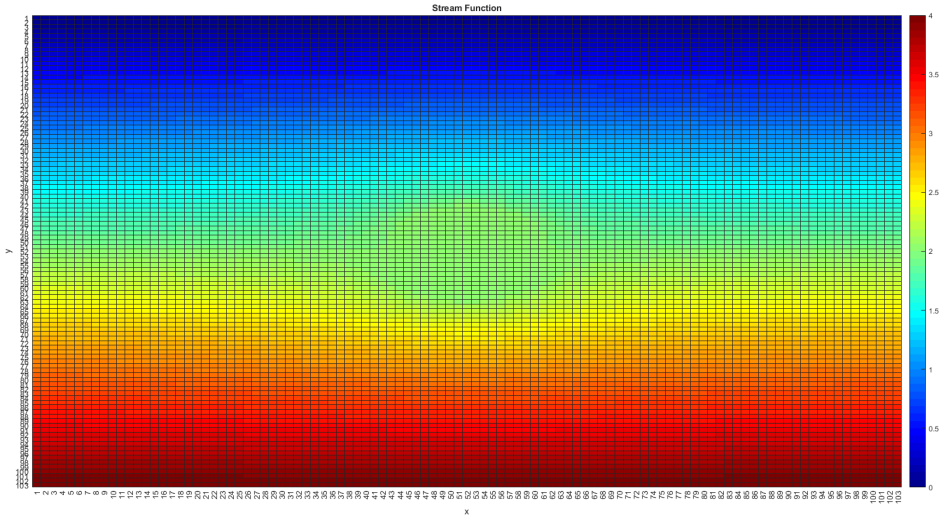

Figure 3.4: Stream Function for a 101x101 mesh in Case 2

Figure 3.5 show the aforementioned phenomena description. Due to the section narrowing the streamlines are closer to their neighbouring streamlines near the cylinder zone. It also remains almost constant or unperturbed in the zones near the inlet and oulet boundaries.

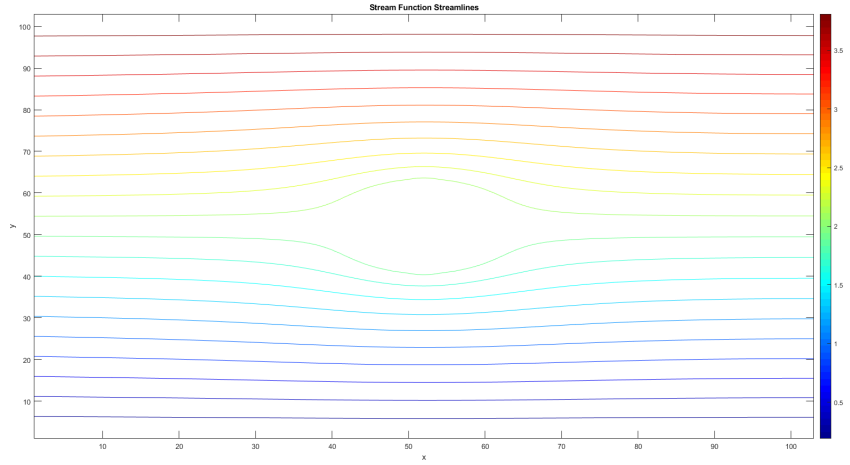

Figure 3.5: Stream Function streamlines for a 101x101 mesh in Case 2

From the velocity field description it is easy to have a previous idea about how the pressure field would be distributed. Figure 3.6 shows how the pressure is greater in the zones where the fluid is decelerated. In the upper and lower region the fluid accelerates which means that the pressure energy latent in the fluid transforms in order to increase the velocity.

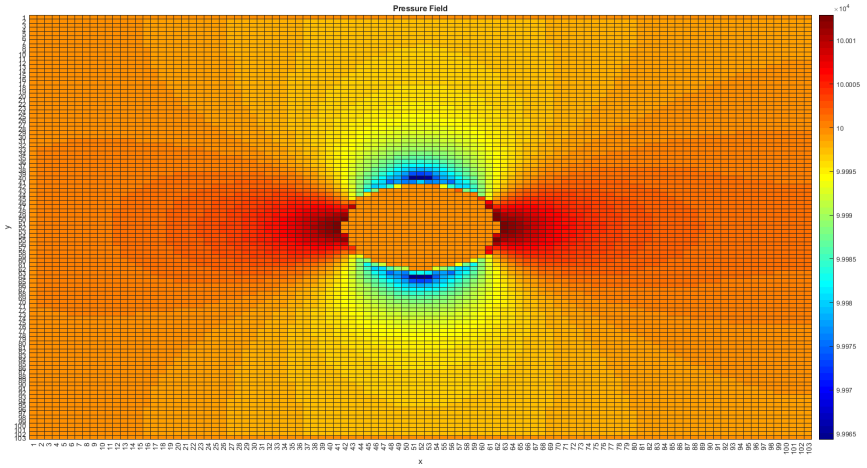

Figure 3.6: Pressure field for a 101x101 mesh in Case 2

A greater pressure zone also implies a greater temperature such as Fig. 3.7 shows. The zones where the fluid is accelerated the temperature reduces. Physically it would mean that the density would also be reduced but for the nature of the study Fig3.8 does not show any density variation because of the incomprehensibility definition. This density reduction can also be reasoned in the point of view of streamline closing up. Fluid needs to reduce its density in order to fit in a smaller section.

