UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Universitat Politècnica de Catalunya

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

ASSIGNMENT 2: GENERIC CONVECTION-DIFFUSION TRANSPORT EQUATION

Computational Engineering - Space and Aeronautical Engineering MSc

Author:
Jorge SIMÓN AZNAR

Contents

1 Proposal

This task involves examining the numerical solving of the convection-diffusion equation. After the code is completed, we will calculate the solution for the Smith-Hutton and diagonal flow problems to validate it.

We will analyze this scenario under steady, two-dimensional, and velocity-known circumstances within a rectangular area defined by length L and height H.

2 Introduction

In this study, our objective is to numerically solve the general convection-diffusion equation. We'll commence our discussion by introducing the Navier-Stokes (N-S) equations. When dealing with ideal gases, the N-S equations can be represented as [**?**]:

$$\frac{\partial(\rho\mathbf{v})}{\partial t}+\nabla(\rho\mathbf{v})=0 \tag{2.1}$$

$$\frac{\partial(\rho\mathbf{v})}{\partial t}+\nabla(\rho\mathbf{v}\mathbf{v})=\nabla(\mu\nabla\mathbf{v})+[\nabla(\tau-\mu\nabla\mathbf{v})-\nabla p+\rho\mathbf{g}] \tag{2.2}$$

$$\frac{\partial(\rho T)}{\partial t}+\nabla(\rho\mathbf{v}T)=\nabla\left(\frac{\lambda}{c_v}\nabla T\right)+\left[\frac{-\nabla\mathbf{q}^R-p\nabla\mathbf{v}+\tau:\nabla\mathbf{v}}{c_v}\right] \tag{2.3}$$

$$\frac{\partial(\rho Y_k)}{\partial t}+\nabla(\rho\mathbf{v}Y_k)=\nabla(\rho D_{km}\nabla Y_k)+(\dot{\omega}_k) \tag{2.4}$$

All these transport equations encompass terms for unsteady, convection, diffusion, and additional contributions. The general convection-diffusion equation for a generic variable ϕ can be formulated as:

$$\frac{\partial(\rho\phi)}{\partial t}+\nabla(\rho\mathbf{v}\phi)=\nabla(\Gamma_\phi\nabla\phi)+\dot{s_\phi}, \tag{2.5}$$

Here, Γ_ϕ and $\dot{s_\phi}$ represent the diffusion coefficient and the supplementary source/sink term, respectively.

Then, we can utilize the mass conservation equation (2.1) to derive the equivalent convection-diffusion equation:

$$\rho\frac{\partial\phi}{\partial t}+\rho\mathbf{v}\nabla\phi=\nabla(\Gamma_\phi\nabla\phi)+\dot{s_\phi}. \tag{2.6}$$

To perform numerical solutions, we will employ the finite volume method. This technique involves dividing our computational domain into smaller control volumes (CVs). In Figure 2.1, we can observe a visual representation of a CV along with its adjacent CVs.

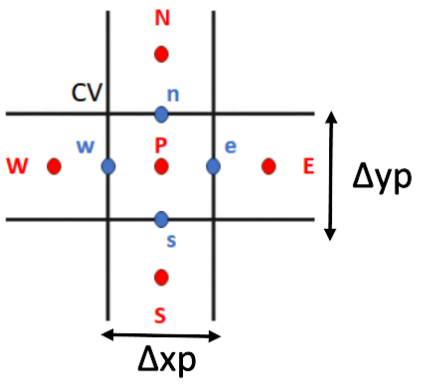


Figure 2.1: Scheme of a CV and it's neighbours

Each CV is bounded by four walls (blue points), each hosting a node: one to the east (e), one to the north (n), one to the west (w), and one to the south (s). In the heart of the CV, we locate a node denoted as P. Surrounding this CV, there are four neighboring CVs (red points), each with its central node identified as East (E), North (N), West (W), and South (S). To specify the dimensions of the CV, we denote its length in the X direction as Δxp and its height in the Y direction as Δyp.

As evident from equation (2.6), there exists a time partial derivative, indicating the need for time discretization. This can be accomplished by dividing the time domain into small time steps of Δt, such that the time progression is given as $t = t_0 + \Delta t \times i$, where i represents the total number of time steps.

Let's consider the integral form of the mass conservation equation (2.1):

$$\frac{\partial}{\partial t}\int_{V_P} \rho dV + \int_{S_f} \rho \mathbf{vn} dS = 0. \tag{2.7}$$

Here, V_P represents the CV volume. Focusing on the central point P (refer to Fig. 2.1), the mass conservation equation can be semi-discretized as:

$$V_P \frac{\partial \overline{\rho}_P}{\partial t} + \dot{m}_e - \dot{m}_w + \dot{m}_n - \dot{m}_s = 0 \tag{2.8}$$

Here, $\overline{\rho}_P = 1/V_P \int_{V_P} \rho dV$, and $\dot{m}_i = \int_{S_i} \rho \mathbf{vn} dS$.

By integrating the above equation over time (between time t^n and t^{n+1}), we derive:

$$V_P \int_{t^n}^{t^{n+1}} \frac{\partial \rho_P}{\partial t} dt + \int_{t^n}^{t^{n+1}} (\dot{m}_e - \dot{m}_w + \dot{m}_n - \dot{m}_s) dt = 0 \tag{2.9}$$

This results in the following implicit discretization form:

$$\frac{\rho_P - \rho_P^0}{\Delta t} V_P + \dot{m}_e - \dot{m}_w + \dot{m}_n - \dot{m}_s = 0 \tag{2.10}$$

In this equation, ρ_P and ρ_P^0 denote the densities at point P for time steps $n+1$ and n, respectively.

