Smith-Huton Problem
Matlab OOP

Student name: Altadill Llasat, Miquel

Teacher name: Rigola Serrano, Joaquim

Department: Centre Tecnològic de Transferència de Calor (CTTC)

Document: Code Report

Delivery Date: 13/06/2019

Aerospace Vehicles Engineering - ESEIAAT
Universitat Politècnica de Catalunya (UPC)

Contents

Acronyms

1D One-Dimensional. 13, 17

2D Two-Dimensional. 10, 13, 17, 18

3D Three-Dimensional. 17

CDS Central Difference Scheme. 14, 15

CV Control Volume. 9, 12–14

EDS Exponential Difference Scheme. 14, 17, 18

FVM Finite Volume Method. 12

HDS Hybrid Difference Scheme. 14, 15, 17

N-S Navier-Stokes. 5, 8, 10, 11, 27

PLDS Powerlaw Difference Scheme. 14, 17

UDS Upwind Difference Scheme. 14, 15

List of Figures

List of Tables

Chapter 1

Approach to the physical phenomenon and mathematical formulation

Contents

Many problems that involves the resolution of differential equations can be solved *analytically* specially those ones that involve simple geometries with simple boundary conditions. But when the problem involve complicated geometries with complex *boundary conditions* and variable properties its needed another method for solving the equations involved in the physical phenomenon. For this cases we can still obtain sufficiently accurate approximate solutions using *numerical methods*, those are based on replacing the differential equation by a set of n algebraic equations for the unknown medium property at n selected points of the medium, and the simultaneous solution of these equation results in the medium property values at those *discrete points*. We are going to call this arbitrary medium property or dependent variable ϕ to refer to it in the following sections. [3]

The numerical solution of heat transfer, fluid flow, and other related processes can begin when the laws governing these processes have been expressed in mathematical form, generally in terms of differential equations. In this section we are going to develop the mathematical formulation and complete derivation of these equations as an initial step for developing the code for modelling these phenomenon [1]. The purpose in this section is to develop the familiarity with the form and meaning of these equations, geometric formulation of the control volume and the main ingredients for developing the *numerical simulation* tools for the case of study.

1.1 Convection and Diffusion

Accurate modelling of the interaction between convective and diffusive processes is a challenging task in numerical approximation of partial differential equations. Many different ideas and approaches have been proposed in different contexts in order to resolve the difficulties such as exponential fitting, compact differences, upwind, etc. being some examples from the fields of finite difference and finite element methods.

It is important to know that mathematical models that involve a combination of convective and diffusive processes are among the most widespread in all of science, engineering and other fields where mathematical models are involved. Although convection is the only new term introduced in this section, its formulation is not very simple. The convection term has an inseparable connection with the diffusion term so they need to be handled as one unit. This section gives us a better understanding of the Navier-Stokes equations before treating the final equation that merges the convection and diffusion phenomena. [4][1]

1.1.1 Navier-Stokes Equations

Before starting the formulation of the Convection-Diffusion equation its important to have clear the meaning of each one of the N-S equations. The N-S equations consists of the continuity equation, which represents the mass conservation principle (Eq.1.1); the momentum conservation equations, one for each problem dimension (Eq.1.2); and the energy conservation equation (Eq.1.3). [5]

$$\frac{d\rho}{dt} + \nabla \cdot (\rho \overrightarrow{v}) = 0 \tag{1.1}$$

$$\frac{d}{dt}(\rho \overrightarrow{v}) + \nabla \cdot (\rho \overrightarrow{v}\,\overrightarrow{v}) = -\nabla p + \nabla \cdot (\overrightarrow{\tau}) + \rho \overrightarrow{g} \tag{1.2}$$

$$\frac{d}{dt}(\rho(u + e_c)) + \nabla \cdot ((u + e_c)\rho \overrightarrow{v}) = -\nabla \cdot (\rho \overrightarrow{v}) + \nabla \mathring{u}(\overrightarrow{v} \cdot \overrightarrow{\tau}) - \nabla \cdot \overrightarrow{q} + \rho \overrightarrow{g} \cdot \overrightarrow{v} + G \tag{1.3}$$

Some previous assumptions have been done in the N-S equations, this hypothesis are:

- Continuity of matter
- Continuum medium assumption
- Relativity effects negligible
- Inertial reference system
- Magnetic and electromagnetic forces negligible

Continuity conservation equation

This equation defines that the variation of mass in the control volume has to be equal to the mass flow through its faces. Therefore, taking reference to the terms of Eq. 1.1:

- The first term $\frac{d\rho}{dt}$ represents the variation of mass inside the control volume in a differential time.
- The second term $\nabla \cdot (\rho \overrightarrow{v})$represents the mass flow through the faces of the control volume.

Momentum conservation equation

This equation shows us that the variation of linear momentum in the control volume plus the momentum flux through the CV faces has to be equal to the sum of the forces that act on the CV. Therefore, taking reference to the terms of Eq. 1.2:

- The first term $\frac{d}{dt}(\rho \overrightarrow{v})$ represents the variation of linear momentum in the control volume.
- The second term $\nabla \cdot (\rho \overrightarrow{v} \overrightarrow{v})$ represents the momentum flux through the faces of its control volume.
- The third term ∇p is the pressure gradient acting like an axial force on the faces of the CV.
- The fourth term $\nabla \cdot (\overrightarrow{\tau})$ is the total stress tensor. This force acts axially and tangentially on the faces of the control volume. Its value depends on the type of fluid (Newtonian, non-Newtonian...).
- The fifth term $\rho \overrightarrow{g}$ is the volumetric force. This force may be a gravitational, electrical, magnetic or electromagnetic.

Energy conservation equation

This equation defines that the variation of internal energy and kinetic energy in a control volume plus the flow of their variables must be equal to the work done on the control volume plus the incoming heat flow through the faces of the CV plus the energy of the sources in the control volume. Therefore, taking reference to the terms of Eq. 1.3:

- The first term $\frac{d}{dt}(\rho(u + e_c))$ represents the variation of the internal and kinetic energy in the CV.
- The second $\nabla \cdot ((u + e_c)\rho \overrightarrow{v})$ represents the energy flow of these variables through the faces of its volume.
- The third $-\nabla \cdot (\rho \overrightarrow{v})$ and fourth ∇ů$(\overrightarrow{v} \cdot \overrightarrow{\tau})$ terms are the work done by superficial forces like pressure and stress.
- The fifth $\nabla \cdot \overrightarrow{q}$ term is the incoming heat flow through the faces of the control volume.
- The sixth term $\rho \overrightarrow{g} \cdot \overrightarrow{v}$ represents the work done by the volumetric forces, in this case there is only the gravitational force work.
- The seventh term G is the work done by the internal forces.

1.1.2 Equation Formulation

The Convection-Diffusion equation is a combination of the conservation equations of mass, linear momentum and energy also called Navier-Stokes equations. In the last chapter we did the description of the equation in terms of temperature T and conductivity k now we can easily recast in terms of the general variable ϕ and its diffusion coefficient Γ, the only omission has been the convection term, which we shall now include.

The convection is created by fluid flow, our task is to obtain a solution for ϕ in the presence of a given flow field. Having somehow acquired the flow field we can calculate the temperature, concentration, enthalpy, or any such quantity that is represented by the general variable ϕ.

Simplified N-S Equations

The N-S explained in the previous section can be simplified for the conditions and assumptions related to our Convection-Diffusion equation formulation.[6] Then we can find the simplified N-S that govern the flow of a Newtonian fluid in Cartesian coordinates assuming:

- Two-Dimensional model
- Laminar flow
- Incompressible flow
- Newtonian fluid
- Boussinesq hypothesis[1]
- Negligible viscous dissipation
- Negligible compression or expansion work
- Non-participating medium in radiation
- Mono-component and mono-phase fluid

The use of constant properties of thermal conductivity, density... implies that we will not be able to solve problem with a huge range in temperatures because all of this properties depend on it.

Simplifying equations from Eq.1.1 to 1.3:

$$\frac{du}{dx} + \frac{dv}{dy} = 0 \tag{1.4}$$

$$\rho\frac{du}{dx} + \rho u\frac{du}{dx} + \rho v\frac{du}{dy} = -\frac{dp}{dx} + \mu\Big(\frac{d^2u}{dx^2} + \frac{d^2u}{dy^2}\Big) \tag{1.5}$$

$$\rho\frac{du}{dx} + \rho u\frac{dv}{dx} + \rho v\frac{dv}{dy} = -\frac{dp}{dy} + \mu\Big(\frac{d^2v}{dx^2} + \frac{d^2v}{dy^2}\Big) + \rho g\beta(T - T_\infty) \tag{1.6}$$

[1] Constant physical properties everywhere except in the body forces term

$$\rho\frac{dT}{dt} + \rho u\frac{dT}{dx} + \rho v\frac{dT}{dy} = \frac{k}{c_p}\Big(\frac{d^2T}{dx^2} + \frac{d^2T}{dy^2}\Big) + \frac{G}{c_p} \tag{1.7}$$

We can note that in this equation are four unknown values: pressure, temperature and the two components of velocity u and v. Furthermore, a boundary condition and an initial condition are required to solve the problem.