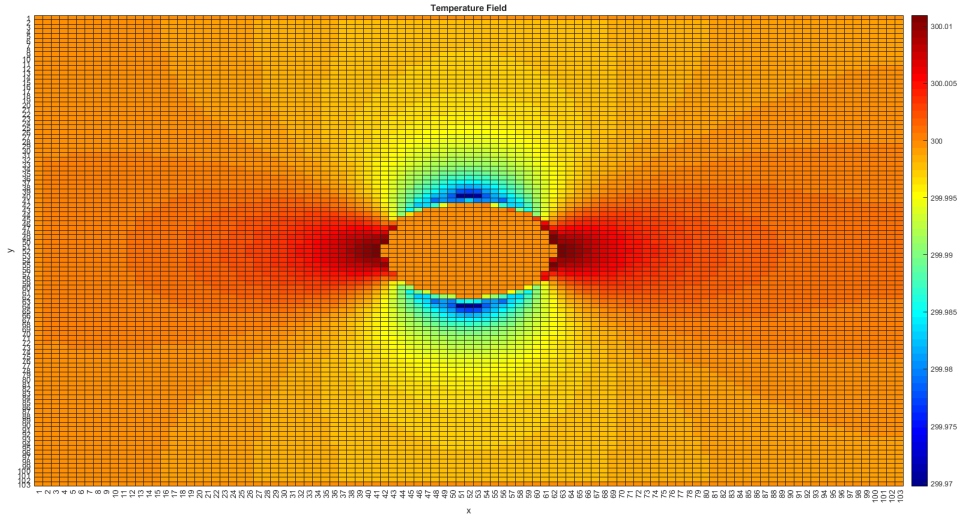

Figure 3.7: Temperature field for a 101x101 mesh in Case 2

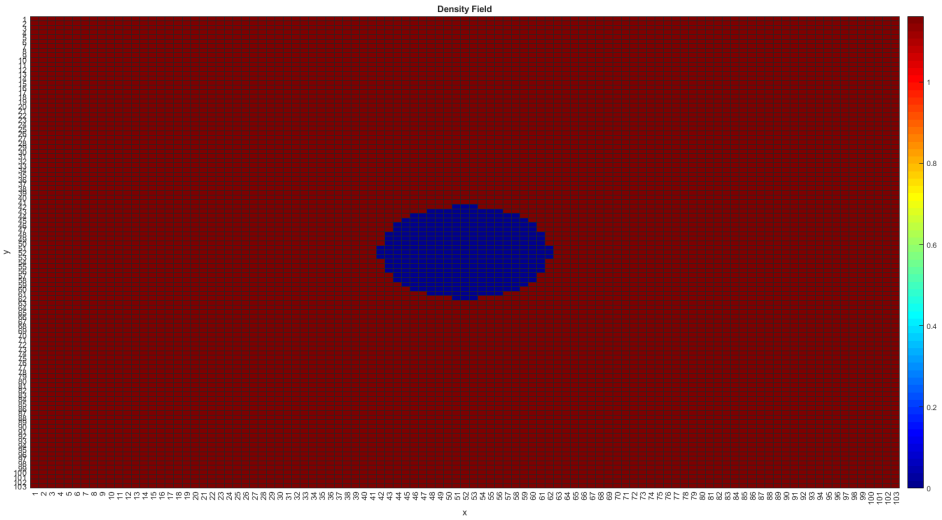

Figure 3.8: Density field for a 101x101 mesh in Case 2

3.3.3 Future Improvements

The development of this study has led to see the large size of the Numeric Analysis field and the number of applications it has. The fact of being working in a specific field always leads to new ideas on how you could improve since you have many possibilities to explore.This section talks about improvements and extensions of the project that have been coming up throughout its development.

Numerical Solver

It would be great to compare the performance of different Solvers applied to the developed cases of study. This would give us information for making a comparative study between each method talking about its advantages or disadvantages against the others for each case.

C Programming & HPC

Unless its complexity, C language gives us a better computational performance than Matlab and also has the option of working directly with the processors. That leaves the possibility to develop a paralleled Numerical Solver which would give a much better performance, this would allow us to run larger simulations in less time.

For further stages of this work, an application to C language and High Performance Computing of the developed codes would be an advantage for studying the results of a paralleled against the old ones.

Computational Resolution

For improving the performance of the code it would be necessary to study how each of the solver parameters would affect to the computational costs of the code. Some example parameters would be the mesh size and nodal distribution.

- **Error Analysis:** It is necessary to have all the sources of error inside your solver located in order to avoid deviation of the final results obtained.

Mesh Refinement

The developed algorithm generates a uniform mesh which works locating each node to an equal distance from its neighbouring node. As an improvement it would be great to have an hyperbolic distribution of the nodes for giving a better resolution near the cylinder boundary. That means that the solver would have a greater concentration of computing nodes near the surrounding of the cylinder increasing the numerical resolution of the code and therefore reducing the possible deviation errors of the results.

Compressible Flows

In order to give a better understanding on the fluid field behaviour when compressibility effects the developed algorithm can be modified for solving this cases.

NACA Profile Study

Once understood the physics behind the potential flow theory it is a good idea to apply the earned knowledge to a real practical case for studying a NACA profile. Since an object has already been introduced i would only be needed to modify the geometric constrain parameters in the blocking-off part for generating a suitable NACA profile.

Bibliography

[1] I. H. Shames, *Fluid Mechanics*. McGraw-Hill, 1995.

[2] J. J. D. Anderson, *Fundamentals of Aerodynamics*, 5th ed. McGraw-Hill, 2007.

