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

Master's In Space And Aeronautical Engineering

Computational Engineering Assignment 2

Generic Convection-Diffusion Equation

October, 2023

Author: Iñaki Fernandez

Index

1 Introduction **5**

2 Problem Description **10**
2.1 Diagonal Flow . . . 10
2.2 Smith-Hutton . . . 11

3 Code Structure **13**

4 Validation and Verification of the Code **14**
4.1 Comparison With Reference Solution . . . 14
4.2 Mesh Refinement . . . 15

5 Results **17**
5.1 Diagonal Flow . . . 17
5.2 Smith-Hutton . . . 18

6 Conclusions **20**

7 Appendix: Python3 Code **24**

If it is in the reader's interest, all the code done for this report can be found in the following repository: **Code**

1 Introduction

In this work our goal is to solve numerically the generic convection-diffusion equation. It is convenient to start the discussion by stating the Navier-Stokes (N-S) equations. For perfect gases, the N-S equations can be written as [1]:

$$\frac{\partial(\rho\mathbf{v})}{\partial t} + \nabla(\rho\mathbf{v}) = 0 \tag{1}$$

$$\frac{\partial(\rho\mathbf{v})}{\partial t} + \nabla(\rho\mathbf{v}\mathbf{v}) = \nabla(\mu\nabla\mathbf{v}) + [\nabla(\tau - \mu\nabla\mathbf{v}) - \nabla p + \rho\mathbf{g}] \tag{2}$$

$$\frac{\partial(\rho T)}{\partial t} + \nabla(\rho\mathbf{v}T) = \nabla\left(\frac{\lambda}{c_v}\nabla T\right) + \left[\frac{-\nabla\mathbf{q}^R - p\nabla\mathbf{v} + \tau : \nabla\mathbf{v}}{c_v}\right] \tag{3}$$

$$\frac{\partial(\rho Y_k)}{\partial t} + \nabla(\rho\mathbf{v}Y_k) = \nabla(\rho D_{km}\nabla Y_k) + (\dot{\omega_k}) \tag{4}$$

All these transport equations have an unsteady term, a convective term, a diffusion term and other terms. The generic convection diffusion equation for a generic variable ϕ can be expressed as:

$$\frac{\partial(\rho\phi)}{\partial t} + \nabla(\rho\mathbf{v}\phi) = \nabla(\Gamma_\phi\nabla\phi) + \dot{s_\phi}, \tag{5}$$

where Γ_ϕ and $\dot{s_\phi}$ are the diffusion coefficient and the extra source/sink term, respectively.

One can use the mass conservation equation (Eq. (1)) to write the equivalent convection-diffusion eqaution:

$$\rho\frac{\partial\phi}{\partial t} + \rho\mathbf{v}\nabla\phi = \nabla(\Gamma_\phi\nabla\phi) + \dot{s_\phi}. \tag{6}$$

To solve this numerically, we will use the finite volume method. This approach involves breaking down our computational domain into smaller Control Volumes (CVs). In Figure 1, we can see a representation of a CV along with its neighboring CVs.

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

As it can be noticed, in equation (6) we have a time partial derivative, This implies that we will also need a discretization along the time. This can be acieved by cutting

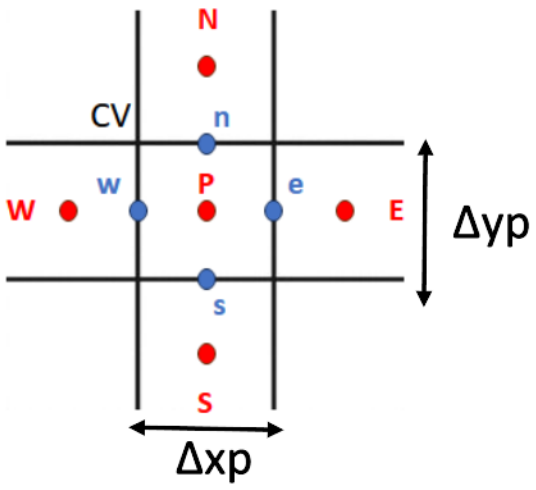


Figure 1: *A scheme of a CV and it's neighbours. In red the centred nodes: P, S, N, W and E. In blue the nodes centred in the walls of the CV: s, n, w and e.*

the time domain into small Δt time steps, such that the time evolution will be given as $t = t_0 + \Delta t \times i$, where i represents the total time steps.

Let's consider the integral form of the mass conservation (Eq. (1))

$$\frac{\partial}{\partial t} \int_{V_P} \rho dV + \int_{S_f} \rho \mathbf{v} \mathbf{n} dS = 0. \tag{7}$$

where V_P is the CV volume. Considering the centred point P (see Fig. 1) the mass conservation equation can be semi-discretized as:

$$V_P \frac{\partial \overline{\rho}_P}{\partial t} + \dot{m}_e - \dot{m}_w + \dot{m}_n - \dot{m}_s = 0 \tag{8}$$

where $\overline{\rho}_P = 1/V_P \int_{V_P} \rho dV$ and $\dot{m}_i = \int_{S_i} \rho \mathbf{v} \mathbf{n} dS$.

Integrating the above equation over time (between time t^n and t^{n+1}) we obtain:

$$V_P \int_{t^n}^{t^{n+1}} \frac{\partial \rho_P}{\partial t} dt + \int_{t^n}^{t^{n+1}} (\dot{m}_e - \dot{m}_w + \dot{m}_n - \dot{m}_s) dt = 0 \tag{9}$$

which gives the following implicit discretization form:

$$\frac{\rho_P - \rho_P^0}{\Delta t} V_P + \dot{m}_e - \dot{m}_w + \dot{m}_n - \dot{m}_s = 0 \tag{10}$$

where ρ_P and ρ_P^0 are the densities at point P for time steps $n+1$ and n, respectively.

Going back to Eq. (6), the numerical implicit approximation of the different integral form terms can be written as:

$$\int_{t^n}^{t^{n+1}} \int_{V_P} \frac{\partial(\rho\phi)}{\partial t} dV dt \approx V_P(\rho_P\phi_P - \rho_P^0\phi_P^0) \tag{11}$$

$$\int_{t^n}^{t^{n+1}} \int_{V_P} \nabla(\rho\mathbf{v}\phi) dV dt \approx (\dot{m}_e\phi_e - \dot{m}_w\phi_w + \dot{m}_n\phi_n - \dot{m}_s\phi_s)\Delta t \tag{12}$$

$$\int_{t^n}^{t^{n+1}} \int_{V_P} \nabla(\Gamma_\phi\nabla\phi) dV dt \approx \left(-\Gamma_w\frac{\phi_P - \phi_W}{d_{PW}}S_w + \Gamma_e\frac{\phi_E - \phi_P}{d_{PW}}S_e - \Gamma_s\frac{\phi_P - \phi_s}{d_{PS}}S_s + \Gamma_n\frac{\phi_N - \phi_P}{d_{PN}}S_n\right) \tag{13}$$

$$\int_{t^n}^{t^{n+1}} \int_{V_P} \dot{s_\phi} dV dt \approx (S_C^\phi + S_P^\phi\phi_P)V_P\Delta t \tag{14}$$

Using the above terms and discretized mass conservation equation, Eq. (10), we obtain the following equation:

$$\begin{gathered}\rho_P^0\frac{\rho_P - \rho_P^0}{\Delta t}V_P + \dot{m}_e(\phi_e - \phi_P) - \dot{m}_w(\phi_w - \phi_P) + \dot{m}_n(\phi_n - \phi_P) - \dot{m}_s(\phi_s - \phi_P) = \\ D_e(\phi_E - \phi_P) - D_w(\phi_P - \phi_W) + D_n(\phi_N - \phi_P) - D_s(\phi_P - \phi_S) + (S_C^\phi + S_P^\phi\phi_P)V_P\end{gathered} \tag{15}$$

where $D_e = \Gamma_e S_e/d_{PE}$ and $D_w = \Gamma_w S_w/d_{PW}$. On the one hand, it can be seen that the source and diffusion terms are well defined in the centred nodes (P,E,W,S and N) of the CVs. On the other hand, the convective terms are given in the nodes of the faces (e,w,s and n) of the CVs. At this point, one has to assume some approximations to evaluate the convective terms in the central nodes.