Analyzing the system of equations closely we can notice a strong coupling between them:

- Pressure - Velocity: for the previous established conditions, there is no specific pressure equation, but the pressure distribution allows the velocity field to satisfy the mass conservation equation.
- Temperature-Velocity: there is only a coupling characterization for natural convection, mixed connection or when the physical properties depend on the temperature. In forced convection and constant physical properties, the velocity field does not depend on temperature field.

Convection-Diffusion Equation

Acknowledging the coupling from the partial differential equations and applying the corresponding assumptions all equations from Eq.(1.4 - 1.7) can be summarized in the convection-diffusion equation:

$$\frac{d(\rho\phi)}{dt} + \nabla(\rho\overrightarrow{v}\phi) = \nabla(\Gamma\nabla\phi) + G \tag{1.8}$$

in Cartesian coordinates, incompressible flow and constant physical properties the equation can also be written as

$$\rho\frac{d\phi}{dt} + \rho u\frac{d\phi}{dx} + \rho v\frac{d\phi}{dy} = \frac{k}{c_p}\Big(\frac{d^2\phi}{dx^2} + \frac{d^2\phi}{dy^2}\Big) + G \tag{1.9}$$

In the previous equation, the first term is the accumulation of ϕ which tells how ϕ change along time. The second and the third term are the net convective flow in the control volume, which gives information about the spatial transport of ϕ. The sum of these has to be equal to the net diffusive flow, which represents the transport of ϕ due to the concentration of gradients, plus the generation of ϕ per unit volume (G). Looking at Eq.1.8 the *diffusion flux* due to the gradient of the general variable ϕ is $-\Gamma(d\phi/dx)$ where ϕ could represent chemical-species diffusion, heat flux, viscous stress, etc.

According to Eq.1.8 we can write a table with the parameters of ϕ, τ and G in order to reproduce the governing equations (Eq. 1.4 - Eq. 1.7).

Equation	ϕ	τ	G
Continuity	1	0	0
Momentum in X direction	u	μ	$-dp/dx$
Momentum in Y direction	v	μ	$-dp/dy + \rho g\beta(T - T_\infty)$
Energy (constant c_p)	T	k/c_p	ϕ/c_p

Table 1.1: Parameters to obtain N-S equations convection-diffusion equation

1.1.3 Equation Discretization

In this section it is shown the implicit finite-volume discretization (FVM) of the convection-diffusion equation. First of all Eq.1.9 has to be integrated into a rectangular CV, Fig.1.1 shows the geometric parameters for the intregration:

$$\frac{(\rho\phi)_P^1-(\rho\phi)_P^0}{\Delta t}\Delta x\Delta y+\Big[(\rho u\phi)_e^1-(\rho u\phi)_w^1\Big]\Delta y+\Big[(\rho v\phi)_n^1-(\rho v\phi)_s^1\Big]\Delta x=$$

$$=\left[\left(\Gamma\frac{d\phi}{dx}\right)_e^1-\left(\Gamma\frac{d\phi}{dx}\right)_w^1\right]\Delta y+\left[\left(\Gamma\frac{d\phi}{dy}\right)_n^1-\left(\Gamma\frac{d\phi}{dy}\right)_s^1\right]\Delta x+G_P^1\Delta x\Delta y \quad (1.10)$$

Note that superindex "1" is used for the value of property ϕ at time $t=t+\Delta t$ and "0" at the previous time step value, for an easier formulation we can state that $\phi^1=\phi$. We assume $\Delta x\Delta y$ as the CV volume V_P and separately as their surfaces S_e , S_w, S_n and S_s

$$\frac{(\rho\phi)_P-(\rho\phi)_P^0}{\Delta t}V_P+\Big[(\rho u\phi)_eS_e-(\rho u\phi)_wS_w\Big]+\Big[(\rho v\phi)_n^1S_n-(\rho v\phi)_sS_s\Big]=$$

$$=\left[\left(\Gamma\frac{d\phi}{dx}\right)_eS_e-\left(\Gamma\frac{d\phi}{dx}\right)_wS_w\right]+\left[\left(\Gamma\frac{d\phi}{dy}\right)_nS_n-\left(\Gamma\frac{d\phi}{dy}\right)_sS_s\right]+G_PV_P \quad (1.11)$$

This formulation can be simplified using the total flux term, defined by:

$$J_x=\rho v\phi-\Gamma\frac{d\phi}{dx} \quad (1.12a)$$

$$J_y=\rho v\phi-\Gamma\frac{d\phi}{dy} \quad (1.12b)$$

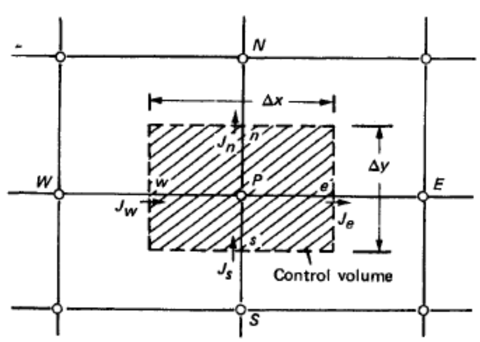


Figure 1.1: Two-Dimensional CV with flux vectors (J)[1]

Equation 1.8 can be expressed with the flux term J as:

$$\frac{d(\rho\phi)}{dt} + \frac{dJ_x}{dx} + \frac{dJ_y}{dy} = G \tag{1.13}$$

Integrating the previous equation into a rectangular CV and assuming an implicit scheme for the temporal integration, Eq. 1.13 yields to:

$$\frac{(\rho\phi)_P - (\rho\phi)_P^0}{\Delta t} V_P + J_e - J_w + J_n - J_s = (G_c + G_P\phi_P)V_P \tag{1.14}$$

The quantities J_e , J_w , J_s and J_n are the integrated total fluxes over the control-volume faces; that is, J_e stands for $\int J_x dy$ over the interface e and so on. The source term has been linearized as we can see in the last term of Eq. 1.14.

We need to note that the flow field has to satisfy the continuity equation (Eq. 1.4) in order to assume convergence:

$$\frac{d}{dx_j}(\rho u_j) = 0 \tag{1.15}$$

Integrating over a rectangular finite volume:

$$\frac{\rho_P - \rho_P^0}{\Delta t} V_P + F_e - F_w + F_n - F_s = 0 \tag{1.16}$$

where F_e, F_n, F_s and F_w are the mass flow rates through the faces of the Control Volume.

$$F_e = (\rho u)_e S_e \tag{1.17a}$$

$$F_w = (\rho u)_w S_w \tag{1.17b}$$

$$F_n = (\rho v)_n S_n \tag{1.17c}$$

$$F_s = (\rho v)_s S_s \tag{1.17d}$$

Multiplying Eq.1.16 by ϕ_P and subtracting it from Eq.1.14:

$$\begin{aligned}(\phi_P - \phi_P^0)\frac{\rho_P^0}{\Delta t} \cdot V_P + (J_e - F_e\phi_P) - (J_w - F_w\phi_P) + \\ + (J_n - F_n\phi_P) - (J_s - F_s\phi_P) = (S_c + S_P\phi_P)\end{aligned} \tag{1.18}$$

The assumption of uniformity over a control-volume face enables us to employ One-Dimensional practices from Ref.[1] for the Two-Dimensional situation.

1.1.4 Numerical Schemes

Numerichal schemes in convection-diffusion problems evaluate the convective and diffusive terms at the CV faces while the dependent variable ϕ is evaluated at the center. Convective flux on any face is given by the arithmetic mean between central node and his neighbours:

$$\left(\frac{d\phi}{dx}\right)_w = \frac{\phi_W - \phi_P}{\delta x_w} \tag{1.19a}$$

$$\left(\frac{d\phi}{dx}\right)_e = \frac{\phi_E - \phi_P}{\delta x_w} \tag{1.19b}$$

$$\left(\frac{d\phi}{dy}\right)_n = \frac{\phi_N - \phi_P}{\delta y_n} \tag{1.19c}$$

$$\left(\frac{d\phi}{dy}\right)_s = \frac{\phi_S - \phi_P}{\delta y_s} \tag{1.19d}$$

Convective and diffusive terms need to be calculated using numerical schemes that evaluates values of ϕ at the nodal points. There are two types of schemes: low order numerical schemes; and high order schemes. The *order* of a numerical scheme is the number of neighbour nodes that are involved to evaluate the dependent variable at the cell face.