[3] Y. A. Cengel, *Heat transfer: a practical approach*, 2nd ed., 2004.

[4] CTTC, *Course on Numerical Methods in Heat Transfer and Fluid Dynamics. Non-Viscous Flows*. ESEIAAT.

[5] S. V.Patankar, *Numerical Heat Transfer and Fluid Flow*, 1st ed., 1980.

[6] M. A., *The no-slip condition of fluid dynamics. Academic Press*. Erkenntnis, 2004.

Appendix A

Interface Conductivity Energy Balance

Figure A.1 refers to the composite slab steady one dimensional analysis without heat sources. For finding a generic solution to this problem it is needed to propose an energy balance over the interface. As the temperature over the interface is constant the heat flux over the interface is also constant:

$$q_e^P = q_e^E \tag{A.1}$$

From Eq.**??**:

$$q_e = \frac{k_P(T_P - T_e)}{(\partial x)_{e-}} = \frac{k_E(T_e - T_E)}{(\partial x)_{e+}} \tag{A.2}$$

Taking only T_e to the left side of the equation

$$T_e = \frac{T_P \dfrac{k_P}{(\partial x)_{e-}} + T_E \dfrac{k_E}{(\partial x)_{e+}}}{\dfrac{k_P}{(\partial x)_{e-}} + \dfrac{k_E}{(\partial x)_{e+}}} \tag{A.3}$$

Introducing Eq.A.3 into second term of Eq.A.2:

$$q_e = \frac{k_P}{(\partial x)_{e-}} \left(T_P - \frac{T_P \dfrac{k_P}{(\partial x)_{e-}} + T_E \dfrac{k_E}{(\partial x)_{e+}}}{\dfrac{k_P}{(\partial x)_{e-}} + \dfrac{k_E}{(\partial x)_{e+}}} \right) \tag{A.4}$$

After reordering terms and simplifying gives the desired form of the equation:

$$q_e = \frac{k_P k_E}{k_P(\partial x)_{e+} + k_E(\partial x)_{e-}}(T_P - T_E) = \frac{T_P - T_E}{\frac{(\partial x)_{e+}}{k_E} + \frac{(\partial x)_{e-}}{k_P}} \tag{A.5}$$

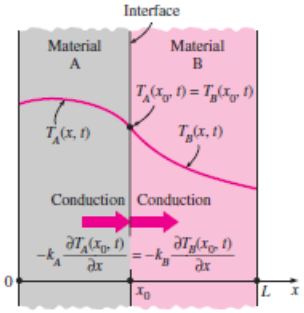

Figure A.1: Boundary conditions at the interface (3)

Appendix B

Case 1: Matlab Code

This appendix contains all the functions needed to solve the Case 1 part of this project. It is related to Chapter 2. The following sections shows each function by the order of its use in the algorithm.[1]

B.1 Main Algorithm

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%  ISENTROPIC POTENTIAL FLOW   %%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%  Miquel Altadill Llasat  %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc
clear all
close all


InputData;

%% MESH GENERATION
[nodeX,faceX,nodeY,faceY] = UniformMesh(domainP, meshSizes);

%% FIELD INITIALIZATION
[stream,p,T,rho,v] = InitializeField(p0, T0, istream,rho0, nodeX,nodeY,v0);

%% COEFFICIENTS
[coeff] = interiorcoefficients(rho,nodeX,faceX,nodeY,faceY,rho0);

%% BOUNDARY CONDITIONS
[stream,coeff,v] = BoundaryConditions(stream,v0,H,nodeY,meshSizes,coeff,v);

%% SOLVER
% - GS Based Solver
[stream,p,T,rho,v] = ...
    GSsolver(stream,coeff,nodeX,nodeY,sigma,meshSizes,p,T,rho,T0,p0,v0,fluidC,v);

%% PostProcess
PostProcess(stream,p,T,rho,v);
```

[1]This code can be downloaded by clicking "here".

B.2 Input Data

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%% Domain
% Domain Length
domainP=[0  1; 0 1];        %First  row for X dim
                            %Second row for Y dim
L   = domainP(1,2)-domainP(1,1);
H   = domainP(2,2)-domainP(2,1);


%% Mesh Size
% N  = Nodes X
% M  = Npdes Y
% H  = Domain Heigth - Y
% L  = Domain Length - X
% Mesh adds 2 extra nodes in each direcction
% for computing the boundaries

N  =  50;
M  =  50;
meshSizes=[N M];


%% Iterative solver parameters
% maxIter   =   Maximum number of iterations
% sigma     =   Error Parameter

maxIter=1e4;
sigma=1e-5;

istream = 3;


%% Phisical Constants
%  Fluid -> Air at 300K
% R     =   Gas Constant [J/kgK]

fluidC.R   = 287;
% Suposing ideal diatomic gas
fluidC.cp   = (5/2)*fluidC.R;
fluidC.gam = 1.394;

%% Reference Values %%
% rho   =   Density [kg/m^3]
% p     =   Pressure [Pa]
% T     =   Temperature [K]
% v     =   Velocity [m/s]
% R     =   GasConstant [kJ/kmolK]
% cp    =   Coefficent of pressure [Ws/kg K]
% Index 0 accounts for initial conditions ref vals.