Returning to Eq. (2.6), we can express the numerical implicit approximations of the different integral terms as follows:

$$\int_{t^n}^{t^{n+1}} \int_{V_P} \frac{\partial(\rho\phi)}{\partial t} dV dt \approx V_P(\rho_P\phi_P - \rho_P^0\phi_P^0) \tag{2.11}$$

$$\int_{t^n}^{t^{n+1}} \int_{V_P} \nabla(\rho \mathbf{v} \phi) dV dt \approx (\dot{m}_e\phi_e - \dot{m}_w\phi_w + \dot{m}_n\phi_n - \dot{m}_s\phi_s)\Delta t \tag{2.12}$$

$$\int_{t^n}^{t^{n+1}} \int_{V_P} \nabla(\Gamma_\phi \nabla\phi) dV dt \approx \left(-\Gamma_w \frac{\phi_P - \phi_W}{d_{PW}} S_w + \Gamma_e \frac{\phi_E - \phi_P}{d_{PW}} S_e - \Gamma_s \frac{\phi_P - \phi_s}{d_{PS}} S_s + \Gamma_n \frac{\phi_N - \phi_P}{d_{PN}} S_n\right) \tag{2.13}$$

$$\int_{t^n}^{t^{n+1}} \int_{V_P} \dot{s_\phi} dV dt \approx (S_C^\phi + S_P^\phi \phi_P) V_P \Delta t \tag{2.14}$$

Utilizing the above terms and the discretized mass conservation equation, Eq. (2.10), we obtain the following equation:

$$\rho_P^0 \frac{\rho_P - \rho_P^0}{\Delta t} V_P + \dot{m}_e(\phi_e - \phi_P) - \dot{m}_w(\phi_w - \phi_P) + \dot{m}_n(\phi_n - \phi_P) - \dot{m}_s(\phi_s - \phi_P) = \tag{2.15}$$
$$D_e(\phi_E - \phi_P) - D_w(\phi_P - \phi_W) + D_n(\phi_N - \phi_P) - D_s(\phi_P - \phi_S) + (S_C^\phi + S_P^\phi \phi_P) V_P$$

Here, $D_e = \Gamma_e S_e / d_{PE}$ and $D_w = \Gamma_w S_w / d_{PW}$. It's worth noting that the source and diffusion terms are well-defined in the central nodes (P, E, W, S, and N) of the CVs, while the convective terms are attributed to the nodes on the faces (e, w, s, and n) of the CVs. At this stage, certain approximations need to be made to evaluate the convective terms in the central nodes.

Various methods can be used to evaluate the convective terms, and the choice depends on the specific requirements of the problem. Here are a few commonly used methods:

1. Central-Difference Scheme (CDS): The central-difference scheme (CDS) is a straightforward approach, which is given by the equation:

$$\phi_e - \phi_P = f_e(\phi_E - \phi_P) \tag{2.16}$$

   where $f_e = d_{Pe}/d_{PE}$. While CDS is a second-order scheme in terms of accuracy, it can lead to stability issues in certain cases.

2. Upwind-Difference Scheme (UDS): For incompressible flows or gases at low Mach numbers, the upwind-difference scheme (UDS) is often a good choice. This scheme considers that the convective terms are more influenced by upstream conditions than downstream conditions. UDS is more stable than CDS but sacrifices accuracy since it's a first-order scheme. The UDS approximation can be expressed as:

$$\dot{m}_e(\phi_e^{UDS} - \phi_P) = \frac{\dot{m}_e - |\dot{m}_e|}{2}(\phi_E - \phi_P) \tag{2.17}$$

3. Exponential-Difference Scheme (EDS): In the case of the exponential-difference scheme (EDS), the f_e function is defined as:

$$f_e = \frac{e^{Pe\, d_{Pe}/d_{PE}} - 1}{e^{Pe} - 1} \tag{2.18}$$

   Indeed, the Peclet number (*Pe*) plays a crucial role in evaluating the accuracy and stability of convective schemes. The exponential-difference scheme (EDS) is considered a first-order accurate scheme, and for higher accuracy, more sophisticated methods can be employed. Some of these include:

4. Second-Order Upwind Linear Extrapolation (SUDS): SUDS is a second-order scheme that provides improved accuracy over the first-order schemes like UDS. It achieves higher accuracy by employing linear extrapolation based on upwind information.

5. Quadratic Upwind Interpolation for Convective Kinematics (QUICK): QUICK is another second-order accurate scheme. It utilizes quadratic interpolation for the convective terms, which further enhances accuracy.

6. High-Resolution Schemes (HRS): High-resolution schemes are a class of methods that aim to improve the accuracy of convection terms. These schemes can offer higher-order accuracy and improved performance in capturing steep gradients and discontinuities in the solution.

   In the context of high-resolution schemes, the basic idea for the faces (f) can be expressed as:

$$\phi_f^{HRS} - \phi_P = (\phi^{UDS} f - \phi P) + (\phi_f^{HRS*} - \phi_f^{UDS*}) \tag{2.19}$$

   These higher-order schemes are particularly useful in scenarios where accuracy is of paramount importance, such as simulations involving sharp gradients or complex flow behaviors.

The choice of which scheme to use will depend on the specific demands of the problem and the balance between accuracy and computational efficiency.

In summary, when using the Upwind-Difference Scheme (UDS), the final form of the discretized generic convection-diffusion equation can be expressed as follows:

$$a_P \phi_P = a_e \phi_E + a_W \phi_W + a_N \phi_N + a_S \phi_S + b_P \tag{2.20}$$

Where:

$$a_E = D_e - \frac{\dot{m}e - |\dot{m}e|}{2} \;\; ; \;\; a_W = D_W + \frac{\dot{m}w + |\dot{m}w|}{2} \tag{2.21}$$

$$a_N = D_n - \frac{\dot{m}n - |\dot{m}n|}{2} \;\; ; \;\; a_S = D_s + \frac{\dot{m}s + |\dot{m}s|}{2} \tag{2.22}$$

$$a_P = a_E + a_W + a_N + a_S + \frac{\rho_P^0 V_P}{\Delta t} - S_P^{\phi} V_P \tag{2.23}$$

$$\begin{aligned} b_P = \frac{\rho_P^0 V_P}{\Delta t} \phi_P^0 + S_C^{\phi} V_P - \dot{m}e(\phi e^{HRS*} - \phi_e^{UDS*}) + \dot{m}w(\phi w^{HRS*} - \phi_w^{UDS*}) \\ -\dot{m}_n(\phi_n^{HRS*} - \phi_n^{UDS*}) + \dot{m}_s(\phi_s^{HRS*} - \phi_s^{UDS*}) \end{aligned} \tag{2.24}$$

where, for instance, $\dot{m}_e$ can be approximated as $\rho(V_E - V_P)\Delta x \Delta z/2$.

It's important to note that in this work, no source term ($S_C = 0$) and no High-Resolution Scheme (HRS) correction are considered. This means that in the resolution of the equations mentioned earlier, fewer terms need to be taken into account.