There are different evalutation methods for the convective terms. The simplest one is the central-difference scheme (CDS):

$$\phi_e - \phi_P = f_e(\phi_E - \phi_P) \tag{16}$$

where $f_e = d_{Pe}/d_{PE}$. Even though CDS is a second-order scheme, is yields to stability problems.

For incomprensible flows, or gases at low Mach numbers, the upwind-difference scheme (UDS) is a good choice since the convective terms are more influenced by upstream conditions than downstream ones. UDS is more stable than EDS but is a first-order accurate, so we gain stability but we lose accuracy. The UDS approximation can be written as:

$$\dot{m}_e(\phi_e^{UDS} - \phi_P) = \frac{\dot{m}_e - |\dot{m}_e|}{2}(\phi_E - \phi_P) \tag{17}$$

Other usefull scheme is the exponential-difference scheme (EDS). In this case, f_e function is defined as:

$$f_e = \frac{e^{Ped_{Pe}/d_{PE}} - 1}{e^{Pe} - 1} \tag{18}$$

where $Pe = \frac{\rho_e v_{xe} d_{PE}}{\Gamma_e}$ is the so-called Peclet number. EDS is still a first-order accurate scheme. More accurate ones (up to second-order) can be found in the literature as second-order upwind linear extrapolation (SUDS) and quadratic upwind interpolation for convective kinematics (QUICK). High-resolution schemes (HRS) can be also applied. The basic idea for the faces, f, can be expressed as:

$$\phi_f^{HRS} - \phi_P = (\phi_f^{UDS} - \phi_P) + (\phi_f^{HRS*} - \phi_f^{UDS*}) \tag{19}$$

In summary, using UDS the final form of the discretized generic convection-diffusion equation can be written as it follows:

$$a_P\phi_P = a_e\phi_E + a_W\phi_W + a_N\phi_N + a_S\phi_S + b_P \tag{20}$$

$$a_E = D_e - \frac{\dot{m}_e - |\dot{m}_e|}{2} \;; \quad a_W = D_W + \frac{\dot{m}_w + |\dot{m}_w|}{2} \tag{21}$$

$$a_N = D_n - \frac{\dot{m}_n - |\dot{m}_n|}{2} \;; \quad a_S = D_s + \frac{\dot{m}_s + |\dot{m}_s|}{2} \tag{22}$$

$$a_P = a_E + a_W + a_N + a_S + \frac{\rho_P^0 V_P}{\Delta t} - S_P^{\phi} V_P \tag{23}$$

$$\begin{aligned} b_P = \frac{\rho_P^0 V_P}{\Delta t}\phi_P^0 + S_C^{\phi} V_P - \dot{m}_e(\phi_e^{HRS*} - \phi_e^{UDS*}) + \dot{m}_w(\phi_w^{HRS*} - \phi_w^{UDS*}) \\ -\dot{m}_n(\phi_n^{HRS*} - \phi_n^{UDS*}) + \dot{m}_s(\phi_s^{HRS*} - \phi_s^{UDS*}) \end{aligned} \tag{24}$$

where for example, $\dot{m}_e \approx \rho(V_E - V_P)\Delta x \Delta z/2$.

Here we have to remark that in this work we considered no source term, $S_C = 0$ and no HRS correction. This implies that in the resolution of the above equations we will have to consider less terms.

We don't have to forget that the Boundary Conditions (BC) play a key roll in numerical calculations. A more detailed description around BC will be given in Section **2**.

2 Problem Description

In this section, we will outline the key elements of the two problem addressed within this work: the Diagonal Flow problem and the Smith-Hutton problem.

2.1 Diagonal Flow

For the Diagonal Flow scenario, we consider a symmetrical box with dimensions $L \times L$, as illustrated in Figure 2. This box is discretized into CVs, which have the same shape as depicted in Figure 1.

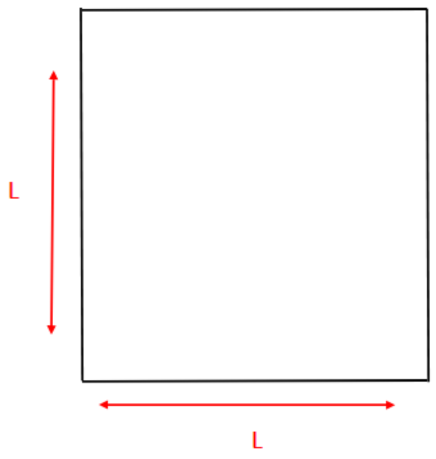


Figure 2: *A scheme of the diagonal flow case.*

For this problem, we impose Dirichlet boundary conditions on all four walls. Specifically, on the bottom and right walls, we set $\phi = \phi_{low}$ at $(x, y = 0)$ and $(x = L, y)$, while on the top and left walls, we set $\phi = \phi_{high}$ at $(x = 0, y)$ and $(x, y = H)$.

In terms of the coefficients a_i, where $i \in [P, E, W, N, S]$, and b_p, the values at the bottom and right walls are assigned as follows: $a_E = a_W = a_N = a_S = 0$, $a_P = 1$, and $b_P = \phi_{low}$. Similarly, for the top and left walls, the coefficients are set as $a_E = a_W = a_N = a_S = 0$, $a_P = 1$, and $b_P = \phi_{high}$.

The velocity field is described by the following equations:

$$v_x = V_{In} \cos(\alpha) \ ; \quad v_y = V_{In} \sin(\alpha) \tag{25}$$

Here, V_{In} is a constant value, and $\alpha = \pi/4$. The velocity field represented by Eq. (25) is visualized in Figure 3.

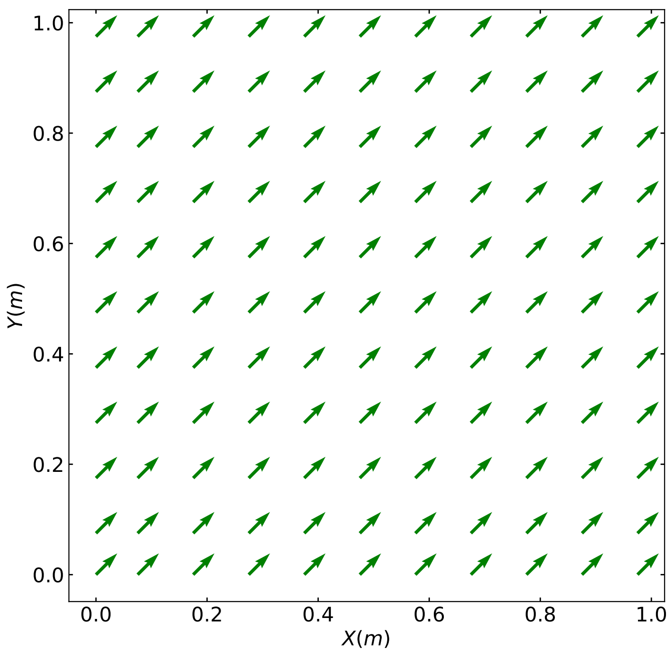


Figure 3: *Velocity field for the diagonal flow problem. Parameters:* $N = M = 20$.

2.2 Smith-Hutton

For the Smith-Hutton scenario, we consider a non-symmetrical box with dimensions $2L \times L$, as illustrated in Figure 4. This box is discretized into CVs, which have the same shape as depicted in Figure 1.