For the scope of this project we are only going to treat low order numerical schemes such as: CDS, UDS, HDS, EDS, PLDS. This numerical schemes evaluate the variable using nearest nodes (east (E), west(W), nort(N) and south (S)) with a scheme order of one or two. The *order* of a numerical scheme is defined by the number of neighbouring nodes that are used to evaluate the dependent variable at the cell face. Figure 1.2 shows the values of the variable ϕ given by the different schemes for various values of the Peclet Number (Pe).

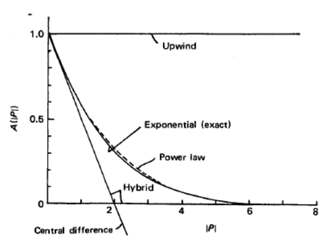


Figure 1.2: The function $A(|Pe|)$ for various low order schemes [1]

Central Difference Scheme (CDS)

It is a second order scheme where the variable at the cell face is calculated as the arithmetic mean of the variable at the neighbour nodes of the face. For the east face of the CV:

$$\phi_e = \frac{1}{2}(\phi_P + \phi_E) \tag{1.20}$$

Scheme	Formula for $A(\vert Pe\vert)$
Central difference	$1 - 0.5\vert Pe\vert$
Upwind	1
Hybrid	$[\![0, 1 - 0.5\vert Pe\vert\ ]\!]$
Power Law	$[\![0, (1 - 0.1\vert Pe\vert)^5\ ]\!]$
Exponential	$\vert Pe\vert/(e^{\vert Pe\vert} - 1)$

Table 1.2: The function $A(|Pe|)$ for different schemes [1]

Using a general notation to reefer to the control volume face or center:

$$\phi_{if} = \frac{1}{2}(\phi_P + \phi_{ib}) \tag{1.21}$$

Looking at Fig.1.2 we note that all schemes except the CDS give physically realistic solutions because it can produce values that lie outside the $[0-1]$ range established by the Scarborough criterion, see Appendix **??**. We can find the formula of $A(|Pe|)$ for this scheme in Table 1.2:

$$A(|Pe_{if}|) = 1 - 0.5|Pe_{if}| \tag{1.22}$$

Upwind Difference Scheme (UDS)

It is a first order scheme where the value of ϕ at the cell face is equal to the value of ϕ at the grid point on the upwind side of the face.

$$\phi_e = \phi_P \quad if \quad F_e > 0 \tag{1.23a}$$

$$\phi_e = \phi_E \quad if \quad F_e < 0 \tag{1.23b}$$

What that means is that if u is positive, the value of ϕ at the face will be the value of ϕ at the left grid point. However, if u is negative, the value of ϕ at the face cell will be the value of ϕ at the right grid point. It will be the same reasoning for v. It is defined a new operator for this criterion as $[\![A, B]\!]$. Thus,

$$F_e\phi_e = \phi_P[\![F_e, 0]\!] - \phi_E[\![-F_e, 0]\!] \tag{1.24}$$

This scheme solves the problem that the CDS has because all the coefficients in the equations are always positive or null. That means that the Scarborough criterion is always satisfied. We can find the formula of $A(|Pe|)$ for this scheme in Table 1.2:

$$A(|Pe_{if}|) = 1 \tag{1.25}$$

Hybrid Difference Scheme (HDS)

It is a combination of central difference scheme and upwind difference scheme as it exploits the favorable properties of both of these schemes. This scheme uses CDS for low velocities and UDS for high velocities, it consists in approximating the value of the dimensionless form of a_E (Eq. 1.26) to three linear zones. Fig. 1.3 shows this approximation.

$$\frac{a_E}{D_e} = \frac{Pe_e}{exp(Pe_e) - 1} \tag{1.26}$$

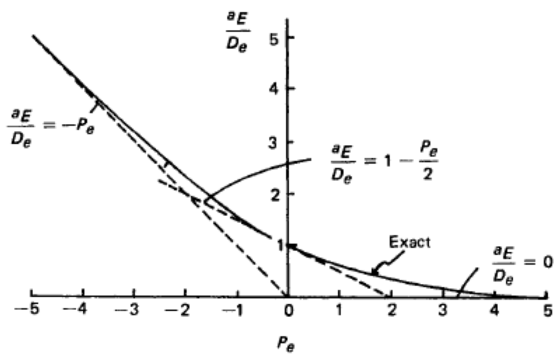


Figure 1.3: Variation of coefficient a_E with Peclet number [1]

For positives values of Pe_e the grid point E is the *downstream* neighbor and its influence is seen to decrease as Pe_e increases. When Pe_e is negative the point E is the *usptream* neighbor and has a large influence. The tree straight lines represent the three limiting cases, they can be seen to form an envelope of, and represent a reasonable approximation to, the exact curve. Then,

For $Pe_e < -2$,

$$\frac{a_E}{D_e} = -Pe_e \tag{1.27}$$

For $-2 \leq Pe_e \leq -2$,

$$\frac{a_E}{D_e} = 1 - \frac{Pe_e}{2} \tag{1.28}$$

For $Pe_e > 2$,

$$\frac{a_E}{D_e} = 0 \tag{1.29}$$

This expressions can be compacted into the following form[2]:

$$a_E = D_e \left[\!\left[-Pe_e, 1 - \frac{Pe_e}{2}, 0 \right]\!\right] \tag{1.30a}$$

$$a_E = \left[\!\left[-F_e, D_e - \frac{F_e}{2}, 0 \right]\!\right] \tag{1.30b}$$

We need to note that it is identical with the central-difference scheme for the Peclet number range $-2 \leq Pe_e \leq 2$, and outside this range it reduces to the upwind scheme in which the diffusion has been set equal to zero. For that reason and from Table 1.2 the function of $A(|Pe|)$ is

$$A(|Pe_i f|) = [\![0, 1 - 0.5|Pe_i f|]\!] \tag{1.31}$$

[2] This special symbol $[\![]\!]$ stands for the largest of the quantities contained within it.

Exponential Difference Scheme (EDS)

It is a second order scheme where the evaluation of the variables at the cell faces come from te exact solution of the Eq. 1.9 for the steady One-Dimensional problem without source term [6]. From [1] we know that exact solution is:

$$\frac{\phi - \phi_0}{\phi_L - \phi_0} = \frac{exp(Pe \cdot x/L) - 1}{exp(Pe) - 1} \tag{1.32}$$

Where ϕ_0 is the value of ϕ at the left boundary ($x = 0$); ϕ_L the value of ϕ at the right boundary ($x = L$); x is the position of the left interface node; Pe is the Peclet number; and L is the distance of the domain ($0 \leq x \leq L$). Remember that the Peclet number is defined by:

$$Pe \equiv \frac{\rho u L}{\Gamma} \tag{1.33}$$

From Eq.1.32 it can be seen that P is the ratio of strengths of convection and diffusion, which gives us a better understanding about the meaning of this number inside the case of study. The nature of Eq.1.32 ac be understood from Fig. 1.4 where the variation of $\phi \sim x$ for different values of the Peclet number is shown.

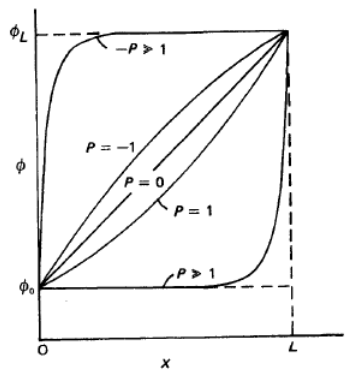


Figure 1.4: Exact solution for the one-dimensional convection-diffusion problem [1]

This scheme gives an exact solution for 1D for any Peclet number, although it is not exact for the 2D and 3D situations. Another disadvantage would be the extra time it takes to compute the solution with exponential functions. We can find the formula of $A(|Pe|)$ for this scheme in Table1.2:

$$A(|Pe_{if}|) = |Pe_i f|/(e^{|Pe_i f|} - 1) \tag{1.34}$$

Power-law Difference Scheme (PLDS)

Taking the HDS it seems a little premature to set the diffusion effects equal to zero as soon as the Peclet number exceeds 2, a better aproximation to the exact curve is givven by the power-law scheme. It is a second order scheme where the variable at the cell face is calculated with an approximation of the EDS by a polinomial of fifth degree.