Additionally, Boundary Conditions (BC) play a crucial role in numerical calculations. A more comprehensive description of the Boundary Conditions will be provided in Section **3**

3 Problem Description

3.1 Simth - Hutton

For the Smith-Hutton scenario, we use a non-symmetrical box with dimensions of $2L \times L$. In this case, boundary conditions are imposed as follows:

- **Inlet**

  $-1 \leq x \leq 0$ and $y = 0$

  $\phi = 1 + \tanh[10(2x+1)]$

  $a_E = a_W = a_N = a_S = 0$

  $a_P = 1$

  $b_P = 1 + \tanh[10(2x+1)]$

- **Outlet**

  $0 < x < 1$ and $y = 0$

  $\frac{\partial \phi}{\partial y} = 0$

  $a_E = a_W = a_S = 0$

  $a_N = a_P = 1$

  $b_P = 0$

- **Left Wall**

  $x = -1$

  $\phi = 1 - \tanh(10)$

  $a_E = a_W = a_N = a_S = 0$

  $a_P = 1$

  $b_P = 1 - \tanh(10)$

- **Top Wall**

  $-1 \leq x \leq 1$ and $y = 1$

  $\phi = 1 - \tanh(10)$

  $a_E = a_W = a_N = a_S = 0$

  $a_P = 1$

  $b_P = 1 - \tanh(10)$

- **Right Wall**

  $x = 1$

  $\phi = 1 - \tanh(10)$

  $a_E = a_W = a_N = a_S = 0$

  $a_P = 1$

  $b_P = 1 - \tanh(10)$

Finally, the velocity field for the Smith-Hutton problem is defined by the following equations:

$$v_x = 2y(1 - x^2) \tag{3.1}$$

$$v_y = -2x(1 - y^2) \tag{3.2}$$

These equations describe the velocity components, with v_x representing the horizontal velocity component and v_y representing the vertical velocity component.

3.2 Diagonal Flow

For the Diagonal Flow scenario, which features a symmetrical box with dimensions of $L \times L$, the boundary conditions are imposed as follows:

- **Bottom and Right Walls**

  $\phi = \phi_{low}$ at $(x, y = 0)$ and $(x = L, y)$

  $a_E = a_W = a_N = a_S = 0$

  $a_P = 1$

  $b_P = \phi_{low}$

- **Top and Left Walls**

  $\phi = \phi_{high}$ at $(x = 0, y)$ and $(x, y = H)$

  $a_E = a_W = a_N = a_S = 0$

  $a_P = 1$

  $b_P = \phi_{high}$

The velocity field is described by the following equations:

$$v_x = V_{In} \cos(\alpha) \tag{3.3}$$

$$v_y = V_{In} \sin(\alpha) \tag{3.4}$$

In these equations, V_{In} is a constant value, and $\alpha = \pi/4$. These equations define the velocity components for the Diagonal Flow scenario.

4 Code Structure

The code structure for solving the convection-diffusion equation is outlined as follows:

1. **Input Data**

   The input data is segregated into two sections: one for physical aspects of the problem, including parameters like the cavity's length and height (L and H), 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 section is dedicated to numerical aspects, such as the number of Control Volumes (CVs) (N and M).

2. **Mesh**

   Generate the mesh with $N \times M$ CVs over L and H.

3. **Matrixes**

   Define all the matrixes with dimension of $(N+2) \times (M+2)$.

4. **Velocity Fields**

   Compute the velocity fields, as explained in Section **3**.

5. **Initialization**

   Initialize ϕ and ϕ^0. Evaluate all the internal nodes' coefficients a_i, where $i \in [P, E, W, N, S]$ and b_P using the chosen evaluation scheme. Compute the boundary conditions (BCs).

6. **Time Step**

   Evaluation of the new time steps: $t = t + \Delta t$.

7. **Gauss-Seidel**

   Implement the Gauss-Seidel algorithm to solve for all internal nodes.

8. **Check Convergence**

   Ask for a new time step if needed (Is $|\phi - \phi^0| < \varepsilon$? If the answer is NO, a new time step is needed).

9. **Final Calculations**

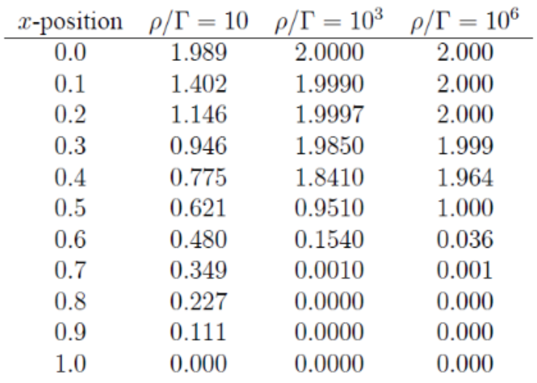
   Final calculations and create all the plots and save them.

5 Code Verification

Before presenting the results obtained in this work, it is essential to perform an analysis to ensure the proper functionality of the code. This analysis includes, confirming that it converges to a solution, validating the results against known solutions or benchmark cases and a mesh refinement to obtain the suitable size.

5.1 Comparison

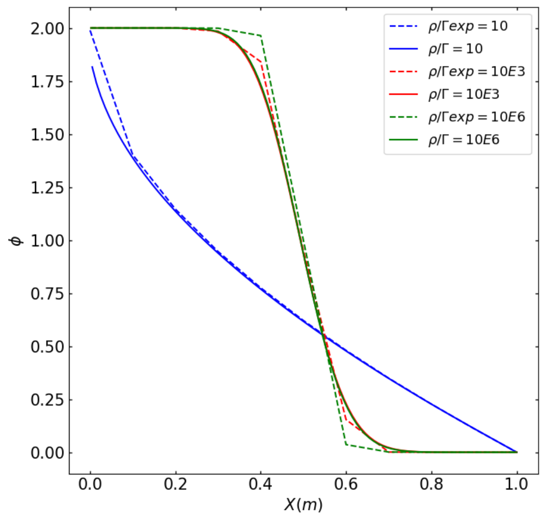
In this first section we will compare our results with some reference solutions, which we can observe in the following table:



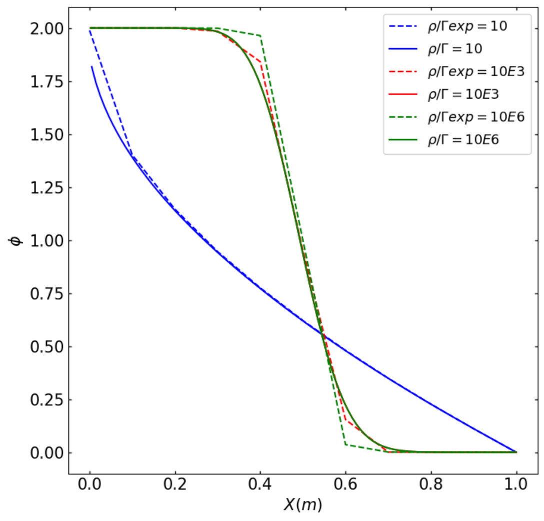
x-position	$\rho/\Gamma = 10$	$\rho/\Gamma = 10^3$	$\rho/\Gamma = 10^6$
0.0	1.989	2.0000	2.000
0.1	1.402	1.9990	2.000
0.2	1.146	1.9997	2.000
0.3	0.946	1.9850	1.999
0.4	0.775	1.8410	1.964
0.5	0.621	0.9510	1.000
0.6	0.480	0.1540	0.036
0.7	0.349	0.0010	0.001
0.8	0.227	0.0000	0.000
0.9	0.111	0.0000	0.000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

Figure 5.1: Values of ϕ for the Smith-Hutton problem with different ρ/Γ.

For this comparison, we will use the two differents schemes UDS and EDS and we will see the error obtained respect to the reference data.



(a) Values of ϕ for different ρ/Γ with UDS scheme and $N = 200$ & $M = 100$.



(b) Values of ϕ for different ρ/Γ with EDS scheme and $N = 200$ & $M = 100$.

Figure 5.2: Values of ϕ for the Smith-Hutton problem with different ρ/Γ.

In Figure 5.2, the different values of ϕ at the outlet for $\rho/\Gamma = 10, 10^3$, and 10^6 using the Upwind-

Difference Scheme (UDS) and Exponential-Difference Scheme (EDS) are presented. It is observed that the main discrepancies occur at the upper and lower values of $\rho/\Gamma = 10^3$ and 10^6 and at the initial values for $\rho/\Gamma = 10$, being practically identical for both schemes.

If we compute the errors for both schemes we obtain the following results:

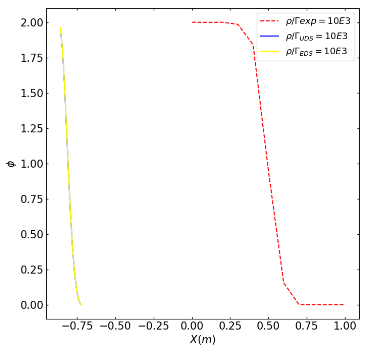
ρ/Γ	**UDS**$_{err}$ (%)	**EDS**$_{err}$(%)
10	6.83	6.59
10^3	1.47	1.41
10^6	2.09	2.03

Table 5.1: Relative errors of ϕ respect to the reference for UDS and EDS schemes.

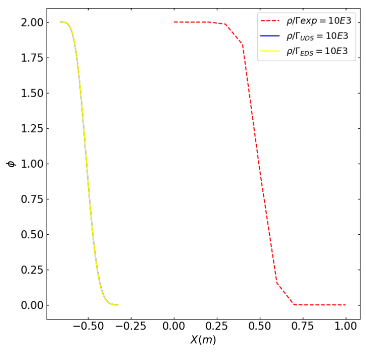
We can see how lower errors are obtained for the EDS scheme than for the UDS, although the difference is minimal, which is why the graphs appear to be the same. Therefore, by obtaining such low errors we can consider at first that our code is working correctly.

5.2 Mesh Refinement

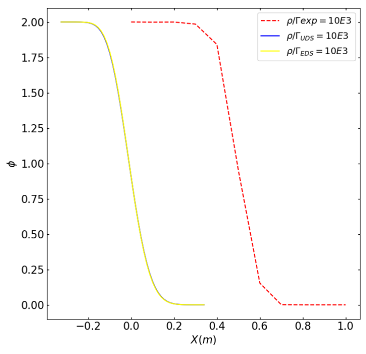
In this section we will compare the results obtained for different mesh sizes, the objective of which will be to choose an appropriate size for the accuracy - computational cost ratio.



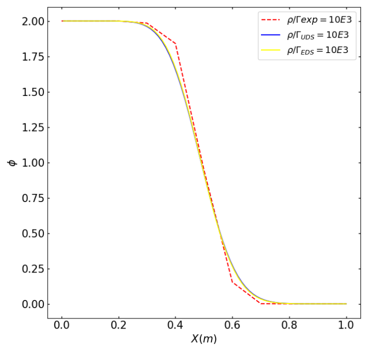
(a) Mesh 20x10



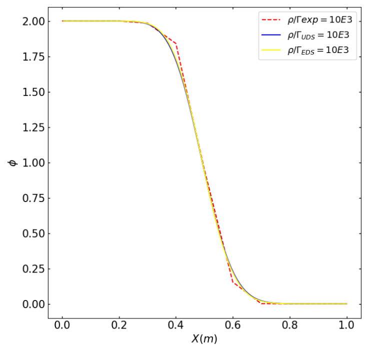
(b) Mesh 50x25



(c) Mesh 100x50



(d) Mesh 150x75



(e) Mesh 200x100

Figure 5.3: Values of ϕ for different meshes with $\rho/\Gamma = 10E3$.

As can be seen, the value of ϕ in the outlet gets closer and closer to the reference values as we increase the mesh. If we compute the evolution of the error

Mesh (NxM)	**UDS**$_{err}$ (%)	**EDS**$_{err}$(%)
$20x10$	18.19	18.12
$50x25$	7.37	7.30
$100x50$	3.48	3.42
$150x75$	2.15	2.08
$200x100$	1.47	1.41

Table 5.2: Relative errors of ϕ respect to the reference for UDS and EDS schemes and different meshes.

As expected, the errors become smaller the larger the mesh, in accordance with the observations in 5.3. Therefore, from now on the results we will show will be for a mesh of 200x100 and EDS scheme because it is with that the most accurate results are obtained.