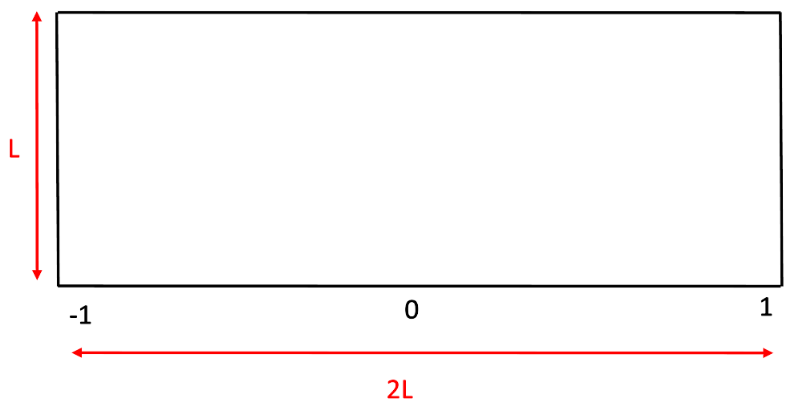


Figure 4: *A scheme of the Smith-Hutton case.*

For this case, the inlet of the problem goes from -1 to 0, while the outlet goes from 0 to 1. We impose Dirichlet boundary conditions on all three walls (left, top and right) and in the inlet. For the outlet we impose Neumann BCs. Specifically, on the left, top and right walls, we set $\phi = 1 - \tanh(10)$ at $(x = -1, y)$, $(-1 \leq x \leq 1, y = 1$ and $(x = 1, y)$, while for the inlet, we set $\phi = 1 + \tanh[10(2x + 1)]$ at $(-1 \leq x \leq 0, y = 0)$; and for the oulet $\partial\phi/\partial y = 0$ at $(0 < x < 1, y = 0)$.

In terms of the coefficients a_i, where $i \in [P, E, W, N, S]$, and b_p, the values at the left, top and right walls are assigned as follows: $a_E = a_W = a_N = a_S = 0$, $a_P = 1$, and $b_P = 1 - \tanh(10)$. Similarly, for the inlet, the coefficients are set as $a_E = a_W = a_N = a_S = 0$, $a_P = 1$, and $b_P = 1 + \tanh[10(2x + 1)]$, while for the outlet $a_E = a_W = a_S = 0$, $a_N = a_P = 1$, and $b_P = 0$.

The velocity field is described by the following equations:

$$v_x = 2y(1 - x^2) \; ; \quad v_y = -2x(1 - y^2) \tag{26}$$

The velocity field represented by Eq. (26) is visualized in Figure 5.

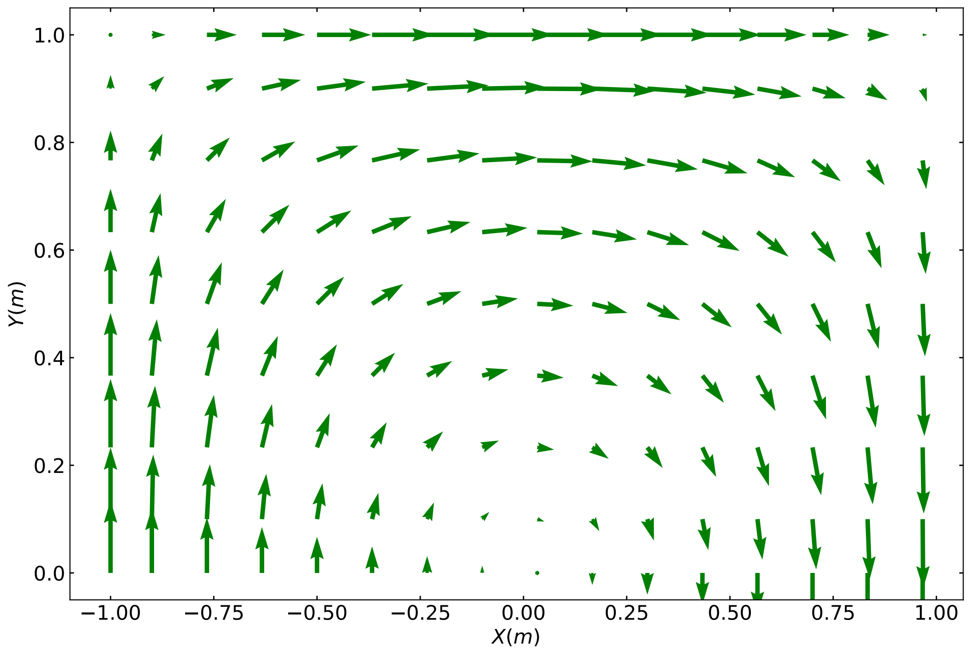


Figure 5: *Velocity field for the Smith-Hutton problem. Parameters:* $N = 30$ & $M = 15$.

3 Code Structure

In this section we introduce the code structure. As it will be shown in Section 7, the code is written for Python3 language. Further discussion on the code can be found in the repository linked below the **Index**.

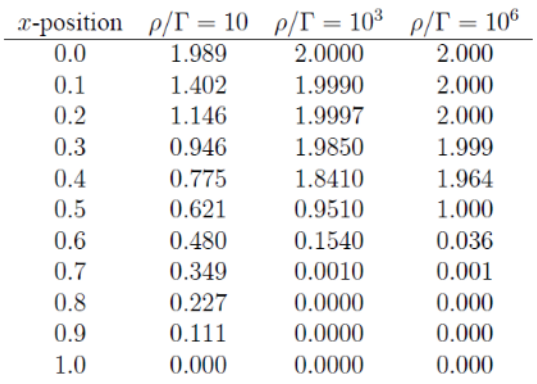
- Establish the input data, which is segregated into two distinct sections: one dedicated to the physical aspects of the problem encompassing parameters like the cavity's length and height (L and H), the input velocity v_{in}, thermodynamic variables such as initial temperature, pressure, and density (T_{in}, P_{in}, and ρ_{in}) and the ρ/Γ relation. The other is dedicated to numerical aspects as the CVs quantity (N and M) etc.
- Generate the mesh with $N \times M$ CVs over L and H.
- Define all the matrices with dimension of $(N + 2) \times (M + 2)$.
- Compute the velocity fields, which has been explained in Section 2.
- Initialize ϕ and ϕ^0 and evaluate all the internal nodes $a_i \;\; i \in [P, E, W, N, S]$ and b_P (here we chose the evaluation scheme). In addition to this we compute the BCs.
- Evaluation of the new time steps: $t = t + \Delta t$.
- Solve the Gauss-Seidel algorithm for all internal nodes.
- Ask for new time step if needed (is $|\phi - \phi^0 < \varepsilon|$? if answer is NO we need a new time step).
- Final calculations and create all the plots and save them.

4 Validation and Verification of the Code

Before presenting the results obtained in this work, it is crucial to conduct an analysis to ensure the code's proper functionality.

4.1 Comparison With Reference Solution

The presentation slides in our class included reference values, as presented in Fig. 6, for the outlet solution of ϕ in the Smith-Hutton problem across various ρ/Γ values. Therefore, it is important to conduct a comparative analysis between our solutions for the outlet values using different schemes (UDS and EDS) and the provided reference values.



x-position	$\rho/\Gamma = 10$	$\rho/\Gamma = 10^3$	$\rho/\Gamma = 10^6$
0.0	1.989	2.0000	2.000
0.1	1.402	1.9990	2.000
0.2	1.146	1.9997	2.000
0.3	0.946	1.9850	1.999
0.4	0.775	1.8410	1.964
0.5	0.621	0.9510	1.000
0.6	0.480	0.1540	0.036
0.7	0.349	0.0010	0.001
0.8	0.227	0.0000	0.000
0.9	0.111	0.0000	0.000
1.0	0.000	0.0000	0.000

Figure 6: *Reference values of ϕ for the outlet of the Smith-Hutton problem for different ρ/Γ values.*

Firstly, in Fig. 7 we show the different values of ϕ in the outlet for $\rho/\Gamma = 10, 10^3$ & 10^6 for UDS. We can observe that the main discrepancies occur at the upper and lower values of $\rho/\Gamma = 10^3$ & 10^6 and at the initial values for $\rho/\Gamma = 10$. This errors might be corrected with a bigger mesh and it will be analysed in Section **4.2**.

Next, we replicate the Figure 7 using the EDS approach, as depicted in Fig. 8a. If we compare them, we can observe that there is minimal disparity between the results obtained with UDS and EDS. In Fig. 8b, we extend this comparison to $\rho/\Gamma = 10^3$. Both curves exhibit striking similarity, with the EDS curve consistently being closer to the reference curve. To offer a clearer perspective, we provide zoomed-in views of the upper and lower sections in Fig. 8c and 8d, respectively.