From [1] we can get the compact form for the coefficient a_E:

$$a_E = D_e \left[\!\!\left[0, \left(1 - \frac{0.1|F_e|}{D_e}\right)^5 \right]\!\!\right] + [\![0, -F_e]\!] \tag{1.35}$$

We can find the formula of $A(|Pe|)$ for this scheme in Table1.2:

$$A(|Pe_{if}|) = [\![0, (1 - 0.1|Pe_i f|)^5]\!] \tag{1.36}$$

1.1.5 Final Discretization Equation

From Eq. 1.18 and according to [1, 6] the Two-Dimensional final discretization equation can now be rewritten as

$$a_P\phi_P = a_E\phi_E + a_S\phi_S + a_W\phi_W + a_N\phi_N + b \tag{1.37}$$

Where the coefficients a_i can be evaluated as:

$$a_E = D_e \cdot A(|Pe_e|) + [\![-F_e, 0]\!] \tag{1.38a}$$

$$a_W = D_w \cdot A(|Pe_w|) + [\![F_w, 0]\!] \tag{1.38b}$$

$$a_N = D_n \cdot A(|Pe_n|) + [\![-F_n, 0]\!] \tag{1.38c}$$

$$a_S = D_s \cdot A(|Pe_s|) + [\![F_s, 0]\!] \tag{1.38d}$$

$$a_P^0 = \frac{\phi_P^0 V_P}{\Delta t} \tag{1.38e}$$

$$b = G_C V_P + a_P^0 \phi_P^0 \tag{1.38f}$$

$$a_P = a_E + a_S + a_W + a_N + a_P^0 - G_P V_P \tag{1.38g}$$

Where the flow rates through the faces (F_{if}) are:

$$F_e = (\rho u)_e S_e \tag{1.39a}$$

$$F_w = (\rho u)_w S_w \tag{1.39b}$$

$$F_n = (\rho v)_n S_n \tag{1.39c}$$

$$F_s = (\rho v)_s S_s \tag{1.39d}$$

The corresponding conductances are defined by

$$D_e = \frac{\Gamma_e S_e}{(\delta x)_e} \tag{1.40a}$$

$$D_w = \frac{\Gamma_w S_w}{(\delta x)_w} \tag{1.40b}$$

$$D_n = \frac{\Gamma_n S_n}{(\delta x)_n} \tag{1.40c}$$

$$D_s = \frac{\Gamma_s S_s}{(\delta x)_s} \tag{1.40d}$$

and the Peclet numbers by

$$P_e = \frac{F_e}{D_e} \quad P_n = \frac{F_n}{D_n} \quad P_s = \frac{F_s}{D_s} \quad P_w = \frac{F_w}{D_w} \tag{1.41}$$

We need to note that all the formulation has been done in order that the value of $A(|Pe_i f|)$ depends on the numerical scheme used. This value can be found in Table 1.2.

1.2 Numerical Grid

No specific information has been provided as to where the control volume faces are located in relation to the grid points, since the discretization equations have been displayed in general terms so that it will be applicable to any particular way of locating the control volume faces. There are many possible ways for locating the control-volume but for the aim of this thesis we are going to focus in two different grids. The description of each one will refer to a two-dimensional situation, although the concepts involved are applicable to one and three-dimensional situations.

1.2.1 Grid A: Faces located midway between the grid points

One of the most intuitive practise to construct the control volume is to place their faces *midway* between neighboring grid points as we can see in Fig.1.5a. For building this grid to a 2-D plate we should place grid points on his boundaries. Another observation is that the grid is nonuniform; on consequence the grid point P does not lie at the geometric center of the control volume.

1.2.2 Grid B: Grid points placed at the centers 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 center of each control-volume. As we can see in the Fig.A.3b when the control volume sizes are non-uniform, their faces does not lie midway between the grid points.

The fact that the grid point P in Fig.1.5a may not be at the geometric center of the control volume represents a disadvantage. That is because 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[1]. The Grid A also presents objections in the calculation of the heat fluxes at the control volume faces, if we take grip point e in Fig.1.5a, for example, we can se that it is not at the center of the control-volume face i which it lies. Then, we assume that the heat flux at e prevails over the entire face brings some inaccuracy.

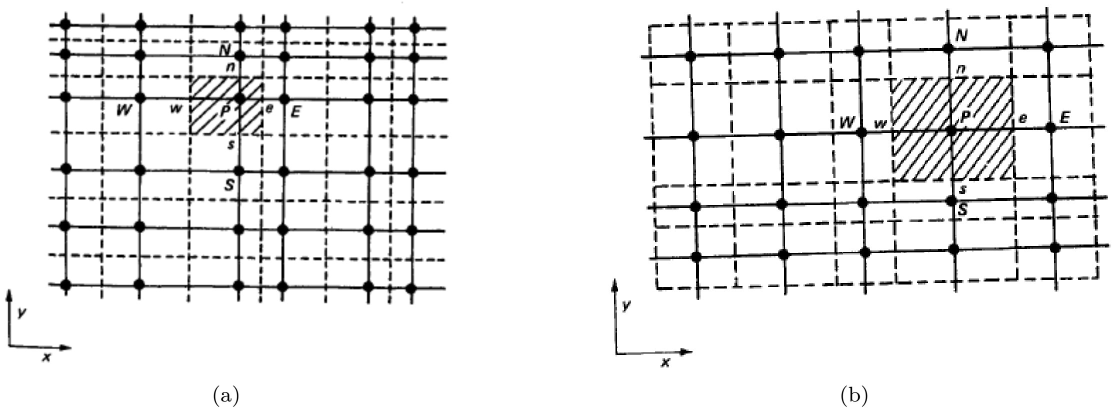


Figure 1.5: Location of control-volume faces (a)Grid A (b)Grid B [1]

Grid B does not have this problems because the point P lies at the center of the control volume by definition and points such as e lies at the center 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 from the Grid A over Grid B but the aforementioned advantages that Grid B represents over the grid A makes us consider that the election of Grid B for our problem formulation is going to be the most suitable. We need 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. We can see this arrangement of the Grid B in Fig. 1.6 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.

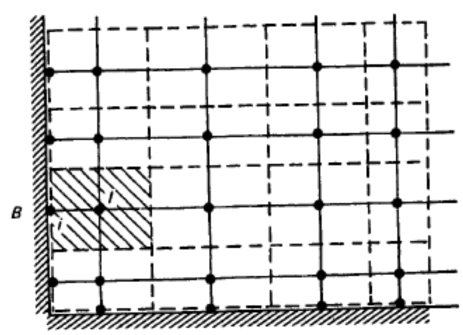


Figure 1.6: Boundary control volumes in Practice B[1]

Chapter 2

Case of Study

Contents

2.1 Convection-Diffusion Solenoidal Flow Problem

In this section it is going to be developed the concepts explained in Section 1.1 applied to a practical case where the convection-diffusion phenomena is involved proposed by the CTTC. This problem can also be called Smith-Hutton Problem.

This is a recirculating flow problem which involves streamline curvature studied by Smith and Hutton. In their study they concluded that in a high-convection regime modelling ”remains the art of compromise between diffusive and oscillatory errors”.[7]

2.1.1 Problem Definition

This problem is based in a two-dimensional test problem devised by Smith and Hutton which concerns steady-state convection and diffusion of a scalar field ϕ in a prescribed velocity field $\overrightarrow{v}$ with a known constant diffusivity D. The objective is to find the field ϕ for a given value of the relation ϕ/Γ. Figure 2.1 shows a visual scheme of the problem.

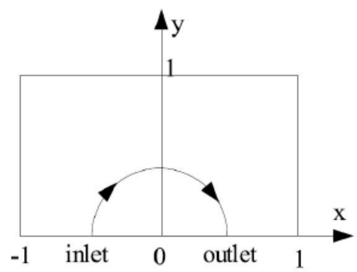


Figure 2.1: Smith-Hutton problem

As we can see in Fig.2.1 the flow domain considered is a rectangle: $-1 \leq x \leq 1$, $0 \leq y \leq 1$. And the velocity field $\overrightarrow{v}$ is given by

$$u(x,y) = 2y(1-x^2) \tag{2.1a}$$
$$v(x,y) = -2x(1-y^2) \tag{2.1b}$$

2.1.2 Boundary Conditions

Table 2.1 gives us the boundary conditions for the parameter ϕ in our case of study.

Field ϕ	$x[m]$	$y[m]$
$\phi = 1 + tanh[(2x+1)\alpha]$	$-1 < x < 0$	$y = 0$
	$x = -1$	$0 < y < 1$
$\phi = 1 + tanh(\alpha)$	$-1 < x < 1$	$y = 1$
	$x = 1$	$0 < y < 1$
$d\phi/dy = 0$	$0 < x < 1$	$y = 0$

Table 2.1: Boundary conditions for Smith-Hutton Problem ($\alpha = 10$)

As we can see in the plots shown in Fig. 2.2 the hyperbolic tangent function[1] in the inlet boundary condition ($-1 < x < 0$, $y = 0$) gives a values of ϕ almost 0 for $-1 < x < -0.5$ and rapidly grows to a 2 value for $-0.5 < x < 0$. For all the other boundaries the value of $\phi$is approximated to 0.