6 Results

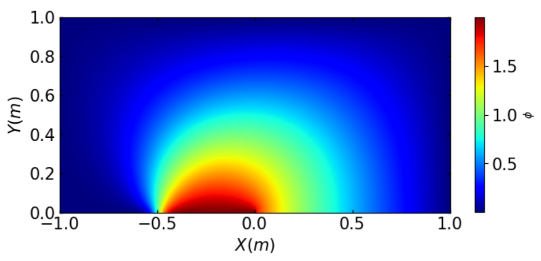
6.1 Smith - Hutton

In this section we will analyze the results obtained for the Smith-Hutton case. But before we start, let's see what input data we have used in our solver:

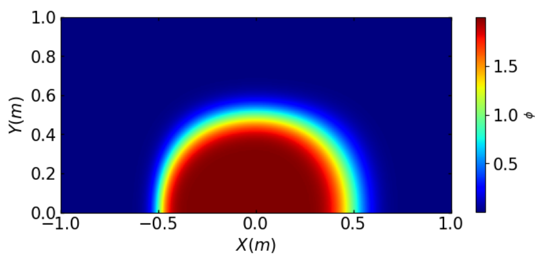
Name	**Symbol**	**Value**	**Units**
X Direction Mesh	N	200	#
Y Direction Mesh	M	100	#
Accuracy	δ	10^{-6}	#
Time Step	Δt	0.1	s
Length	L	2	m
Height	H	1	m
Input Temperature	T_{in}	293	K
Input Pressure	P_{in}	1.013×10^5	N / m^2
Input Density	ρ_{in}	1.205	kg / m^3

Table 6.1: Input data

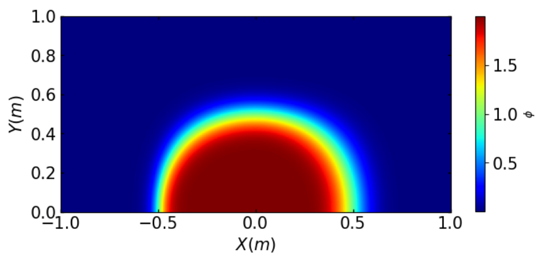
Once we know the input data, we proceed to plot the different results obtained by following the steps mentioned in section 4.



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



(b) $\rho/\Gamma = 10E3$



(c) $\rho/\Gamma = 10E6$

Figure 6.1: Values of ϕ for Smith - Hutton problem with a mesh of 200x100 and EDS scheme.

The primary findings are showcased in Figure 6.1. This graph illustrates how ϕ changes for three distinct ρ/Γ values. As anticipated, we observe variations in ϕ from the inlet to the outlet. Notably, for the smallest ρ/Γ value, the results exhibit less symmetry, indicating a notable presence of diffusion. This suggests that the convective term is more pronounced in cases with lower Peclet values.

As we move to higher ρ/Γ values, the solution becomes more symmetric, though some diffusion is still noticeable. This signifies that the diffusive effect decreases with increasing ρ/Γ values.

These findings provide valuable insights into how the convection-diffusion equation behaves for different ρ/Γ values. They highlight how the solution's symmetry and the dominance of convection or diffusion effects change with varying parameter values.

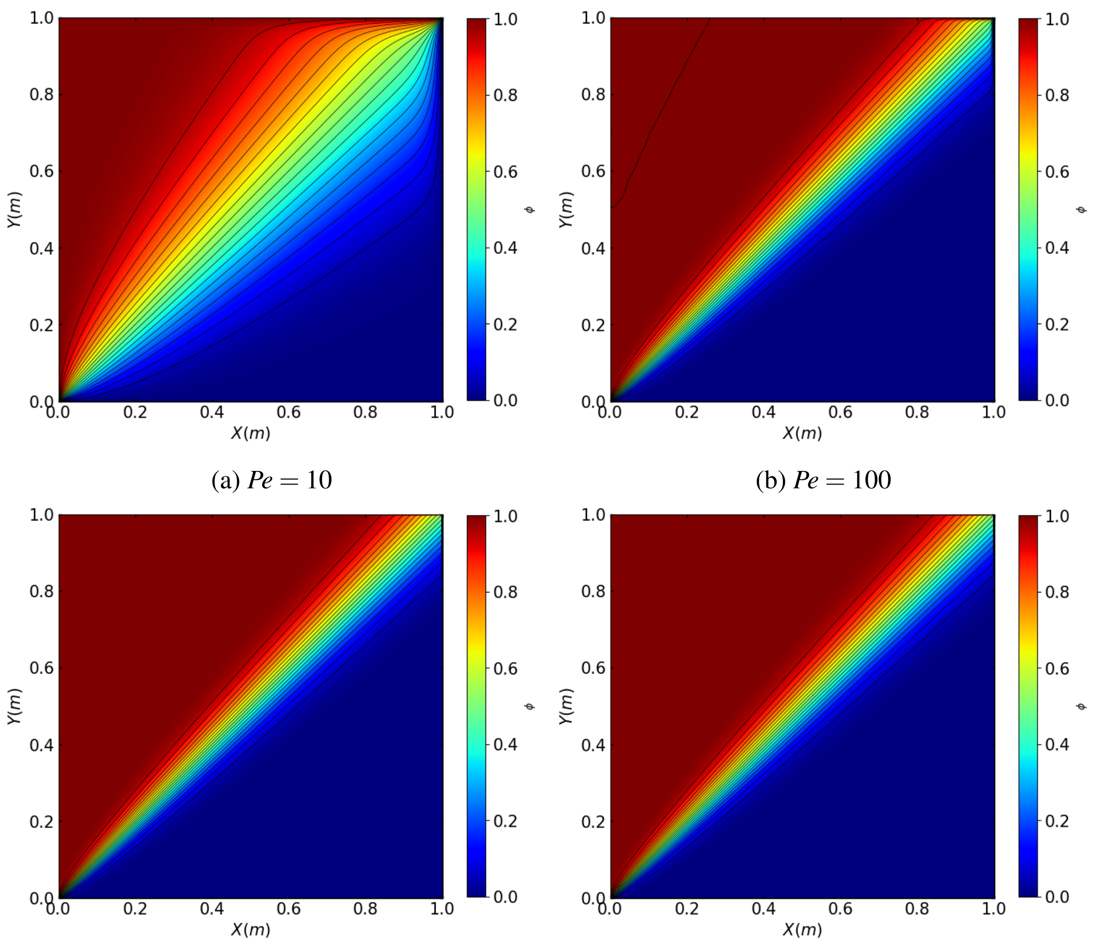
6.2 Diagonal Flow

In this section we are going to analyze the results of the Diagonal Flow problem. In Table 6.2 we show the physical and numerical values used to present the results.