It is useful to put all these solutions into numbers. In Table 6 we compare the errors of ϕ in the outlet for different ρ/Γ values between EDS and UDS. It can be seen that we obtain smaller errors for EDS than UDS for $\rho/\Gamma = 10$ & 10^3, while we obtain the same error for $\rho/\Gamma = 10^6$ (there is a difference for the fourth decimal, having EDS the smallest error).

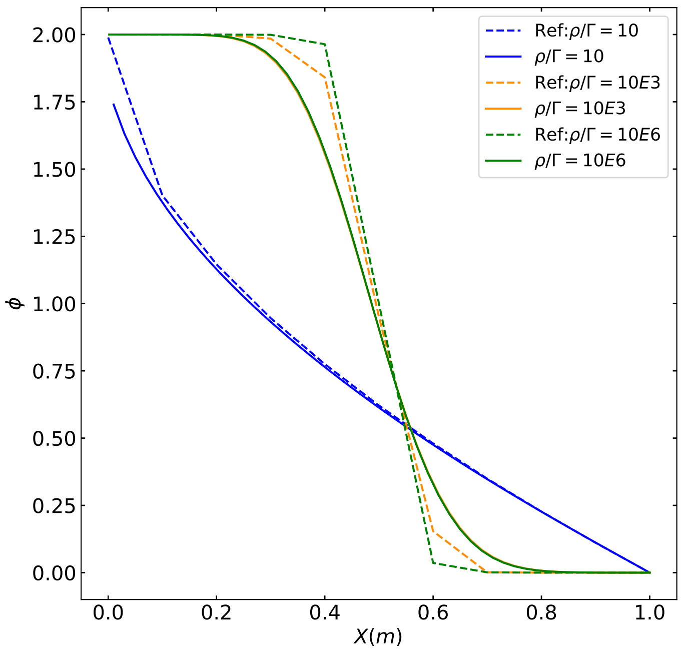


Figure 7: *Values of ϕ for the outlet of the Smith-Hutton problem for different ρ/Γ values (solid lines) and the reference values (dashed lines). Here we make use of UDS and $N = 100$ & $M = 50$.*

ρ/Γ	**Error**^{UDS} (%)	**Error**^{EDS}(%)
10	8.46	7.97
10^3	3.48	3.42
10^6	4.02	4.02

Table 1: *Relative errors of the outlet values of ϕ respect to the reference for both schemes, UDS and EDS.*

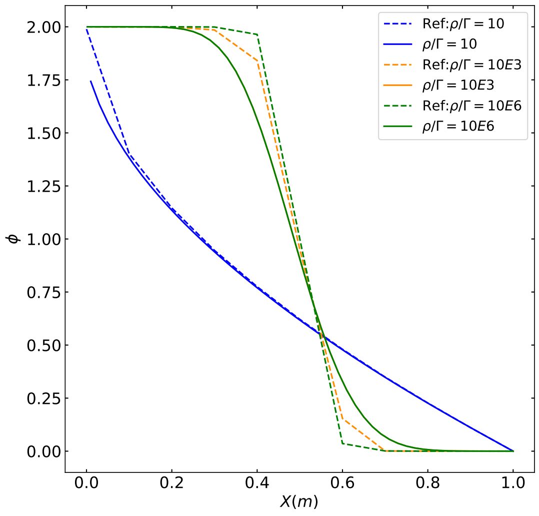
4.2 Mesh Refinement

Another method to confirm that the code functions correctly involves observing that simulations with larger meshes should exhibit smaller errors respect to the reference. To demonstrate this, we generated Table 2, 3 and 4 for the three ρ/Γ values and for various mesh configurations: $N = 20$ & $M = 10$, $N = 50$ & $M = 25$, $N = 100$ & $M = 50$, and $N = 200$ & $M = 100$. The results are shown in terms of UDS and EDS.

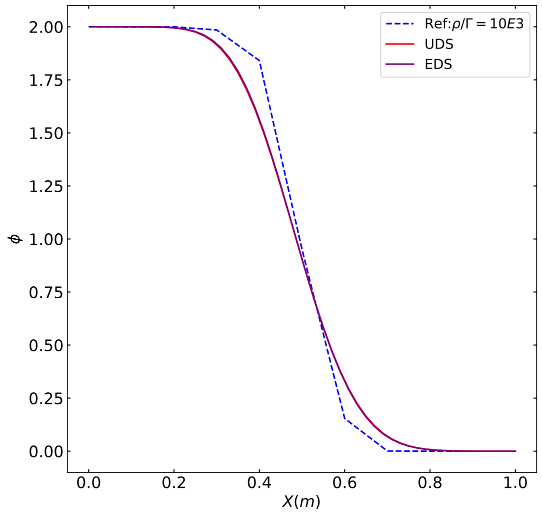
Mesh	**Error**^{UDS}(%)	**Error**^{EDS}(%)
$N = 20$ & $M = 10$	20.41	18.37
$N = 50$ & $M = 25$	11.61	10.69
$N = 100$ & $M = 50$	8.46	7.97
$N = 200$ & $M = 100$	6.83	6.31

Table 2: *Relative errors of the oulet values of ϕ respect to the reference for both schemes, UDS and EDS. Here we make use of $\rho/\Gamma = 10$.*

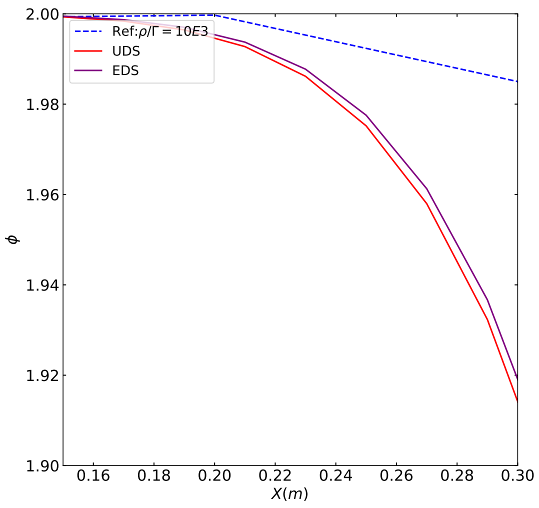
The tendency is quite clear: for bigger meshes we obtain smaller relative errors respect to the reference values. In addition to this, the errors of EDS are smaller respect to UDS. There



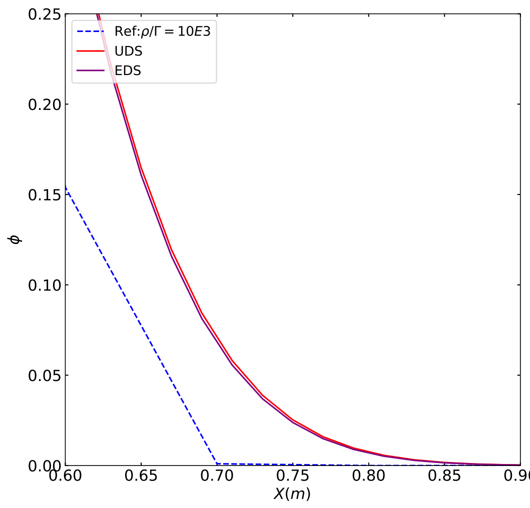
(a) *EDS with $N = 100$ & $M = 50$.*



(b) *EDS and UDS with $N = 100$ & $M = 50$.*



(c) *EDS and UDS with $N = 100$ & $M = 50$.*



(d) *EDS and UDS with $N = 100$ & $M = 50$.*

Figure 8: *Values of ϕ for the outlet of the Smith-Hutton problem in **a)** for different ρ/Γ values (solid lines) and the reference values (dashed lines), in **b)** the comparison between UDS and EDS for $\rho/\Gamma = 10^3$, in **c)** a zoom into the upper part and in **d)** a zoom into the lower part. Here we make use of EDS and $N = 100$ & $M = 50$.*

Mesh	**Error**^{UDS}(%)	**Error**^{EDS}(%)
$N = 20$ & $M = 10$	18.18%	18.11
$N = 50$ & $M = 25$	7.36	7.30
$N = 100$ & $M = 50$	3.48	3.42
$N = 200$ & $M = 100$	1.47	1.41

Table 3: *Relative errors of the oulet values of ϕ respect to the reference for both schemes, UDS and EDS. Here we make use of $\rho/\Gamma = 10^3$.*

is an exception, for $\rho/\Gamma = 10^6$ the relative errors are practically the same for all the meshes.