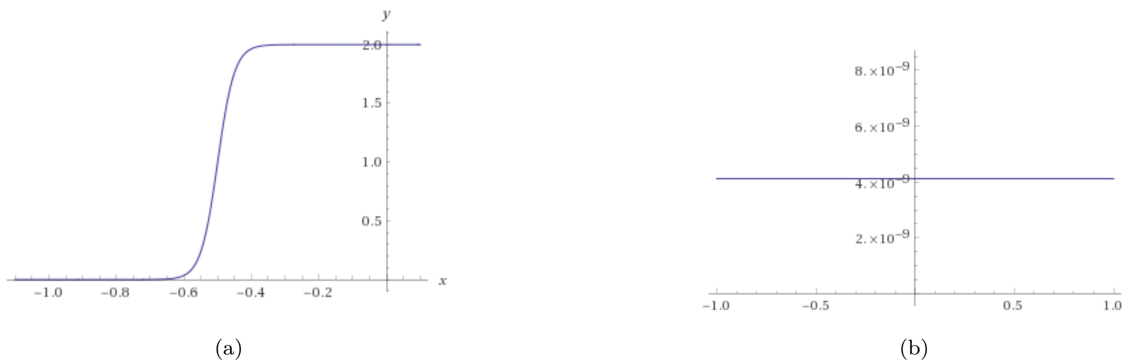


Figure 2.2: Plots for the hyperbolic tangent functions (a)Inlet (b)No flow boundaries

2.1.3 Discretization

From Section 1.1.2 we can rewrite the convection-diffusion equation (Eq.1.9) considering the source term value as zero:

$$\rho\frac{d\phi}{dt} + \rho u\frac{d\phi}{dx} + \rho v\frac{d\phi}{dy} = \frac{k}{c_p}\Big(\frac{d^2\phi}{dx^2} + \frac{d^2\phi}{dy^2}\Big) \tag{2.2}$$

The implicit coefficient form of the previous equation is known from Section 1.1.5

$$a_P\phi_P = a_E\phi_E + a_S\phi_S + a_W\phi_W + a_N\phi_N + b \tag{2.3}$$

Even though our problem asks for the steady state condition, the terms that contain time dependency were taken into account because we can obtain more conclusions about time evolution of the phenomena and his computational cost. The scheme chosen to solve the convection-diffusion equation is a implicit scheme because it gives physically satisfactory results . The spatial discretization and control volume geometry chosen for our case of study are shown in Table 2.2.

Spatial Property	L	H	N_x	N_y	Δx	Δy
Value	$[-1, 1]$	1	200	100	0.01	0.01

Table 2.2: Domain spatial discretization for Smith-Huton Problem

[1]Smith and Hutton proposed $\alpha = 10$ as representative of a relatively sharp transmission[7]

In Section **??** it is said that the mesh used for this thesis is the Grid B shown in Fig.1.6. In this disposal of the grid points its important to note that there are grid nodes around all the boundary conveniently located at wall faces of the control volume, this mesh structure is saved in a matrix of dimensions $[N_x + 2][N_y + 2]$. The boundary condition information is saved inside this extra dimensions. Using this Grid allows the direct determination of boundary conditions and an easier analysis of the coefficients at this points. It is important to take into account that the distance between between this nodes and its neighbors is half of the central grid nodes.

Finally, it is seen that in Table 2.2 the control volume dimensions are the same ($\Delta x = \Delta y$) because the number of nodes for the y-dimension are the half as the domain length. Selecting the same number of nodes in both dimensions would suppose adding more importance to the y-dimension which is not useful for the computational performance of the code.

2.1.4 Algorithm

With the purpose of understanding the algorithm developed for solving this problem Fig.2.3 gives us an idea about the main processes inside the code. For this problem a Matlab Object Oriented code[2] has been developed as a first contact with this programming method. The core of the code is the Main function, which contains all the needed methods and input data. Inside the main we can find four principal functions:

- Uniform Mesh: in charge of the domain discretization and compute of the velocity field needed for each problem.
- Coefficient Compute: in charge of computing the needed coefficients for each case, the ones that are dependent on the field ϕ and the non-dependent.
- Solver: it is in charge of finding the value of ϕ for the coefficients previously calculated.
- Solver Shell: that function returns us the final results for the ϕ field. It contains the Solver.

The needed data for starting the computations is modified from the "inputData" file. It contains the points that define our domain (P_1, P_2), the requested solutions points, the sizes of our mesh (N_x, N_y), the initial field ϕ value and his boundary conditions, the physical properties needed for solving the coefficients and finally the solver parameters.

Figure 2.4 shows another diagram in order to represent the transient state of the Smith-Hutton problem. The first program iterated until the steady state was reached without taking into account what happened each time step. With this new algorithm it is possible to represent the evolution of the problem along the time steps. The code used for developing this program was the same as the first but with some modifications inside the solver shell. The developed Matlab OOP code can be found inside Code Attachments document.

[2]This code can be downloaded with a document that describes the case of study by clicking "here".