Name	Symbol	Value	Units
X Direction Mesh	N	200	#
Y Direction Mesh	M	200	#
Accuracy	δ	10^{-6}	#
Time Step	Δt	0.1	s
Length	L	1	m
Height	H	1	m
Input Temperature	T_{in}	293	K
Input Pressure	P_{in}	1.013×10^5	N / m^2
Input Velocity	V_{in}	5.15	m / s
Input Density	ρ_{in}	1.205	kg / m^3
Peclet number	Pe	$[1, 100, 10^3, 10^7]$	#
ϕ top and left walls	ϕ_{high}	1.0	#
ϕ bottom and right walls	ϕ_{low}	0.0	#

Table 6.2: Used numerical and physical input data for simulations.

Once we know the input data that has been used for carry out the solver, we plot the results, obtaining the following:



(a) $Pe = 10$

(b) $Pe = 100$

(c) $Pe = 10^3$.

(d) $Pe = 10^7$

Figure 6.2: Values of ϕ for the Diagonal Flow problem for different Pe values.

In this figure we have represented the lines for a better observation of the results. We depict the variations in ϕ for different Peclet numbers. Two distinct regions are evident: 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 the Peclet number increases, the extent of diffusion diminishes. This trend is attributed to the direct relationship between the Peclet number and the diffusion term.

In an ideal scenario, free from numerical errors, as *Pe* approaches infinity, we would anticipate the elimination of diffusion in our solution. This would result in the upper-mid region being saturated with ϕ_{high} values, while the lower-mid region would predominantly contain ϕ_{low} values. Consequently, our solution aligns with this expected behavior.

Furthermore, our findings indicate that for large Peclet values, there is a point beyond which further reduction of solution diffusion is not achievable, or any reduction is extremely minimal. This limitation is primarily attributed to the presence of numerical errors, which introduce false diffusion.

7 Conclusions

This study investigated the behavior of the generic convection-diffusion equation through two scenarios: the Smith-Hutton and Diagonal Flow problems. The emphasis was on comparing the results with reference solutions and examining the impact of mesh refinement.

In the comparison with the reference solution for the Smith-Hutton problem, several key observations were made. For ρ/Γ values of 10, 10^3 and 10^6 both the Upwind-Difference Scheme (UDS) and Exponential-Difference Scheme (EDS) exhibited relatively small errors compared to the reference, with EDS consistently outperforming UDS, demonstrating the code's ability to reproduce reference values.

The mesh refinement studies confirmed that simulations with larger meshes led to smaller errors compared to the reference, which aligns with expectations. Larger meshes resulted in reduced relative errors. Furthermore, EDS consistently outperformed UDS in terms of accuracy. Additionally, the computational time for both schemes was considered. Larger meshes required more computational time, while larger ρ/Γ values reduced the computation time. UDS generally exhibited faster computation, which could be advantageous for larger simulations.

In the results section, the focus was on the Exponential-Difference Scheme (EDS) due to its higher precision. In the Smith-Hutton problem, different ρ/Γ values were explored. It was evident that for smaller ρ/Γ values, the solution displayed less symmetry, indicating a significant diffusion effect. As ρ/Γ increased, the solution became more symmetrical, though some degree of diffusion persisted. Notably, numerical errors introduced false diffusion for the largest ρ/Γ values.

For the Diagonal Flow problem, simulations covered a range of Peclet numbers. The findings revealed that as the Peclet number increased, the influence of diffusion decreased. However, for very high Peclet numbers, numerical errors introduced a false diffusion component.