```

```
p0   = 100000;
T0   = 273+27;
v0   = 4;
rho0 = p0/(fluidC.R*T0);
```

B.3 Mesh Generation

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%  MESH GENERATION   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [nodeX,faceX,nodeY,faceY] = UniformMesh(domainP, meshSizes)

xLength = domainP([1],[2])-domainP([1],[1]);
yLength = domainP([2],[2])-domainP([2],[1]);
domainLengths=[xLength, yLength];

%% X AXIS
dim=1;
[nodeX,faceX]=facesZVB(domainLengths(dim),...
          meshSizes(dim),domainP([dim],[1]));



%% Y AXIS
dim=2;
[nodeY,faceY]=facesZVB(domainLengths(dim),...
  meshSizes(dim),domainP([dim],[1]));

%nodeY = flip(nodeY);
%faceY = flip(faceY);

end

function [nx,fx]=facesZVB(length,numCV,initPoint)

fx=linspace(initPoint,initPoint+length,numCV+1);
nx(1,1)=initPoint;
nx(1,2:numCV+1)=(fx(2:end)+fx(1:end-1))*0.5;
nx(1,numCV+2)=initPoint+length;

end
```

B.4 Field Initialization

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%  FIELD INITIAILIZATION  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
5  function [stream,p,T,rho,v] = InitializeField(p0, T0, istream,rho0, ...
       nodeX,nodeY,v0);
6  sizeX = numel(nodeX);
7  sizeY = numel(nodeY);
8
9  stream = zeros(sizeY,sizeX);
10 p      = zeros(sizeY,sizeX);
11 T      = zeros(sizeY,sizeX);
12 rho    = zeros(sizeY,sizeX);
13
14 stream(:,:) = istream;
15 p(:,:)      = p0;
16 T(:,:)      = T0;
17 rho(:,:)    = rho0;
18
19
20 %% Velocity Field
21 v.vp     = zeros(sizeY,sizeX);
22
23 %Face Velocity  Nfaces = Nnodes-1
24 v.vxn    = zeros(sizeY-1,sizeX-1);
25 v.vye    = zeros(sizeY-1,sizeX-1); %No velocity expected in Y direction
26
27 % Velocity Field
28 v.vp(:,:)     = v0;
29 v.vxn(:,:)    = v0;
30 %vye = 0
31
32 end
```

B.5 Interior Coefficients

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%%%%%%%%%%%%%%%%%%%%%%  INTERIOR COEFFICIENT   %%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5  function [coeff] = interiorcoefficients(rho,nodeX,faceX,nodeY,faceY,rho0)
6
7      sizeX = numel(nodeX);
8      sizeY = numel(nodeY);
9
10     coeff.ap = zeros(sizeY,sizeX);
11     coeff.ae = zeros(sizeY,sizeX);
12     coeff.aw = zeros(sizeY,sizeX);
13     coeff.an = zeros(sizeY,sizeX);
14     coeff.as = zeros(sizeY,sizeX);
15     coeff.bp = zeros(sizeY,sizeX);
16
17
18     % Interior nodes Filling
19     for i = 2:sizeX-1
20         for j = 2:sizeY-1
21
22             %[rhoh] = harmonicmean(i,j,nodeX,nodeY,faceX,faceY,rho0,rho);
23
```

```
24              Dy = faceY(j)-faceY(j-1);
25              Dx = faceX(i)-faceX(i-1);
26
27              dPE = nodeX(i+1)-nodeX(i);
28              dPW = nodeX(i) - nodeX(i-1);
29              dPN = nodeY(j+1) - nodeY(j);
30              dPS = nodeY(j)  - nodeY(j-1);
31
32              coeff.ae(j,i) = (rho0/rho(j,1+1)) * (Dy/dPE);
33              coeff.aw(j,i) = (rho0/rho(j,i-1)) * (Dy/dPW);
34              coeff.an(j,i) = (rho0/rho(j+1,i)) * (Dx/dPN);
35              coeff.as(j,i) = (rho0/rho(j-1,i)) * (Dx/dPS);
36              coeff.ap(j,i) = ...
                    coeff.ae(j,i)+coeff.aw(j,i)+coeff.an(j,i)+coeff.as(j,i);
37              coeff.bp(j,i) = 0;
38
39
40              % Using HArmonic Mean
41
42
43  %             coeff.ae(j,i) = rhoh.e * (Dy/dPE);
44  %             coeff.aw(j,i) = rhoh.w * (Dy/dPW);
45  %             coeff.an(j,i) = rhoh.n * (Dx/dPN);
46  %             coeff.as(j,i) = rhoh.s * (Dx/dPS);
47  %             coeff.ap(j,i) = ...
        coeff.ae(j,i)+coeff.aw(j,i)+coeff.an(j,i)+coeff.as(j,i);
48  %             coeff.bp(j,i)= 0;
49
50          end
51      end
52
53
54  end
```

B.6 Boundary Conditions

```
1
2
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %%%%%%%%%%%%%%%%%%%%%%%%  BOUNDARY CONDITIONS   %%%%%%%%%%%%%%%%%%%%%%%%%%%%
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6
7
8  function [stream,coeff,v] = BoundaryConditions(stream,v0,H,nodeY,meshSizes,coeff,v)
9
10      N = meshSizes(1);
11      M = meshSizes(2);
12
13      %% Top
14      stream(M+2,:) = v0*H;
15      coeff.ap(M+2,:)      = 1;
16      coeff.aw(M+2,:)      = 0;
17      coeff.ae(M+2,:)      = 0;
18      coeff.an(M+2,:)      = 0;
19      coeff.as(M+2,:)      = 0;
```