Mesh	**Error**^{UDS}(%)	**Error**^{EDS}(%)
$N = 20$ & $M = 10$	18.63	18.63
$N = 50$ & $M = 25$	7.88	7.88
$N = 100$ & $M = 50$	4.02	4.02
$N = 200$ & $M = 100$	2.03	2.03

Table 4: *Relative errors of the oulet values of ϕ respect to the reference for both schemes, UDS and EDS. Here we make use of $\rho/\Gamma = 10^6$.*

It is also noteworthy to discuss the required computational time for each problem to reach convergence. In both cases (Diagonal Flow and Smith-Hutton), we have noticed that a larger mesh size leads to an increased computational time, as it is expected. Furthermore, we have observed that smaller Peclet numbers (as well as lower values of ρ/Γ) requires a longer computational time. Additionally, when comparing both schemes, we have found that the EDS demands slightly more computational time compared to the UDS method.

5 Results

Here we present the results obtained for the Diagonal Flow and Smith-Hutton problems. As it have been shown in the above section, EDS is a little bit more precise. Therefore, the presented results will be computed using EDS.

5.1 Diagonal Flow

Starting with the Diagonal Flow problem, in Table 5 we show the physical and numerical values used to present the results.

The primary results are depicted in Fig. 9. In this figure, we illustrate the variations in ϕ for different Peclet numbers. It is evident that there are two distinct regions: an upper section closely aligned with ϕ_{high} and a lower section closely aligned with ϕ_{low}, with a transitional diffusion zone in between. As we increase the Peclet number, the extent of diffusion decreases. This trend can be attributed to the direct relationship between the Peclet number and the diffusion term. In an ideal scenario (free from numerical errors), as $Pe \to \infty$, we would expect the elimination of diffusion in our solution, resulting in the upper-mid region being filled with ϕ_{high} values and the lower-mid region populated with ϕ_{low} values. Consequently, our solution aligns with this expectation.

Moreover, we have observed in our results that for large Peclet values, there exists a point beyond which we cannot further reduce the diffusion of the solution (or the change is very small). This limitation arises due to numerical errors, introducing false diffusion.

Name	Symbol	Value	Units
X Direction Mesh	N	140	#
Y Direction Mesh	M	N	#
G-S/Time Convergence Crit.	ε	10^{-6}	#
Time Step	Δt	0.1	s
Cavity Length	L	1	m
Cavity Height	H	1	m
Input Temperature	T_{in}	298	K
Input Pressure	P_{in}	1.013×10^5	N / m^2
Input Velocity	V_{in}	5.15	m / s
Input Density	ρ_{in}	1.18	kg / m^3
Peclet number	Pe	$[1 - 10^7]$	#
ϕ top and left walls	ϕ_{high}	1.0	#
ϕ bottom and right walls	ϕ_{low}	0.0	#

Table 5: *Used numerical and physical input data for simulations.*

Notably, while our results are presented in terms of EDS, the same behavior is observed when employing UDS.

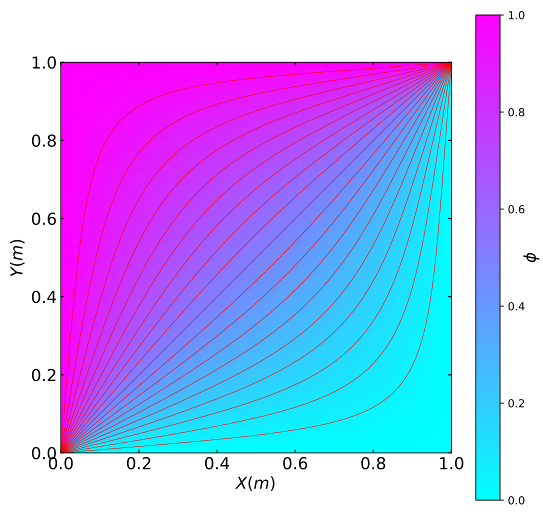
5.2 Smith-Hutton

Continuing with the Smith-Hutton problem, in Table 5 we show the physical and numerical values used to present the results.

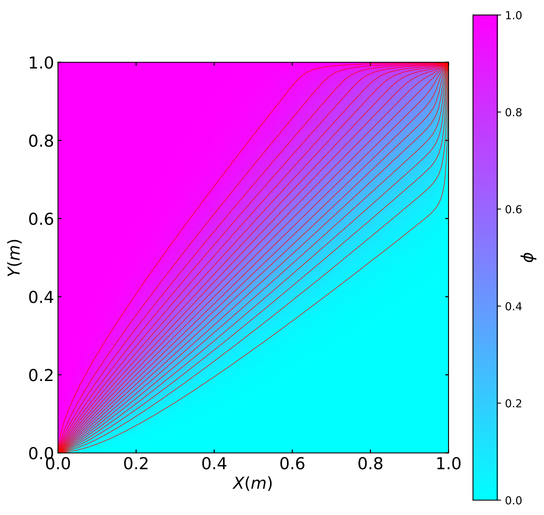
Name	Symbol	Value	Units
X Direction Mesh	N	160	#
Y Direction Mesh	M	$\frac{N}{2}$	#
G-S/Time Convergence Crit.	ε	10^{-6}	#
Time Step	Δt	0.1	s
Cavity Length	L	2	m
Cavity Height	H	1	m
Input Temperature	T_{in}	298	K
Input Pressure	P_{in}	1.013×10^5	N / m^2
Input Velocity	V_{in}	5.15	m / s
Input Density	ρ_{in}	1.18	kg / m^3

Table 6: *Used numerical and physical input data for simulations.*

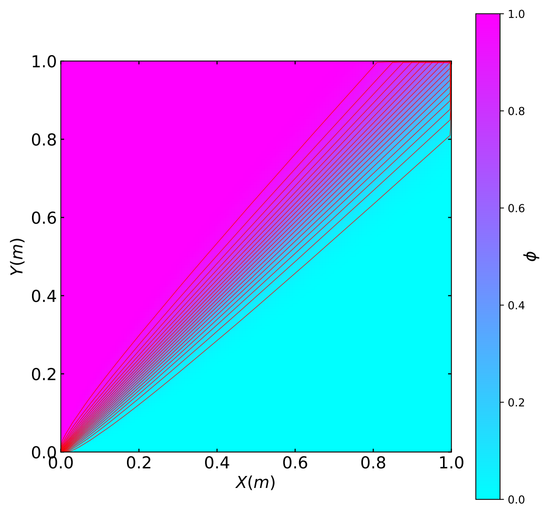
The main results are presented in Fig. 10. In this figure, we depict the variations in ϕ for three different ρ/Γ values. As expected, the variation of ϕ is observed from the inlet to the outlet. Notably, for the smallest ρ/Γ value, we obtain the least symmetrical results, indicating a significant presence of diffusion, meaning that the convective term is more



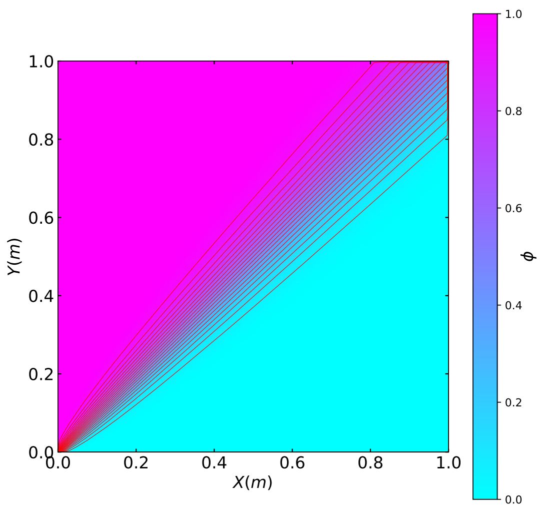
(a) $Pe = 1$



(b) $Pe = 100$



(c) $Pe = 10^5$.