8 Code

In this section we show the code for the Smith - Hutton problem, the code for the Diagonal Flow problem will be send in a .zip by ATENEA.

```
"""Smith-Hutton.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1SHjp0SAUuFIY5SV6Api_M0gPYlplpfYj
"""

import numpy as np
import matplotlib.pyplot as plt
import random
from google.colab import drive
drive.mount('/content/drive')
import os

"""1 - INPUT DATA"""

# Physical Data
L = 2.0  #m
H = 1.0  #m
P_in = 101325  #Pa
T_in = 293  #K
Rho_in = P_in / (287*T_in)  #kg/m^3

# Numeral Data
N = 200
M = 100
rho_gamma = 10E3  # there will be 3 different cases: 10, 1000 and 10E6
gamma = Rho_in/rho_gamma
dt = 0.1  # time step
delta = 0.000001  # accuracy
delta_N = L/N
delta_M = H/M
t_max = 1000000 # G-S loop steps.
time_max = 1000000 # Time loop steps.

"""MESH"""

# Control Volumes
X_cv = np.linspace(0, L, N+1)
X_cv_2 = np.linspace(-L/2, L/2, N+1)
Y_cv = np.linspace(0, H, M+1)

#print(X_cv)
# Mesh Generation
X_P  = np.zeros(N+2)
X_P_2  = np.zeros(N+2)
Y_P  = np.zeros(M+2)

for i in range(1, N+1):
    X_P[i] = (X_cv[i-1] + X_cv[i])/2
    X_P_2[i] = (X_cv_2[i-1] + X_cv_2[i])/2

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

X_P[0] = X_cv[0]
X_P[-1] = X_cv[-1]
```

```
X_P_2[0] = X_cv_2[0]
X_P_2[-1] = X_cv_2[-1]
Y_P[0] = Y_cv[0]
Y_P[-1] = Y_cv[-1]

#print(X_P)
# Velocity Field
Vx = np.zeros((M+2, N+2))
Vy = np.zeros((M+2, N+2))
for i in range(N+2):
  for j in range(M+2):
    Vx[j][i] = 2*Y_P[j]*(1-(X_P_2[i])**2)
    Vy[j][i] = -2*X_P_2[i]*(1-(Y_P[j])**2)
#print(Vx)
#print(Vy)
# Define Metrics
a_E = np.zeros((M+2,N+2))
a_W = np.zeros((M+2,N+2))
a_S = np.zeros((M+2,N+2))
a_N = np.zeros((M+2,N+2))
b_P = np.zeros((M+2,N+2))
a_P = np.zeros((M+2,N+2))
m_e = 0.0
m_s = 0.0
m_w = 0.0
m_n = 0.0
D_e = 0.0
D_w = 0.0
D_n = 0.0
D_s = 0.0
phi = np.zeros((M+2,N+2))
phi0 = np.zeros((M+2,N+2))
phi_aux = np.zeros((M+2,N+2))

"""INITIAL MAP"""

# Internal nodes
for i in range(1, N+1):
    for j in range(1, M+1):
        phi0[j][i] = 1.0
        phi[j][i] = 1.0

# Inlet nodes
for i in range(int((N+2) / 2)):
    for j in range(M+2):
        phi0[0][i] = 1.0 + np.tanh(10.0 * (2.0 * X_P_2[i] + 1.0))
        phi[0][i] = 1.0 + np.tanh(10.0 * (2.0 * X_P_2[i] + 1.0))

# Walls
for i in range(N+2):
    for j in range(M+2):
        phi0[j][0]  = 1.0 - np.tanh(10.0)
        phi0[-1][i] = 1.0 - np.tanh(10.0)
        phi0[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)

"""PREVIOUS CALCULATIONS"""

def scheme(type, Pe):
  if type == 0:
    return 1
  if type == 1:
    return np.abs(Pe)/(np.exp(np.abs(Pe))-1)

# Internal nodes
type = 0 ################################# 0 --> upwind & 1 --> Exponential

for i in range(1, N+1):
  for j in range(1, M+1):
      De = (gamma*delta_M)/np.abs(X_P_2[i] - X_P_2[i + 1])
      m_e = Rho_in * (Vx[j][i + 1] + Vx[j][i]) * delta_M / 2.0
      Pe = m_e/De
      a_E[j][i]=De * scheme(type,Pe) - ((m_e - np.abs(m_e)) / 2.0)
      #a_E[j][i]=De * scheme(type,Pe) + np.max((-1)*m_e, 0)

      Dw = (gamma*delta_M)/np.abs(X_P_2[i] - X_P_2[i - 1])
      m_w = Rho_in * (Vx[j][i - 1] + Vx[j][i]) * delta_M / 2.0
      Pw = m_w/Dw
      #a_W[j][i]=Dw * scheme(type,Pw) + np.max((1)*m_w, 0)
      a_W[j][i]=Dw * scheme(type,Pw) + ((m_w + np.abs(m_w)) / 2.0)

      Dn = (gamma*delta_N)/np.abs(Y_P[j] - Y_P[j + 1])
      m_n = Rho_in * (Vy[j + 1][i] + Vy[j][i]) * delta_N / 2.0
      Pn = m_n/Dn
      #a_N[j][i]=Dn * scheme(type,Pn) + np.max((-1)*m_n, 0)
      a_N[j][i]=Dn * scheme(type,Pn) - ((m_n - np.abs(m_n)) / 2.0)

      Ds = (gamma*delta_N)/np.abs(Y_P[j] - Y_P[j - 1])
      m_s = Rho_in * (Vy[j - 1][i] + Vy[j][i]) * delta_N / 2.0
      Ps = m_s/Ds
      #a_S[j][i]=Ds * scheme(type,Ps) + np.max((1)*m_s, 0)
      a_S[j][i]=Ds * scheme(type,Ps) + ((m_s + np.abs(m_s)) / 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_N * delta_M)/dt

"""EVALUATION DISCRETIZATION COEFFICIENTS"""

# 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_2[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):
        # Left
        a_E[j][0]  = 0.0
        a_W[j][0]  = 0.0
        a_N[j][0]  = 0.0
        a_S[j][0]  = 0.0
        b_P[j][0]  = 1.0 - np.tanh(10.0)
        a_P[j][0]  = 1.0
        # Right
        a_E[j][-1] = 0
        a_W[j][-1] = 0
        a_N[j][-1] = 0
        a_S[j][-1] = 0
        a_P[j][-1] = 1
        b_P[j][-1] = 1.0 - np.tanh(10.0)
        # Top
        a_E[-1][i] = 0.0
        a_W[-1][i] = 0.0
        a_N[-1][i] = 0.0
        a_S[-1][i] = 0.0
        a_P[-1][i] = 1.0
        b_P[-1][i] = 1.0 - np.tanh(10.0)

"""GAUSS-SEIDEL METHOD"""

for time in range (time_max):
  r2 = np.sum(phi0)
  phi_aux = phi0
  for t in range(t_max):
    for i in range(int((N+2) / 2), N+2):
                phi_aux[0,i] = phi_aux[1,i]
    # Internal Nodes
    for i in range (1, N+1):
        for j in range (1, M+1):
          b_P[j][i] = ((Rho_in * delta_N * delta_M)/dt) * phi_aux[j][i]
          phi[j][i] = (a_E[j][i] * phi_aux[j][i+1] + a_W[j][i]*phi_aux[j][i-1] //
          + a_N[j][i] * phi_aux[j+1][i] + a_S[j][i] * phi_aux[j-1][i] //
          + b_P[j][i])/a_P[j][i]

    r = np.sum(phi_aux)
    sum = np.sum(phi)

    if np.abs(sum-r) <= delta:
      break

    else:
      phi_aux = phi
  sum2 = np.sum(phi)
  if np.abs(sum2 - r2) <= delta:
    print("Time␣loop␣has␣converged␣successfully")
    break
  else:
    phi0 = phi