```
20      coeff.bp(M+2,:)      = v0*H;
21
22      %No slip BC
23      v.vp(M+2,:)            = 0;
24      v.vxn(M+1,:)           = 0;
25
26      %% Bottom
27      stream(1,:)    = 0;
28      coeff.ap(1,:)      = 1;
29      coeff.aw(1,:)      = 0;
30      coeff.ae(1,:)      = 0;
31      coeff.an(1,:)      = 0;
32      coeff.as(1,:)      = 0;
33      coeff.bp(1,:)      = v0*H;
34
35      %No slip BC
36      v.vp(1,:)            = 0;
37      v.vxn(1,:)           = 0;
38
39      %% Inlet
40      stream(:,1) = v0*nodeY(:);
41      coeff.ap(:,1)      = 1;
42      coeff.ae(:,1)      = 0;
43      coeff.aw(:,1)      = 0;
44      coeff.an(:,1)      = 0;
45      coeff.as(:,1)      = 0;
46      coeff.bp(:,1)      = 0;
47
48      %% Outlet
49      % Stream not defined because it is already set when initiaizing field
50      coeff.ap(:,N+2)      = 1;
51      coeff.aw(:,N+2)      = 1;
52      coeff.ae(:,N+2)      = 0;
53      coeff.an(:,N+2)      = 0;
54      coeff.as(:,N+2)      = 0;
55      coeff.bp(:,N+2)      = v0*nodeY(:);
56
57
58  end
```

B.7 Gauss-Seidel Solver

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%  GAUSS-SEIDEL SOLVER  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [stream,p,T,rho,v] = ...
    GSsolver(stream,coeff,nodeX,nodeY,sigma,meshSizes,p,T,rho,T0,p0,v0,fluidC,v)



    N = meshSizes(1);
    M = meshSizes(2);

    eT   = 1;
    ep   = 1;
    erho = 1;
    n = 0;


    while eT> sigma  || ep> sigma  || erho > sigma

    % Initialize stream parameter;

    n = n+1
    % Discretized Stream Func Solve
    % Solver Core
    for i = 2:N+1
        for j = 2:M+1

            stream(j,i) = (coeff.ae(j,i)*stream(j,i+1) + ...
                           coeff.aw(j,i)*stream(j,i-1) + ...
                           coeff.an(j,i)*stream(j+1,i) + ...
                           coeff.as(j,i)*stream(j-1,i)) / coeff.ap(j,i);


        end
    end

    [stream] = neumannbc(stream,N);

    %Density Temperature and Pressure Solve
    % Field Solver
    Tini   = T;
    pini   = p;
    rhoini = rho;

    for i = 2:N+1
        for j = 2:M+1


            v.vxn(j,i) = (stream(j+1,i)-stream(j,i) ) / (nodeY(j+1) - nodeY(j));
            v.vye(j,i) = (stream(j,i+1)-stream(j,i) ) / (nodeX(i+1) - nodeX(i));

            % Due to the for dimensions it is necessary to keep face
            % velocity to zero, this if ensures that
            if j == M+1
```

```
               v.vxn(j,i) = 0;
               v.vye(j,i) = 0;
            end

            vxP = (v.vxn(j,i)+v.vxn(j-1,i))/2;
            vyP = (v.vye(j,i)+v.vye(j,i-1))/2;

            % Due to the for dimensions the mean value is the point value
            if j == 2
               vxP = v.vxn(j,i);
               vyP = v.vye(j,i);
            end
            if j == M+1
               vxP = v.vxn(j-1,i);
               vyP = v.vye(j,i-1);
            end

            v.vp(j,i) = sqrt(vxP^2 + vyP^2);

            % If -> Total energy conserved
            % Then:
            T(j,i) = T0 + (v0^2-v.vp(j,i)^2)/(2*fluidC.cp);

            % Isentropic relation applyed
            p(j,i) = p0 * (T(j,i)/T0)^(fluidC.gam/(fluidC.gam-1));

            % Density from gas relation
            % Air considered as an ideal gas
            rho(j,i) = p(j,i)/(fluidC.R*T(j,i));

        end
    end

    [p] = neumannbc(p,N);
    [rho] = neumannbc(rho,N);
    [T] = neumannbc(T,N);
    [v.vp] = neumannbc(v.vp,N);

    eT   = max(max(abs(T-Tini)));
    ep   = max(max(abs(p-pini)));
    erho = max(max(abs(rho-rhoini)));


    end



end
```

B.8 Neumann Boundary Condition

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%   NEUMANN BOUND. COND.  %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
function [x] = neumannbc(x,N)

x(:,N+2) = x(:,N+1);

end
```

B.9 Postproces

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%  POSTPROCESS  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function PostProcess(stream,p,T,rho,v)

figure(1);
h = heatmap(stream);  %heatmap
colormap(jet);
colorbar;
title('Stream Function');
xlabel('x'), ylabel('y');



figure(2);
contour(stream,20);
colormap(jet);
colorbar;
title('Stream Function Streamlines');
xlabel('x'), ylabel('y');


figure(3);
h = heatmap(p);  %heatmap
colormap(jet);
colorbar;
title('Pressure Field');
xlabel('x'), ylabel('y');


figure(4);
h = heatmap(T);  %heatmap
colormap(jet);
colorbar;
title('Temperature Field');
xlabel('x'), ylabel('y');


figure(5);
h = heatmap(rho);  %heatmap
colormap(jet);
colorbar;
title('Density Field');
xlabel('x'), ylabel('y');

```

```
48 figure(6)
49 h = heatmap(v.vp);
50 colormap(jet);
51 colorbar;
52 title('Velocity Field');
53 xlabel('x'), ylabel('y');
54
55 end
```