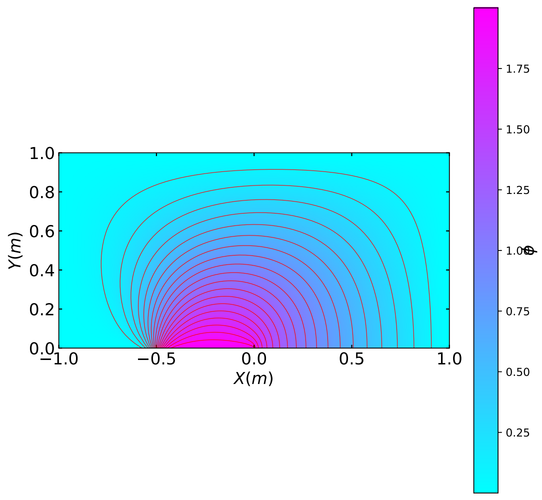


(d) $Pe = 10^7$

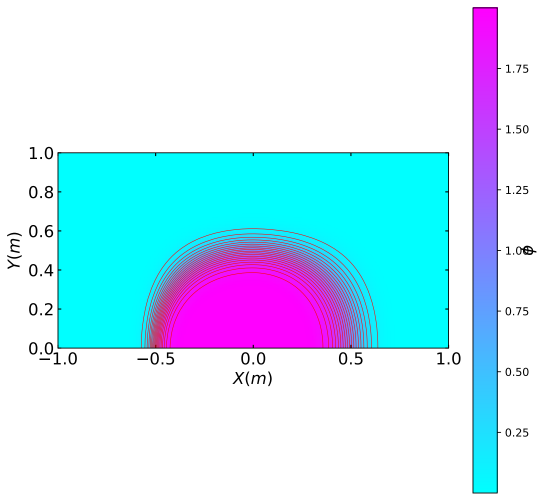
Figure 9: *Values of ϕ for the Diagonal problem in **a)** for $Pe = 1$, in **b)** for $Pe = 100$, in **c)** for $Pe = 10^5$ and in **d)** for $Pe = 10^7$. Here we make use of EDS. All the used parameters are shown in Table 5.*

dominant for low Peclet values. With an increase in the ρ/Γ value, the solution becomes more symmetrical, although some diffusion is still evident. This means that the diffusive term decreases with increasing ρ/Γ values. Similar to the Diagonal Flow problem, we observe false diffusion caused by numerical errors for the biggest ρ/Γ value. The same behavior is observed when using UDS.

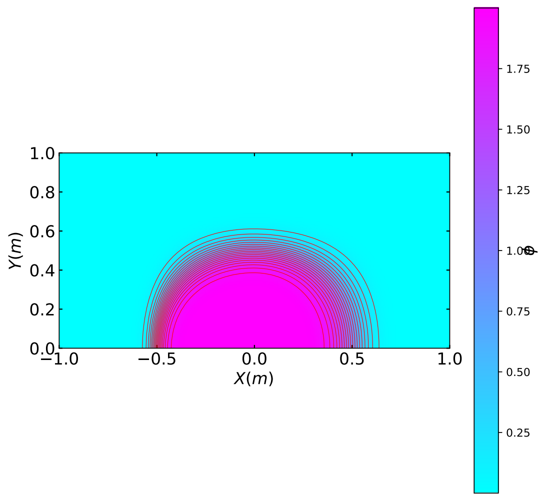
For a better observation of the results we also show them in a 3D plot in Fig. 11.



(a) $\rho/\Gamma = 10$



(b) $\rho/\Gamma = 10^3$



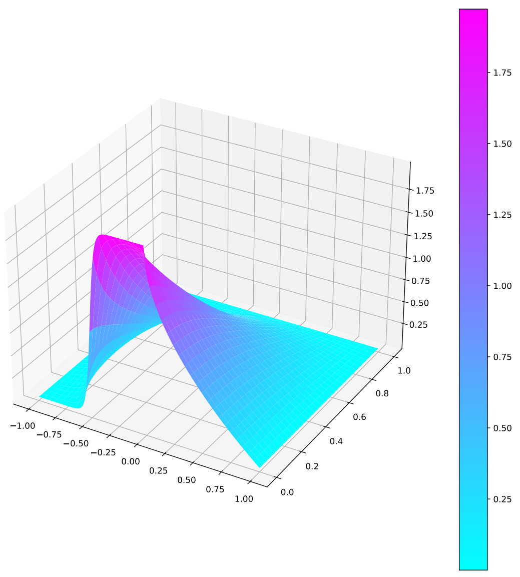
(c) $\rho/\Gamma = 10^6$

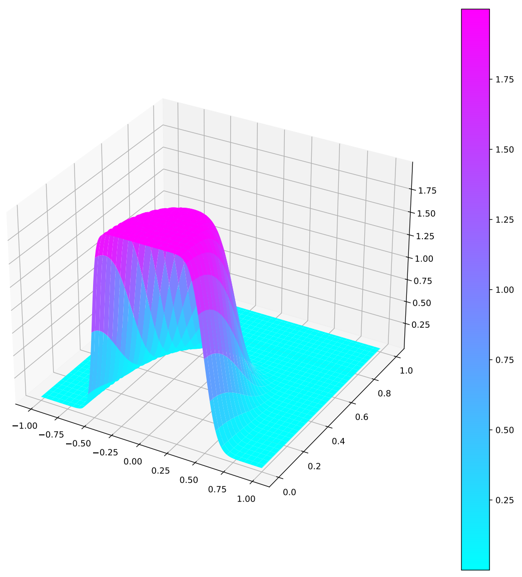
Figure 10: *Values of ϕ for the Smith Hutton problem in **a)** for $\rho/\Gamma = 10$, in **b)** for $\rho/\Gamma = 10^3$, and in **c)** for $\rho/\Gamma = 10^6$. Here we make use of EDS. All the used parameters are shown in Table 6.*

6 Conclusions

In this work we have analysed the generic convection-diffusion equation for two problems: Diagonal Flow and Smith-Hutton problems, focusing on the comparison with reference solutions and mesh refinement.

In the comparison with the reference solution for the Smith-Hutton problem, it was observed that for $\rho/\Gamma = 10$ and 10^3, both the UDS and EDS approaches exhibited small errors compared to the reference, with EDS consistently outperforming UDS. For $\rho/\Gamma = 10^6$, the errors were very similar (same up to fourth decimal) between EDS and UDS. The results confirmed that the code could reproduce the reference values.


(a) $\rho/\Gamma = 10$


(b) $\rho/\Gamma = 10^3$

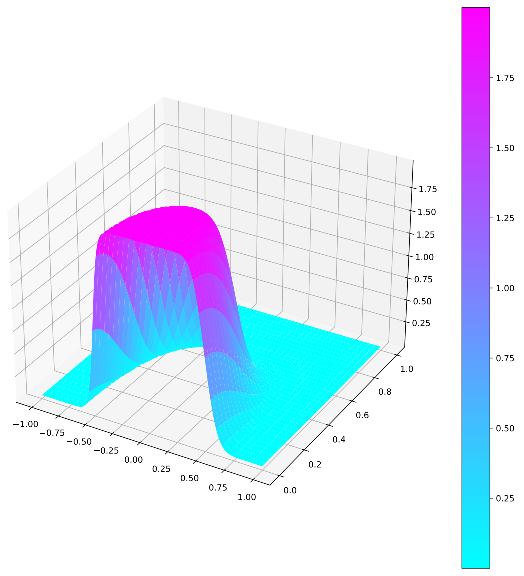

(c) $\rho/\Gamma = 10^6$

Figure 11: *Values of ϕ projected in Z direction for the Smith-Hutton problem in **a)** for $\rho/\Gamma = 10$, in **b)** for $\rho/\Gamma = 10^3$, and in **c)** for $\rho/\Gamma = 10^6$. Here we make use of EDS. All the used parameters are shown in Table 6.*

Mesh refinement studies demonstrated that simulations with larger meshes resulted in smaller errors relative to the reference, as expected. Larger meshes reduced the relative errors, and EDS consistently outperformed UDS in terms of accuracy, except the $\rho/\Gamma = 10^6$, which showed same results (again up to fourth decimal).