# Ruta a la carpeta en la que deseas guardar el archivo binario
ruta_carpeta = ’/content/drive/My␣Drive/Compu/Assignment␣2’

# Asegurate de que la carpeta exista
os.makedirs(ruta_carpeta, exist_ok=True)

# Nombre del archivo binario
nombre_archivo = f’phi_10E3_UDS_{N}x{M}.bin’

# Ruta completa al archivo
ruta_archivo = os.path.join(ruta_carpeta, nombre_archivo)

# Guarda el array en el archivo binario
with open(ruta_archivo, ’wb’) as archivo:
    np.save(archivo, phi)
```

```
print(f"El␣archivo␣se␣ha␣guardado␣en:␣{ruta_archivo}")

"""VECTORS"""

x_exp = [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
phi_ten     = [1.989, 1.402, 1.146, 0.946, 0.775, 0.621, 0.480, 0.349, 0.227, 0.111, 0.000]
phi_mil   = [2.0000, 1.9990, 1.9997, 1.9850, 1.8410, 0.9510, 0.1540, 0.0010, 0.0000, 0.0000, 0.0000]
phi_million   = [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))

# 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_2, Y_P, phi, levels=20, colors='black', linewidths=0.5)
plt.imshow(phi, cmap= 'jet', extent=(X_P_2.min(), X_P_2.max(), Y_P.min(), Y_P.max()), origin='lower')
plt.colorbar( label = r"$\phi$", shrink = 0.4 ).ax.tick_params( labelsize =15)

plt.show(1)

# Vectors for outlet plot

x_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_exp     = [1.989, 1.402, 1.146,//
0.946, 0.775, 0.621, 0.480, 0.349, 0.227, 0.111, 0.000]
phi_10E3_exp   = [2.0000, 1.9990, 1.9997, //
1.9850, 1.8410, 0.9510, 0.1540, 0.0010, 0.0000, 0.0000, 0.0000]
phi_10E6_exp    = [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_10_plot_UDS = np.zeros(int((N+2) / 2))
phi_10E3_plot_UDS = np.zeros(int((N+2) / 2))
phi_10E6_plot_UDS = np.zeros(int((N+2) / 2))
phi_10_plot_EDS = np.zeros(int((N+2) / 2))
phi_10E3_plot_EDS = np.zeros(int((N+2) / 2))
phi_10E6_plot_EDS = np.zeros(int((N+2) / 2))

#ruta_archivo_10_UDS = '/content/drive/My Drive/Compu/Assignment 2/phi_10_UDS.bin'
ruta_archivo_10E3_UDS = '/content/drive/My␣Drive/Compu/Assignment␣2/phi_10E3_UDS_200x100.bin'
#ruta_archivo_10E6_UDS = '/content/drive/My Drive/Compu/Assignment 2/phi_10E6_UDS.bin'
#ruta_archivo_10_EDS = '/content/drive/My Drive/Compu/Assignment 2/phi_10_EDS.bin'
ruta_archivo_10E3_EDS = '/content/drive/My␣Drive/Compu/Assignment␣2/phi_10E3_EDS_200x100.bin'
#ruta_archivo_10E6_EDS = '/content/drive/My Drive/Compu/Assignment 2/phi_10E6_EDS.bin'

#with open(ruta_archivo_10_UDS, 'rb') as file:
#  phi_10_UDS = np.load(file)
with open(ruta_archivo_10E3_UDS, 'rb') as file:
  phi_10E3_UDS = np.load(file)
#with open(ruta_archivo_10E6_UDS, 'rb') as file:
#  phi_10E6_UDS = np.load(file)
#with open(ruta_archivo_10_EDS, 'rb') as file:
```

```
#  phi_10_EDS = np.load(file)
with open(ruta_archivo_10E3_EDS, 'rb') as file:
  phi_10E3_EDS = np.load(file)
#with open(ruta_archivo_10E6_EDS, 'rb') as file:
#  phi_10E6_EDS = np.load(file)

for i in range(int((N+2) / 2), N+2):
  x_plot[i - int((N+2) / 2)] = X_P_2[i]
  #phi_10_plot_UDS[i - int((N+2) / 2)] = phi_10_UDS[0][i]
  phi_10E3_plot_UDS[i - int((N+2) / 2)] = phi_10E3_UDS[0][i]
  #phi_10E6_plot_UDS[i - int((N+2) / 2)] = phi_10E6_UDS[0][i]
  #phi_10_plot_EDS[i - int((N+2) / 2)] = phi_10_EDS[0][i]
  phi_10E3_plot_EDS[i - int((N+2) / 2)] = phi_10E3_EDS[0][i]
  #phi_10E6_plot_EDS[i - int((N+2) / 2)] = phi_10E6_EDS[0][i]

# Compute the errors
#err_ref_10 = np.sum(phi_10_exp)/len(phi_10_exp)
err_ref_10E3 = np.sum(phi_10E3_exp)/len(phi_10E3_exp)
#err_ref_10E6 = np.sum(phi_10E6_exp)/len(phi_10E6_exp)

#err_10_UDS = np.sum(phi_10_plot_UDS)/len(phi_10_plot_UDS)
err_10E3_UDS = np.sum(phi_10E3_plot_UDS)/len(phi_10E3_plot_UDS)
#err_10E6_UDS = np.sum(phi_10E6_plot_UDS)/len(phi_10E6_plot_UDS)

#err_10_EDS = np.sum(phi_10_plot_EDS)/len(phi_10_plot_EDS)
err_10E3_EDS = np.sum(phi_10E3_plot_EDS)/len(phi_10E3_plot_EDS)
#err_10E6_EDS = np.sum(phi_10E6_plot_EDS)/len(phi_10E6_plot_EDS)

#error_10_UDS = (np.abs(err_10_UDS-err_ref_10)/err_ref_10)*100
error_10E3_UDS = (np.abs(err_10E3_UDS-err_ref_10E3)/err_ref_10E3)*100
#error_10E6_UDS = (np.abs(err_10E6_UDS-err_ref_10E6)/err_ref_10E6)*100

#error_10_EDS = (np.abs(err_10_EDS-err_ref_10)/err_ref_10)*100
error_10E3_EDS = (np.abs(err_10E3_EDS-err_ref_10E3)/err_ref_10E3)*100
#error_10E6_EDS = (np.abs(err_10E6_EDS-err_ref_10E6)/err_ref_10E6)*100

#print(f'Error 10 UDS:', error_10_UDS)
print(f'Error␣10E3␣UDS:', error_10E3_UDS)
#print(f'Error 10E6 UDS:', error_10E6_UDS)
#print(f'Error 10 EDS:', error_10_EDS)
print(f'Error␣10E3␣EDS:', error_10E3_EDS)
#print(f'Error 10E6 EDS:', error_10E6_EDS)
## Figure specifications
fontsize=15
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"$\phi$", fontsize=fontsize)

# Plot
#plt.plot(x_exp, phi_10_exp, color="blue",linestyle = "dashed", label=r"$\rho/\Gamma exp =10$")
#plt.plot(x_plot, phi_10_plot, color="blue",linestyle = "solid", label=r"$\rho/\Gamma=10$")
plt.plot(x_exp, phi_10E3_exp, color="red",linestyle = "dashed", label=r"$\rho/\Gamma␣exp␣=10E3$")
plt.plot(x_plot, phi_10E3_plot_UDS, color="blue",linestyle = "solid", label=r"$\rho/\Gamma_{UDS}=10E3$"
plt.plot(x_plot, phi_10E3_plot_EDS, color="yellow",linestyle = "solid", label=r"$\rho/\Gamma_{EDS}=10E3
#plt.plot(x_exp, phi_10E6_exp, color="green",linestyle = "dashed", label=r"$\rho/\Gamma exp =10E6$")
#plt.plot(x_plot, phi_10E6_plot, color="green",linestyle = "solid", label=r"$\rho/\Gamma=10E6$")
#
# Legend specifications
plt.legend(fontsize=fontsize-2, loc='upper␣right')

plt.show()
```