Appendix C

Case 2: Matlab Code

This appendix contains all the functions needed to solve the Case 1 part of this project. It is related to Chapter 3. The following sections shows each function by the order of its use in the algorithm.[1]

C.1 Main Algorithm

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%  ISENTROPIC POTENTIAL FLOW   %%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%  Miquel Altadill Llasat  %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clc
clear all
close all


InputData;

%% MESH GENERATION
[nodeX,faceX,nodeY,faceY] = UniformMesh(domainP, meshSizes);

%% MATERIAL
[mat] = material(nodeX,nodeY,radi);

%% FIELD INITIALIZATION
[stream,p,T,rho,v] = InitializeField(p0, T0, istream,rho0, nodeX,nodeY,v0,mat);

%% COEFFICIENTS
[coeff] = interiorcoefficients(rho,nodeX,faceX,nodeY,faceY,rho0);

%% BOUNDARY CONDITIONS
[stream,coeff,v] = BoundaryConditions(stream,v0,H,nodeY,meshSizes,coeff,v);

%% SOLVER
% - GS Based Solver
[stream,p,T,rho,v] = ...
    GSsolver(stream,coeff,nodeX,nodeY,sigma,meshSizes,p,T,rho,T0,p0,v0,fluidC,v,mat);
```

[1]This code can be downloaded by clicking "here".

```

%% PostProcess
PostProcess(stream,p,T,rho,v);
```

C.2 Input Data

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%  INPUT DATA   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%% Domain
% Domain Length
domainP=[0  1; 0 1];        %First  row for X dim
                            %Second row for Y dim
L   = domainP(1,2)-domainP(1,1);
H   = domainP(2,2)-domainP(2,1);

% Circle or Cilinder
radi =  H / 10;




%% Mesh Size
% N  = Nodes X
% M  = Npdes Y
% H  = Domain Heigth - Y
% L  = Domain Length - X
% Mesh adds 2 extra nodes in each direcction
% for computing the boundaries

N  =  101;
M  =  101;
meshSizes=[N M];


%% Iterative solver parameters
% maxIter    =   Maximum number of iterations
% sigma      =   Error Parameter

maxIter=1e4;
sigma=1e-5;

istream = 3;


%% Phisical Constants
%  Fluid -> Air at 300K
% R      =   Gas Constant [J/kgK]

fluidC.R   = 287;
% Suposing ideal diatomic gas
fluidC.cp   = (5/2)*fluidC.R;
fluidC.gam = 1.394;
```

```
50
51 %% Reference Values %%
52 % rho   =   Density [kg/m^3]
53 % p     =   Pressure [Pa]
54 % T     =   Temperature [K]
55 % v     =   Velocity [m/s]
56 % R     =   GasConstant [kJ/kmolK]
57 % cp    =   Coefficent of pressure [Ws/kg K]
58 % Index 0 accounts for initial conditions ref vals.
59
60
61 p0   = 100000;
62 T0   = 273+27;
63 v0   = 4;
64 rho0 = p0/(fluidC.R*T0);
```

C.3 Material

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %%%%%%%%%%%%%%%%%%%%%%%%     BLOCK IN - OFF       %%%%%%%%%%%%%%%%%%%%%%%%%%
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 % This function has to place a cilinder in the midddle of the domain of
5 % study, it works using the circunference equation as a constrain.
6 %
7 %
8
9
10 function [mat] = material(nodeX,nodeY,r)
11
12 sizeX = numel(nodeX);
13 sizeY = numel(nodeY);
14
15 mat = zeros(sizeY,sizeX);
16
17 N = sizeX - 2;
18 M = sizeY - 2;
19
20 a = nodeX(floor(sizeX/2) + 1);
21 b = nodeY(floor(sizeY/2) + 1);
22
23
24 for i = 2:N+1
25    for j = 2:M+1
26
27        % Incircle Condition
28
29        if (nodeY(j) - b)^2 + (nodeX(i) - a)^2 <= r^2
30            mat(j,i) = 1;
31        end
32
33
34    end
35 end
36
37
```

```
38 end
```

C.4 Uniform Mesh

```
1
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %%%%%%%%%%%%%%%%%%%%%%%%  MESH GENERATION   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5
6  function [nodeX,faceX,nodeY,faceY] = UniformMesh(domainP, meshSizes)
7
8  xLength = domainP([1],[2])-domainP([1],[1]);
9  yLength = domainP([2],[2])-domainP([2],[1]);
10 domainLengths=[xLength, yLength];
11
12 %% X AXIS
13 dim=1;
14 [nodeX,faceX]=facesZVB(domainLengths(dim),...
15           meshSizes(dim),domainP([dim],[1]));
16
17
18
19 %% Y AXIS
20 dim=2;
21 [nodeY,faceY]=facesZVB(domainLengths(dim),...
22   meshSizes(dim),domainP([dim],[1]));
23
24 %nodeY = flip(nodeY);
25 %faceY = flip(faceY);
26
27 end
28
29 function [nx,fx]=facesZVB(length,numCV,initPoint)
30
31 fx=linspace(initPoint,initPoint+length,numCV+1);
32 nx(1,1)=initPoint;
33 nx(1,2:numCV+1)=(fx(2:end)+fx(1:end-1))*0.5;
34 nx(1,numCV+2)=initPoint+length;
35
36 end
```

C.5 Field Initialization

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%%%%%%%%%%%%%%%%%%  FIELD INITIAILIZATION  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5  function [stream,p,T,rho,v] = InitializeField(p0, T0, istream,rho0, ...
       nodeX,nodeY,v0,mat);
6  sizeX = numel(nodeX);
7  sizeY = numel(nodeY);
```

```

stream = zeros(sizeY,sizeX);
p      = zeros(sizeY,sizeX);
T      = zeros(sizeY,sizeX);
rho    = zeros(sizeY,sizeX);

stream(:,:) = istream;
p(:,:)      = p0;
T(:,:)      = T0;
rho(:,:)    = rho0;


%% Velocity Field
v.vp     = zeros(sizeY,sizeX);

%Face Velocity  Nfaces = Nnodes-1
v.vxn    = zeros(sizeY-1,sizeX-1);
v.vye    = zeros(sizeY-1,sizeX-1); %No velocity expected in Y direction

% Velocity Field
v.vp(:,:)     = v0;
v.vxn(:,:)    = v0;
%vye = 0

%% Obstacle filling

for i = 2:sizeX-1
    for j =2:sizeY-1

        if mat(j,i) == 1
            rho(j,i)     = 0;
            stream(j,i)  = v0*(nodeY(sizeY)/2);
            v.vp(j,i)    = 0;
        end

    end
end

end
```