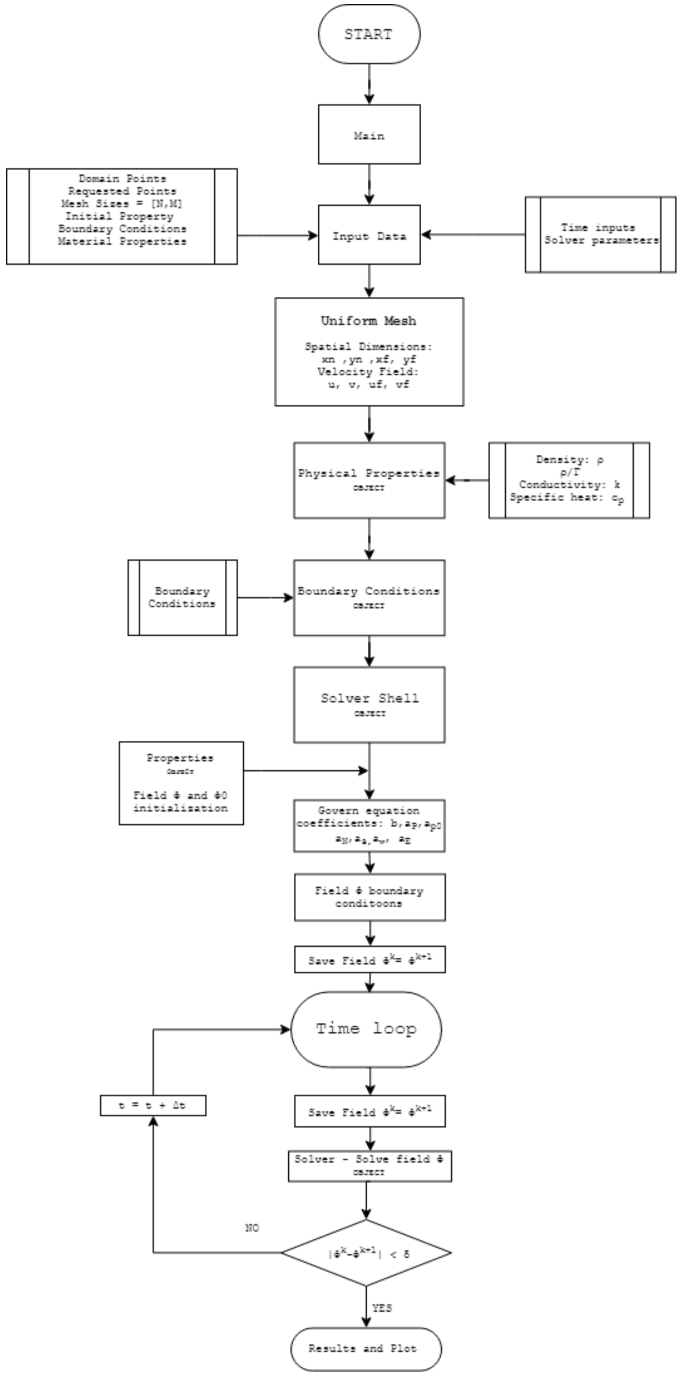


Figure 2.3: Smith-Hutton Steady State Problem algorithm flowchart

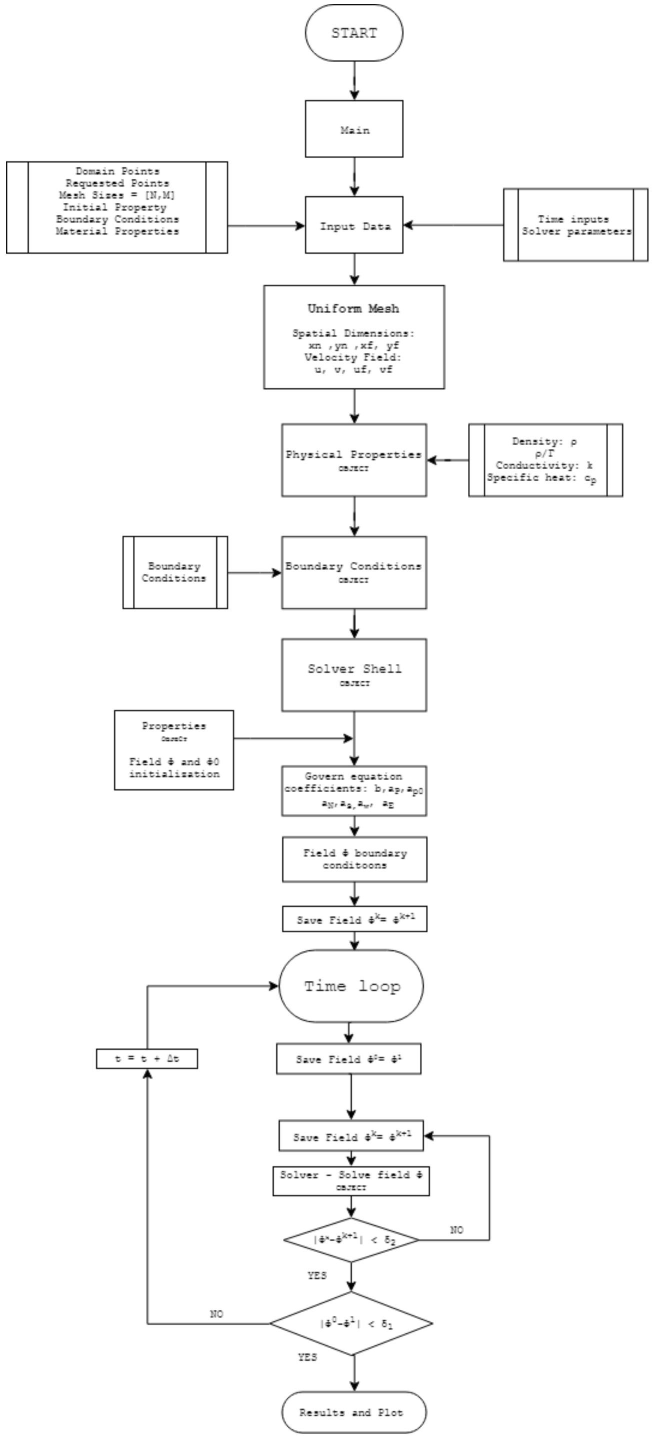


Figure 2.4: Smith-Hutton Transient Problem algorithm flowchart

2.1.5 Results

Once our code is running and working correctly it is needed to compare the results obtained at the outlet of our contour with numerical results provided by [2] in order to check their validity. The results are displayed in Table 2.3 and we can see the complete field for each situation in the plots shown below (Fig. 2.5-A.7).

This results were obtained using the Upwind Numerical Scheme in the computation of our coefficients. There are different possible and more optimal schemes to apply. In this study we only need to check if our correct gives the correct results for each case and for that we don't need to apply different N-S or Solvers. In this case a Gauss-Seidel solver was implemented because its programming simplicity but the algorithms developed in the code are easy to attach with any iterative solver type explained in this thesis.

	$\rho/\Gamma = 10$		$\rho/\Gamma = 1000$		$\rho/\Gamma = 10^6$	
Position x	Expected	Calculated	Expected	Calculated	Expected	Calculated
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	
0.3	0.775		1.8410		1.000	
0.5	0.621		0.9510		0.036	
0.6	0.480		0.1546		0.001	
0.7	0.349		0.0010		0.000	
0.8	0.227		0.0000		0.000	
0.9	0.111		0.0000		0.000	
1.0	0.000		0.0000		0.000	

Table 2.3: Numerical results at the outlet for different ρ/Γ [2]

For more plots you can check Attachment B.

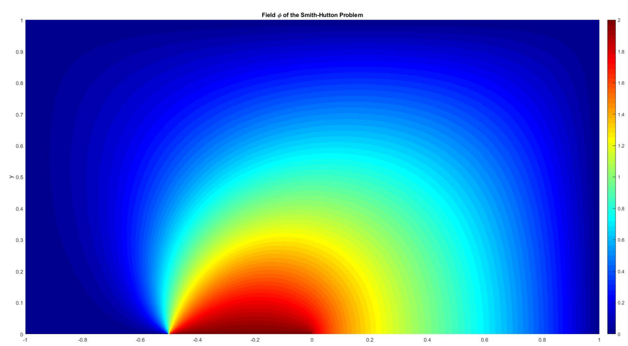


Figure 2.5: Field ϕ of the Smith-Hutton Problem for $\rho/\Gamma = 10$

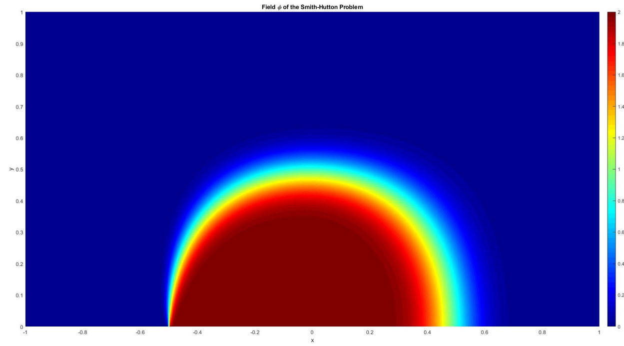


Figure 2.6: Field ϕ of the Smith-Hutton Problem for $\rho/\Gamma = 1000$

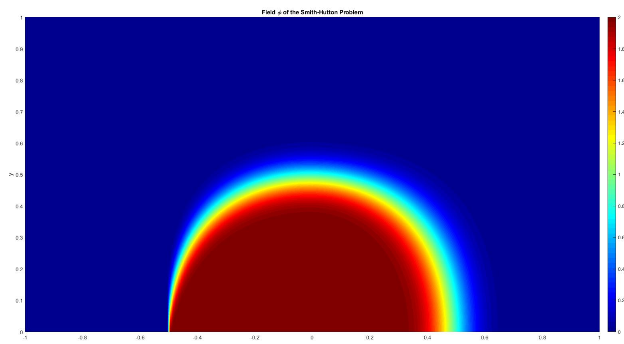


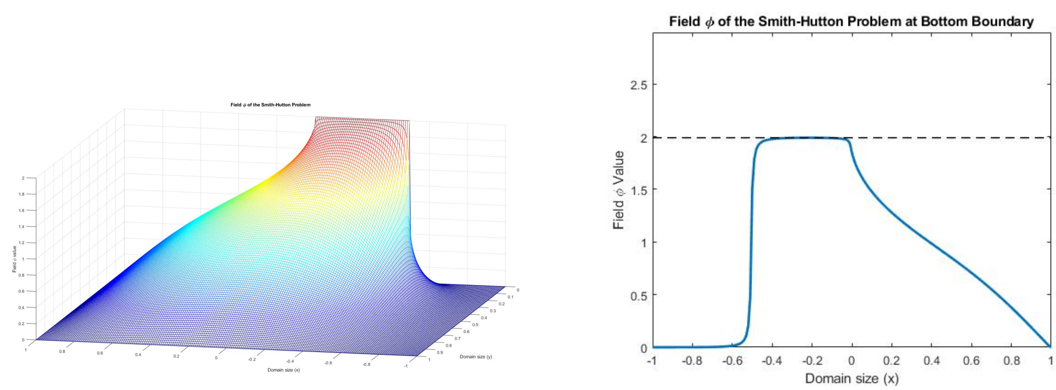
Figure 2.7: Field ϕ of the Smith-Hutton Problem for $\rho/\Gamma = 10^6$

Chapter 3

Conclusions

As the ρ/Γ ratio increases, the convective term grows taking a predominant role against the diffusive term, wich decreases. This behaviour can be observed in the field ϕ plots shown in the figures below. In the first case where $\rho/\Gamma = 10$ we know from the Peclet Eq. 1.4 that for this values the Peclet number is low. Which means that for low Peclet numbers the problem tens to have greater diffusive effects. But as Peclet number increases the convective term gains influence, for that reason a solenoidal field is observed for the cases of greater ϕ/Γ.