Additionally, a comment on the computational time for both schemes was performed. It was found that larger meshes required more computational time, while larger ρ/Γ values reduced the computation time. UDS was generally faster than EDS, which could be advantageous for larger simulations.

In the results section, we presented the results for both the Diagonal Flow and Smith-Hutton problems, focusing on the EDS approach due to its higher precision.

For the Diagonal Flow problem, simulations were conducted across a range of Peclet numbers. The results showed that as the Peclet number increased, the diffusion decreased. However, for very high Peclet numbers, numerical errors introduced a false diffusion component.

In the Smith-Hutton problem, the focus was on different ρ/Γ values. It was observed that for smaller ρ/Γ values, the solution exhibited less symmetry, indicating a significant presence of diffusion. With increasing ρ/Γ, the solution became more symmetrical, but with some remaining diffusion. Numerical errors introduced false diffusion for the largest ρ/Γ values.

References

[1] S. Patankar, *Numerical heat transfer and fluid flow.* Taylor & Francis, 2018.

7 Appendix: Python3 Code

Here we show the code in Python3 language for the Smith-Hutton problem. The rest of the code will be attached in a .zip in atenea. Also it can be found in the link below the **Index** of this report.

```python
#========================================
# Code for the Smith-Hutton Case
# Master in Space and Aeronautical
# Engineering.
# Computational Engineering: Assginament
# 2.
# Author: Inaki Fernandez Tena
# email: inakiphy@gmail.com
#========================================

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

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


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

L        = 2.0                          # Channel lenght.
H        = L/2.0                        # Channel height.
Z        = 1.0                           # Channel depth.
V_in     = 1.15                          # Air velocity in m/s.
T_in     = 298.0                         # Normal air temperature in K.
P_in     = 1.013E05                      # Air pressure at sea level in N/m^2.
R        = 287                           # Ideal gas constant in J/mol*K
rho_in   = P_in / (R * T_in)             # Air density at sea level in kg/m^3.
    Incompresible = constant density at all mesh points.
delta_t  = 0.1
rhoGamma = 10E6
Gamma    = rho_in / rhoGamma
phi_in   = 1.0

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

N          = 200                          # Control volumes in x direction.
M          = int(N / 2)                   # Control volumes in y direction.
delta_X    = L / N                        # Control volume lenght in x direction.
delta_Y    = H / M                        # Control volume height in y direction.
eps        = 1.0E-06                      # Gauss-Seidel method convergence parameter.
t_max      = 1000000                      # G-S loop steps.
time_max   = 1000000                      # Time loop steps.
delta_Z    = 1.0                          # Just one step into z direction