C.6 Interior Coefficients

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%  INTERIOR COEFFICIENT   %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [coeff,rhoh] = interiorcoefficients(rho,nodeX,faceX,nodeY,faceY,rho0)

    sizeX = numel(nodeX);
    sizeY = numel(nodeY);

    coeff.ap = zeros(sizeY,sizeX);
    coeff.ae = zeros(sizeY,sizeX);
    coeff.aw = zeros(sizeY,sizeX);
    coeff.an = zeros(sizeY,sizeX);
```

```
14      coeff.as = zeros(sizeY,sizeX);
15      coeff.bp = zeros(sizeY,sizeX);
16
17
18      % Interior nodes Filling
19      for i = 2:sizeX-1
20          for j = 2:sizeY-1
21
22              Dy = faceY(j)-faceY(j-1);
23              Dx = faceX(i)-faceX(i-1);
24
25              dPE = nodeX(i+1)-nodeX(i);
26              dPW = nodeX(i) - nodeX(i-1);
27              dPN = nodeY(j+1) - nodeY(j);
28              dPS = nodeY(j)  - nodeY(j-1);
29
30              N = nodeY(j+1);
31              S = nodeY(j-1);
32              E = nodeX(i+1);
33              W = nodeX(i-1);
34
35              fn = faceY(j);
36              fs = faceY(j-1);
37              fe = faceX(i);
38              fw = faceX(i-1);
39
40              Px = nodeX(i);
41              Py = nodeY(j);
42
43              if rho(j,i) ~= 0
44
45              rhohe = (dPE) / ( ((fe-Px)/(rho0/rho(j,i))) + ...
                    ((E-fe)/(rho0/rho(j,i+1))) );
46              rhohw = (dPW) / ( ((Px-fw)/(rho0/rho(j,i))) + ...
                    ((fw-W)/(rho0/rho(j,i-1))) );
47              rhohn = (dPN) / ( ((fn-Py)/(rho0/rho(j,i))) + ...
                    ((N-fn)/(rho0/rho(j+1,i))) );
48              rhohs = (dPS) / ( ((Py-fs)/(rho0/rho(j,i))) + ...
                    ((fs-S)/(rho0/rho(j-1,i))) );
49
50              coeff.ae(j,i) = rhohe * (Dy/dPE);
51              coeff.aw(j,i) = rhohw * (Dy/dPW);
52              coeff.an(j,i) = rhohn * (Dx/dPN);
53              coeff.as(j,i) = rhohs * (Dx/dPS);
54              coeff.ap(j,i) = ...
                    coeff.ae(j,i)+coeff.aw(j,i)+coeff.an(j,i)+coeff.as(j,i);
55              coeff.bp(j,i)= 0;
56
57              end
58
59          end
60      end
61
62
63  end
```

C.7 Boundary Conditions

```
1
2
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %%%%%%%%%%%%%%%%%%%%%%  BOUNDARY CONDITIONS   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6
7
8  function [stream,coeff,v] = BoundaryConditions(stream,v0,H,nodeY,meshSizes,coeff,v)
9
10     N = meshSizes(1);
11     M = meshSizes(2);
12
13     %% Top
14     stream(M+2,:) = v0*H;
15     coeff.ap(M+2,:)      = 1;
16     coeff.aw(M+2,:)      = 0;
17     coeff.ae(M+2,:)      = 0;
18     coeff.an(M+2,:)      = 0;
19     coeff.as(M+2,:)      = 0;
20     coeff.bp(M+2,:)      = v0*H;
21
22     %No slip BC
23     v.vp(M+2,:)            = 0;
24     v.vxn(M+1,:)           = 0;
25
26     %% Bottom
27     stream(1,:)    = 0;
28     coeff.ap(1,:)      = 1;
29     coeff.aw(1,:)      = 0;
30     coeff.ae(1,:)      = 0;
31     coeff.an(1,:)      = 0;
32     coeff.as(1,:)      = 0;
33     coeff.bp(1,:)      = v0*H;
34
35     %No slip BC
36     v.vp(1,:)            = 0;
37     v.vxn(1,:)           = 0;
38
39     %% Inlet
40     stream(:,1) = v0*nodeY(:);
41     coeff.ap(:,1)      = 1;
42     coeff.ae(:,1)      = 0;
43     coeff.aw(:,1)      = 0;
44     coeff.an(:,1)      = 0;
45     coeff.as(:,1)      = 0;
46     coeff.bp(:,1)      = 0;
47
48     %% Outlet
49     % Stream not defined because it is already set when initiaizing field
50     coeff.ap(:,N+2)      = 1;
51     coeff.aw(:,N+2)      = 1;
52     coeff.ae(:,N+2)      = 0;
53     coeff.an(:,N+2)      = 0;
54     coeff.as(:,N+2)      = 0;
55     coeff.bp(:,N+2)      = v0*nodeY(:);
```