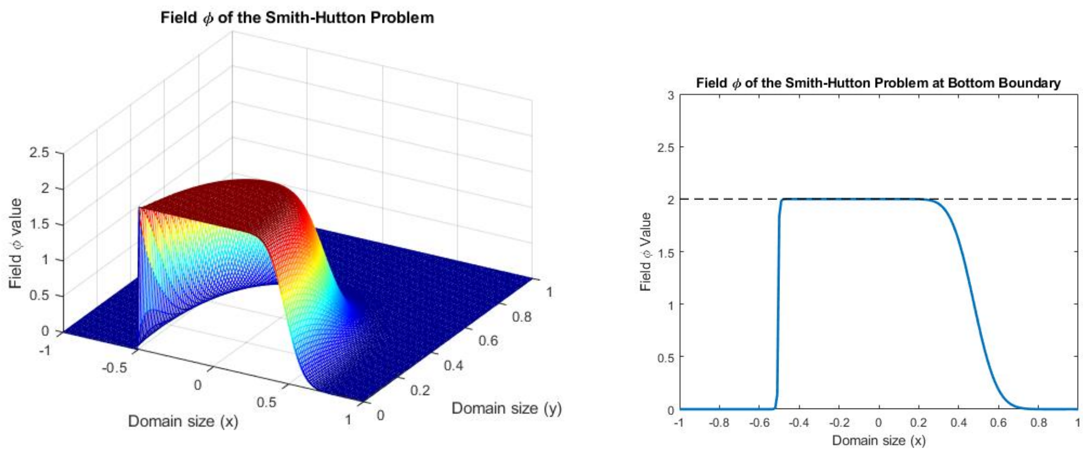
In the following figures we can see some plots that shows us the nature of the results at the "outlet". Each one shows the evolution of the ϕ field values at the bottom boundary nodes. It is seen that the maximum values for this field are defined by the inlet boundary condition at the right half of the inlet for each case. This plots are shown in order to give a deeper vision about the nature of the results.



(a) (b)

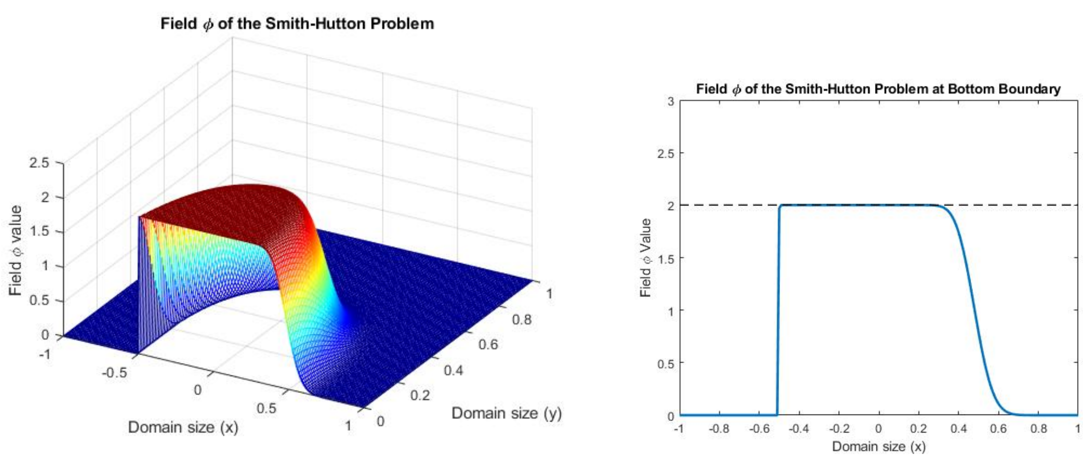
Figure 3.1: Evolution of the "outlet" Field ϕ for $\rho/\Gamma = 10$
(a)3-D Field (b)2-D Plot

In the first case, for low values of ρ/Γ, the diffusive term has the main role. For this reason the field value at the Point $(0,0)$ rapidly decreases untill it reaches value zero at the end of the end of the boundary. This results could be explained from the particle concentration view, thus it is an almost diffusive problem, the particle density is higher what means that our flow of study wont easily reach the form of the velocity field. Figure A.3a shows the Field ϕ values in a 3D plot as an extra way of analyzing the plots previously shown. .



(a) (b)

Figure 3.2: Evolution of the "outlet" Field ϕ for $\rho/\Gamma = 1000$
(a)3-D Field (b)2-D Plot



(a) (b)

Figure 3.3: Evolution of the "outlet" Field ϕ for $\rho/\Gamma = 10^6$
(a)3-D Field (b)2-D Plot

For the other cases (Fig 3.2 - 3.3) we are working with higher values of ρ/Γ. Thus the nature of the results would be convective, the comparison of this cases confirms that because there is almost no difference between results in both situations. In this cases, the fluid particles are able to follow easily the path marked by the velocity field. In the Field ϕ plot of the bottom boundary it is seen a symmetry of the results to the Y axis. This symmetry is imposed by the velocity to our domain and the bigger the value of ρ/Γ we use, the faster this symmetry is achieved. In a physical sense this would mean that the fluid particles can easily follow the path described by the velocity field. This symmetry could be even better if we had not directly supposed that

$$\phi = 2 \quad for \quad -0.5 < x < 0 \tag{3.1}$$

that is why we see a step in the plots presented.

After the extraction of this conclusions from the development of this case, it has been noticed that the density for each case of study have been kept to $\rho = 10kg/m^3$. Realizing that, some studies were realized modifying the density value but there were not variations in the results except in the isobaric plot for the $\phi/\Gamma = 10^6$ situation. For this case ($\rho = 10^6$) the field ϕ maintains the value of the second half of the inlet in his nearby region as we can see in Fig.A.8.

In the code developed we solve the Smith-Hutton problem with a transitory analysis of the convection-diffusion equation. Until now this is not mentioned because all the results presented corresponded to the stationary situation but it is noticed that some results changes for the previous case. The analysis of the nature of this results is left for further studies.

References

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

[2] D. CTTC, *Validation of the Convection-Diffusion Equation, Tech. rep. , ESEIAAT.*

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

[4] K. W. Morton, *Numerical solution of convection-diffusion problems*, 1st ed., 1996.

[5] F. M. White, *Fluid Mechanics*, 5th ed.

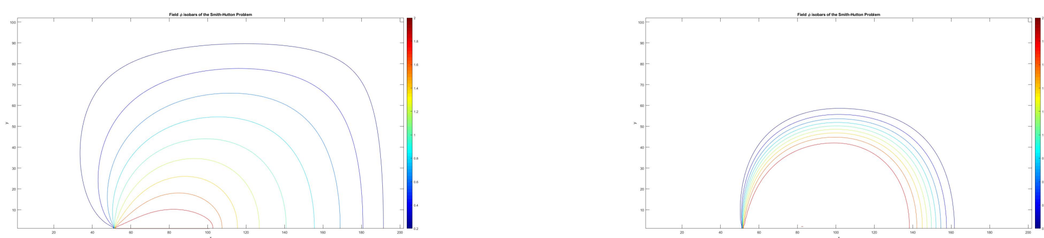
[6] D. CTTC, *Numerical solution of convection. Tech. rep. , ESEIAAT*, 2010.

[7] B. Leonard and S. Mokhtari, *ULTRA-SHARP Solution of the Smith-Hutton Problem.* Department of Mechanical Engineering, The University of Akron, 1992.

Appendix A

Smith-Hutton Problem Results

In this Appendix we can find isobaric and mesh plots of the Smith-Hutton Problem. This results has been obtained using a self developed code[1] with the numerical scheme and solver indicated in the case of study.



(a) (b)

Figure A.1: Field ϕ isobars of the Smith-Hutton Problem for
(a)$\rho/\Gamma = 10$ (b) $\rho/\Gamma = 10$

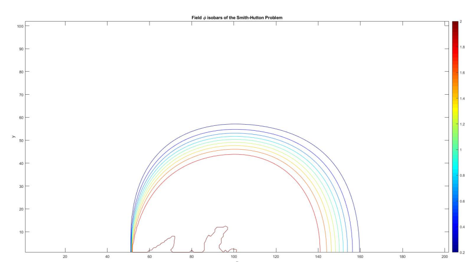
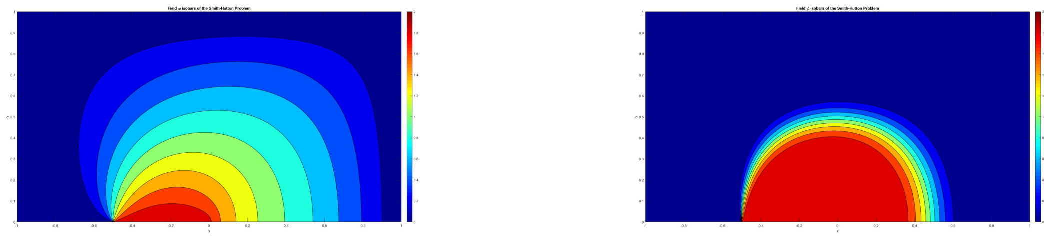


Figure A.2: Field ϕ isobars of the Smith-Hutton Problem for $\rho/\Gamma = 10^6$

[1]This code can be downloaded with a document that describes the case of study by clicking ”here”.