```

```python

# Mesh generation

x_cv = np.linspace(-L/2.0, L/2.0, N+1)              # Generate x points with the same spacing
y_cv = np.linspace(0, H, M+1)               # Generate y points with the same spacing
x_p  = np.zeros(N+2)                        # Vectors for the centered control volumes with N+2 elements
y_p  = np.zeros(M+2)                        # Vectors for the centered control volumes with M+2 elements

# Fill x_p and y_p

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

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

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

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

phi       = np.zeros((M+2,N+2))            # Phi function
phi_zero  = np.zeros((M+2,N+2))            # Phi at t=0 function
phi_ax    = np.zeros((M+2,N+2))            # Phi auxiliar for G-S
rho       = np.zeros((M+2,N+2))            # Density matrix
a_P       = np.zeros((M+2,N+2))            # Auxiliar matrix at point p
a_E       = np.zeros((M+2,N+2))            # Auxiliar matrix at point east
a_S       = np.zeros((M+2,N+2))            # Auxiliar matrix at point south
a_W       = np.zeros((M+2,N+2))            # Auxiliar matrix at point west
a_N       = np.zeros((M+2,N+2))            # Auxiliar matrix at point north
b_P       = np.zeros((M+2,N+2))            # Generation term
v_xP      = np.zeros((M+2,N+2))            # Velocity in x direction
v_yP      = np.zeros((M+2,N+2))            # Velocity in y direction
m_e       = np.zeros((M+2,N+2))            # east mass flow rate
m_s       = np.zeros((M+2,N+2))            # south mass flow rate
m_w       = np.zeros((M+2,N+2))            # west mass flow rate
m_n       = np.zeros((M+2,N+2))            # north mass flow rate
D_e       = np.zeros((M+2,N+2))
D_w       = np.zeros((M+2,N+2))
D_n       = np.zeros((M+2,N+2))
D_s       = np.zeros((M+2,N+2))


#========================================
# Define velocities
#========================================
for i in range(N+2):
    for j in range(M+2):
        v_xP[j,i] = 2.0 * y_p[j] * (1.0 - x_p[i]**2)
        v_yP[j,i] = -2.0 * x_p[i] * (1.0 - y_p[j]**2)

#========================================
# Initialize the map of \phi_{0} and \phi
#=======================================
# Internal nodes
for i in range(1, N+1):
    for j in range(1, M+1):
        phi_zero[j,i] = phi_in
        phi[j,i] = phi_in

# Inlet nodes
for i in range(int((N+2) / 2)):
    for j in range(M+2):
        phi_zero[0,i] = 1.0 + np.tanh(10.0 * (2.0 * x_p[i] + 1.0))
        phi[0,i] = 1.0 + np.tanh(10.0 * (2.0 * x_p[i] + 1.0))
# Walls
for i in range(N+2):
    for j in range(M+2):
        phi_zero[j,0]  = 1.0 - np.tanh(10.0)
        phi_zero[-1,i] = 1.0 - np.tanh(10.0)
        phi_zero[j,-1] = 1.0 - np.tanh(10.0)
        phi[j,0]  = 1.0 - np.tanh(10.0)
        phi[-1,i] = 1.0 - np.tanh(10.0)
        phi[j,-1] = 1.0 - np.tanh(10.0)

#=======================================
# Compute mass flow rates
#=======================================
for i in range(1,N+1):
    for j in range(1,M+1):
        m_e[j,i] = rho_in * (v_xP[j,i + 1] + v_xP[j,i]) * delta_Y * delta_Z / 2.0
        m_w[j,i] = rho_in * (v_xP[j,i - 1] + v_xP[j,i]) * delta_Y * delta_Z / 2.0
        m_n[j,i] = rho_in * (v_yP[j + 1,i] + v_yP[j,i]) * delta_X * delta_Z / 2.0
        m_s[j,i] = rho_in * (v_yP[j - 1,i] + v_yP[j,i]) * delta_X * delta_Z / 2.0

#=======================================
# Compute Di
#=======================================
# Internal nodes
for i in range(1, N+1):
    for j in range(1, M+1):
        D_e[j,i] = Gamma * delta_Y * delta_Z / np.abs(x_p[i] - x_p[i + 1])
        D_w[j,i] = Gamma * delta_Y * delta_Z / np.abs(x_p[i] - x_p[i - 1])
        D_n[j,i] = Gamma * delta_X * delta_Z / np.abs(y_p[j] - y_p[j + 1])
        D_s[j,i] = Gamma * delta_X * delta_Z / np.abs(y_p[j] - y_p[j - 1])

# Internal nodes
for i in range(1, N+1):
    for j in range(1,M+1):
        #Pe_e = m_e[j,i] / D_e[j,i] # For EDS
        #Pe_w = m_w[j,i] / D_w[j,i] # For EDS
        #Pe_n = m_n[j,i] / D_n[j,i] # For EDS
        #Pe_s = m_s[j,i] / D_s[j,i] # For EDS
        #a_E[j,i] = D_e[j,i] * (np.abs(Pe_e) / (np.exp(np.abs(Pe_e)) - 1.0)) - ((m_e[j,i]
             - np.abs(m_e[j,i])) / 2.0) # For EDS
        #a_W[j,i] = D_w[j,i] * np.abs(Pe_w) / (np.exp(np.abs(Pe_w)) - 1.0) + ((m_w[j,i] +
             np.abs(m_w[j,i])) / 2.0) # For EDS
        #a_N[j,i] = D_n[j,i] * np.abs(Pe_n) / (np.exp(np.abs(Pe_n)) - 1.0) - ((m_n[j,i] -
             np.abs(m_n[j,i])) / 2.0) # For EDS
        #a_S[j,i] = D_s[j,i] * np.abs(Pe_s) / (np.exp(np.abs(Pe_s)) - 1.0) + ((m_s[j,i] +
             np.abs(m_s[j,i])) / 2.0) # For EDS
        a_E[j,i] = D_e[j,i] - ((m_e[j,i] - np.abs(m_e[j,i])) / 2.0)
        a_W[j,i] = D_w[j,i] + ((m_w[j,i] + np.abs(m_w[j,i])) / 2.0)
        a_N[j,i] = D_n[j,i] - ((m_n[j,i] - np.abs(m_n[j,i])) / 2.0)
        a_S[j,i] = D_s[j,i] + ((m_s[j,i] + np.abs(m_s[j,i])) / 2.0)
        a_P[j,i] = a_E[j,i] + a_W[j,i] + a_N[j,i] + a_S[j,i] + ((rho_in * delta_X *
            delta_Y * delta_Z) / delta_t)

# Inlet DIRICHLET
for i in range(int((N+2) / 2)):
    for j in range(M+2):
        a_E[0,i]  = 0.0
        a_W[0,i]  = 0.0
        a_N[0,i]  = 0.0
        a_S[0,i]  = 0.0
        b_P[0,i]  = 1 + np.tanh(10.0 * (2.0 * x_p[i] + 1.0))
        a_P[0,i]  = 1.0

# Outlet NEUMANN
for i in range(int((N+2) / 2), N+2):
    for j in range(M+2):
        a_E[0,i]  = 0.0
        a_W[0,i]  = 0.0
        a_N[0,i]  = 1.0
        a_S[0,i]  = 0.0
        b_P[0,i]  = 0.0
        a_P[0,i]  = 1.0
# Walls Dirichlet
for i in range(N+2):
    for j in range(M+2):
        a_E[j,0]  = 0.0                     # Left
        a_W[j,0]  = 0.0                     # Left
        a_N[j,0]  = 0.0                     # Left
        a_S[j,0]  = 0.0                     # Left
        b_P[j,0]  = 1.0 - np.tanh(10.0)     # Left
        a_P[j,0]  = 1.0                     # Left
        a_E[j,-1] = 0                       # Right
        a_W[j,-1] = 0                       # Right
        a_N[j,-1] = 0                       # Right
        a_S[j,-1] = 0                       # Right
        a_P[j,-1] = 1                       # Right
        b_P[j,-1] = 1.0 - np.tanh(10.0)     # Right
        a_E[-1,i] = 0.0                     # Top nodes
        a_W[-1,i] = 0.0                     # Top nodes
        a_N[-1,i] = 0.0                     # Top nodes
        a_S[-1,i] = 0.0                     # Top nodes
        a_P[-1,i] = 1.0                     # Top nodes
        b_P[-1,i] = 1.0 - np.tanh(10.0)     # Top nodes

print("=========================")
print("Starting loops")
print("=========================")
#=======================================
# Begin time step (t+delta_t)
#=======================================
for time in range(time_max):
    r2 = np.sum(phi_zero)                         # Sum the values for t = 0
    #=======================================
    # Begin Gauss-Seidel
    #=======================================
    phi_ax = phi_zero                               # Set the initial value of the axuliary
         stream function
    for t in range(t_max):
        # phi_zero outlet NEUMANN
        for i in range(int((N+2) / 2), N+2):
            for j in range(M+2):
                phi_ax[0,i] = phi_ax[1,i]
        # Inlet nodes of b_p
        for i in range(1, N+1):
            for j in range(1, M+1):
                 b_P[j,i] = ((rho_in * delta_X * delta_Y * delta_Z) / delta_t) * phi_ax[j
                     ,i]

        r = np.sum(phi_ax)                          # Sum the values of the psi auxiliary
            matrix
        for i in range(1, N+1):
            for j in range(1, M+1):
                phi[j, i] = (a_E[j, i] * phi_ax[j, i + 1] + a_W[j, i] * phi_ax[j, i - 1]
                    + a_N[j, i] * phi_ax[j + 1, i] + a_S[j,i] * phi_ax[j - 1, i] + b_P[j,
                     i]) / a_P[j, i]

        sum = np.sum(phi)                          # Sum the values of psi matrix
        if np.abs(sum-r) <= eps:                   # Watch if the |psi - psi_aux| <
            precision
            print("=======================")
            print("G-S algorithm converged")
            print("The requiered steps has been:" +  " " + str(t))
            print("=======================")
            break                                  # If corveges just break the main G-S
                loop
        else:
            phi_ax = phi
    sum2 = np.sum(phi)                             # Sum the values for i+1
    if np.abs(sum2 - r2)<=eps:
        print("=======================")
        print("Time loop converged")
        print("The requiered steps has been:" +  " " + str(time))
        print("=======================")
        break                                  # If corveges just break the main G-S loop
    else:
        phi_zero = phi
# Vectors for outlet plot

x_plot_exp = [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
phi_10     = [1.989, 1.402, 1.146, 0.946, 0.775, 0.621, 0.480, 0.349, 0.227, 0.111,
    0.000]
phi_10E3   = [2.0000, 1.9990, 1.9997, 1.9850, 1.8410, 0.9510, 0.1540, 0.0010, 0.0000,
    0.0000, 0.0000]
phi_10E6   = [2.000, 2.000, 2.000, 1.999, 1.964, 1.000, 0.036, 0.001, 0.000, 0.000,
    0.000]

x_plot   = np.zeros(int((N+2) / 2))
phi_plot = np.zeros(int((N+2) / 2))

# Write the outlet values

# Path to output file
file_output = f"Results/S-H/Outlet_{N}x{M}_{rhoGamma}.dat"

# Generate vectors to write
for i in range(int((N+2) / 2), N+2):
    for j in range(M+2):
        x_plot[i - int((N+2) / 2)]    = x_p[i]
        phi_plot[i- int((N+2) / 2)]   = phi[0,i]

# Write

matrix = np.column_stack((x_plot,phi_plot))

# Save

np.savetxt(file_output, matrix, fmt='%.10f', delimiter='\t')
#========================================
# Generate the plot
#========================================
output_plot1 = f"Results/S-H/Phi_{N}x{M}_{rhoGamma}.pdf"
output_plot2 = f"Results/S-H/Vel_{N}x{M}_{rhoGamma}.pdf"
output_plot3 = f"Results/S-H/Phi3D_{N}x{M}_{rhoGamma}.pdf"
print(" ")
print("=================")
print("Starting the plot")
print(" ")
print("It might take a few seconds")
print("===========================")

# Figure specifications
fontsize=15

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

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

# Plot
plt.contour(x_p, y_p, phi, levels=20, colors='r', linewidths=0.5)
plt.imshow(phi, cmap= 'cool', extent=(x_p.min(), x_p.max(), y_p.min(), y_p.max()), origin
    ='lower')
plt.colorbar()

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

# Add text next to the color bar
text_x = 1.18  # Adjust the x-coordinate as needed
text_y = 0.5   # Adjust the y-coordinate as needed
text = r"$\phi$"
ax.text(text_x, text_y, text, transform=ax.transAxes, rotation=270, va='center', fontsize
    =fontsize)

# Save figure
plt.savefig(output_plot1,bbox_inches='tight')

# Start second plot
plt.figure(2)
plt.figure(figsize = (12,8))
plt.tick_params(axis='both', which='both',length=3, width=1.0,
labelsize=15, right=True, top=True, direction='in') # For ticks in borders
```

```python

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

# Plot
P,Z = np.meshgrid(x_p, y_p)
plt.quiver(P[::2, ::2], Z[::2, ::2], v_xP[::2, ::2], v_yP[::2, ::2], color="green")


# Save figure
plt.savefig(output_plot2,bbox_inches='tight')


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

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

# Plot
# Create a 3D figure
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(111, projection='3d')

# Create the surface plot
surf = ax.plot_surface(P, Z, phi, cmap='cool')

# Add a colorbar
cbar = fig.colorbar(surf)

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


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