```



end
```

C.8 Gauss-Seidel Solver

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%  GAUSS−SEIDEL SOLVER  %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [stream,p,T,rho,v] = ...
    GSsolver(stream,coeff,nodeX,nodeY,sigma,meshSizes,p,T,rho,T0,p0,v0,fluidC,v,mat)



    N = meshSizes(1);
    M = meshSizes(2);

    eT   = 1;
    ep   = 1;
    erho = 1;
    n = 0;


    while eT> sigma  || ep> sigma  || erho > sigma

    % Initialize stream parameter;

    n = n+1
    % Discretized Stream Func Solve
    % Solver Core
    for i = 2:N+1
        for j = 2:M+1

            if mat(j,i) == 0
            stream(j,i) = (coeff.ae(j,i)*stream(j,i+1) + ...
                           coeff.aw(j,i)*stream(j,i−1) + ...
                           coeff.an(j,i)*stream(j+1,i) + ...
                           coeff.as(j,i)*stream(j−1,i)) / coeff.ap(j,i);
            end

        end
    end

    [stream] = neumannbc(stream,N);

    %Density Temperature and Pressure Solve
    % Field Solver
    Tini   = T;
    pini   = p;
    rhoini = rho;

    for i = 2:N+1
        for j = 2:M+1

```

```
49              % Only compute at fluid media mat=0
50              if mat(j,i) ==0
51
52                  v.vxn(j,i) = (stream(j+1,i)-stream(j,i) ) / (nodeY(j+1) - ...
                        nodeY(j));
53                  v.vye(j,i) = (stream(j,i+1)-stream(j,i) ) / (nodeX(i+1) - ...
                        nodeX(i));
54
55                  % Due to the for dimensions it is necessary to keep face
56                  % velocity to zero, this if ensures that
57                  if j == M+1
58                     v.vxn(j,i) = 0;
59                     v.vye(j,i) = 0;
60                  end
61
62                  vxP = (v.vxn(j,i)+v.vxn(j-1,i))/2;
63                  vyP = (v.vye(j,i)+v.vye(j,i-1))/2;
64
65                  % Due to the for dimensions the mean value is the point value
66                  if j == 2
67                     vxP = v.vxn(j,i);
68                     vyP = v.vye(j,i);
69                  end
70                  if j == M+1
71                     vxP = v.vxn(j-1,i);
72                     vyP = v.vye(j,i-1);
73                  end
74
75                  v.vp(j,i) = sqrt(vxP^2 + vyP^2);
76
77                  % If -> Total energy conserved
78                  % Then:
79                  T(j,i) = T0 + (v0^2-v.vp(j,i)^2)/(2*fluidC.cp);
80
81                  % Isentropic relation applyed
82                  p(j,i) = p0 * (T(j,i)/T0)^(fluidC.gam/(fluidC.gam-1));
83
84                  % Density from gas relation
85                  % Air considered as an ideal gas
86                  rho(j,i) = p(j,i)/(fluidC.R*T(j,i));
87
88              end
89
90          end
91      end
92
93      [p] = neumannbc(p,N);
94      [rho] = neumannbc(rho,N);
95      [T] = neumannbc(T,N);
96      [v.vp] = neumannbc(v.vp,N);
97
98      eT   = max(max(abs(T-Tini)));
99      ep   = max(max(abs(p-pini)));
100     erho = max(max(abs(rho-rhoini)));
101
102     %[coeff] = interiorcoefficients(rho,nodeX,faceX,nodeY,faceY,rho0);
103
104
105     end
```

```
106
107
108
109 end
```

C.9 Postproces

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%%%%%%%%%%%%%%%%%%%%%%  POSTPROCESS  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5
6  function PostProcess(stream,p,T,rho,v)
7
8  figure(1);
9  h = heatmap(stream);  %heatmap
10 colormap(jet);
11 colorbar;
12 title('Stream Function');
13 xlabel('x'), ylabel('y');
14
15
16
17 figure(2);
18 contour(stream,20);
19 colormap(jet);
20 colorbar;
21 title('Stream Function Streamlines');
22 xlabel('x'), ylabel('y');
23
24
25 figure(3);
26 h = heatmap(p);  %heatmap
27 colormap(jet);
28 colorbar;
29 title('Pressure Field');
30 xlabel('x'), ylabel('y');
31
32
33 figure(4);
34 h = heatmap(T);  %heatmap
35 colormap(jet);
36 colorbar;
37 title('Temperature Field');
38 xlabel('x'), ylabel('y');
39
40
41 figure(5);
42 h = heatmap(rho);  %heatmap
43 colormap(jet);
44 colorbar;
45 title('Density Field');
46 xlabel('x'), ylabel('y');
47
48 figure(6)
```

```
49 h = heatmap(v.vp);
50 colormap(jet);
51 colorbar;
52 title('Velocity Field');
53 xlabel('x'), ylabel('y');
54
55 end
```