(a) (b)

Figure A.3: Field ϕ color isobars of the Smith-Hutton Problem for (a)$\rho/\Gamma = 10$ (b) $\rho/\Gamma = 10$

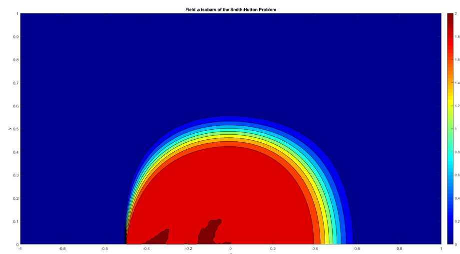


Figure A.4: Field ϕ color isobars of the Smith-Hutton Problem for $\rho/\Gamma = 10^6$

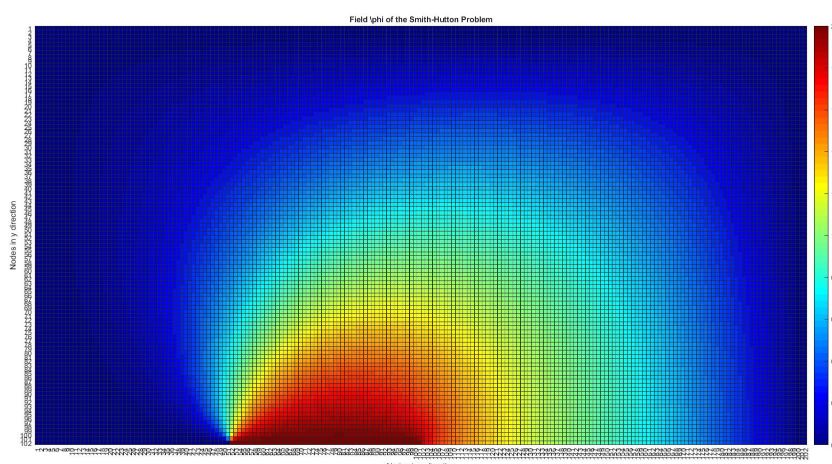


Figure A.5: Field ϕ grid of the Smith-Hutton Problem for $\rho/\Gamma = 10$

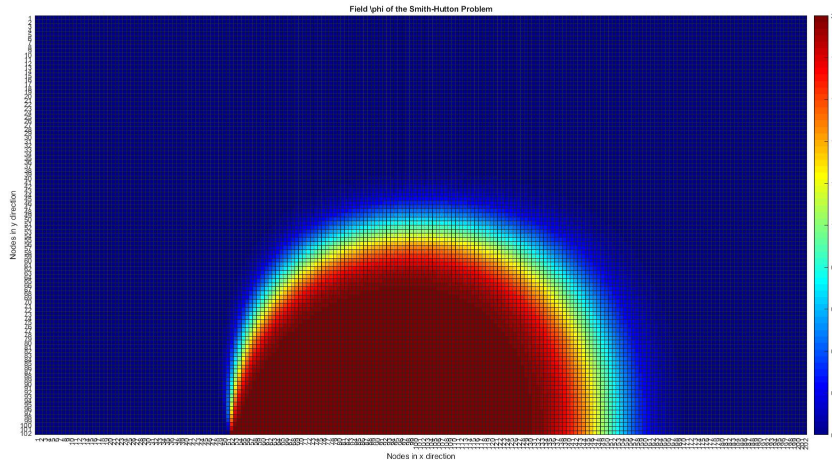


Figure A.6: Field ϕ grid of the Smith-Hutton Problem for $\rho/\Gamma = 1000$

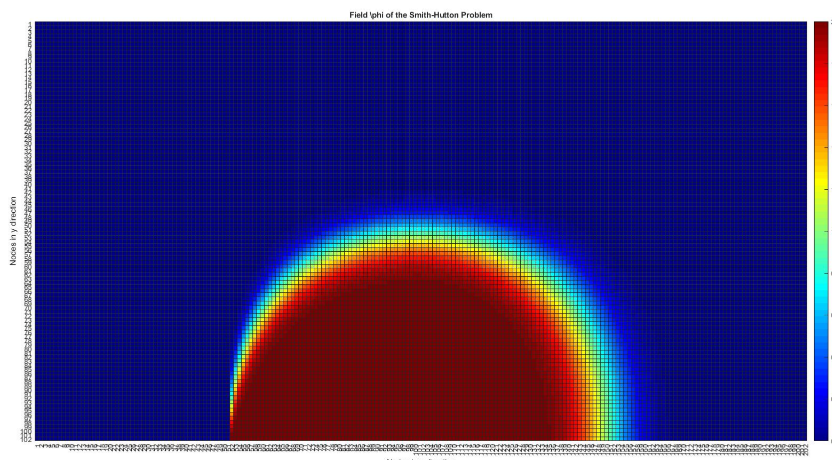


Figure A.7: Field ϕ grid of the Smith-Hutton Problem for $\rho/\Gamma = 10^6$

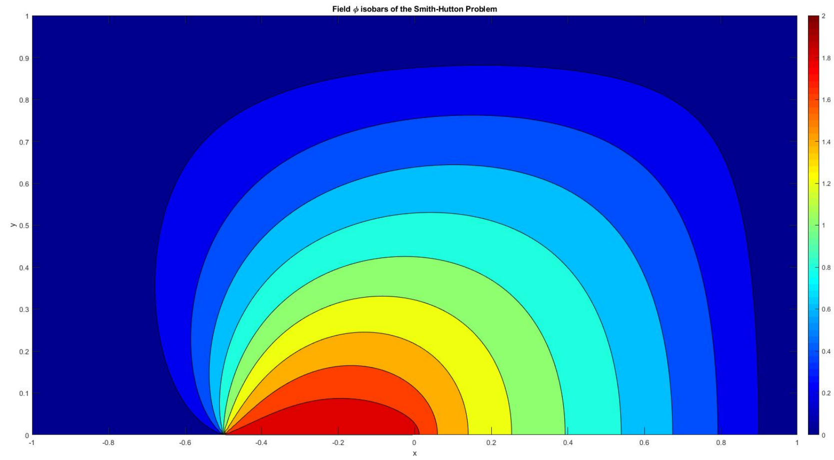


Figure A.8: Field ϕ of the Smith-Hutton Problem for $\rho/\Gamma = 10^6$ and $\rho = 10^6$

Appendix B

Smith-Hutton Problem Code 1

In this Appendix we can see the code developed with Matlab OOP for solving the Smith-Hutton Problem. This results has been obtained using a self developed code[1] with the numerical scheme and solver indicated in the case of study. This code was made in the purpose of studding the steady state of the Smith-Hutton problem.

B.1 Main Algorithm

```
% ---------EXERCICE 4 CODEv4----------- %

clear all
more off

%Load input data
InputData

tic;
mesh=UniformMesh(domainPoints,meshSizes);
fprintf('MeshTime %f\n',toc); tic;

physProp=PhysProp(mesh,rhogamma,cp,k,rho);
fprintf('PhysPropTime %f\n',toc); tic;

boundCond=BoundCond(inletProp, outletProp, leftProp, rightProp, upperProp);
fprintf('BoundCondTime %f\n',toc); tic;

tcd2D=TransientConvectionDiffusion2D(mesh, physProp, boundCond, timeStep, ...
    initProp, refTime);
fprintf('CreateTHC2DTime %f\n',toc); tic;

[PropReqPoints,timeReqPoints]=tcd2D.solveTime(lastTime, reqPoints, maxIter, ...
    maxDiff,PostProcess);
```

[1]This code can be downloaded with a document that describes the case of study by clicking ”here”.

B.2 Input Data

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

%Domain lengths
%------------------------
domainPoints=[-1 1; 0 1];       %First  row for X dim
                                %Second row for Y dim


%Requested points (x: OUTLET)
%--------------------------
reqPoints=[0.1 0; 0.2 0; 0.3 0 ; 0.4 0;  0.5 0; 0.6 0; 0.7 0; 0.8 0; 0.9 0; 1 0]; ...
    %[x ; y] points

%Mesh sizes
%--------------------
meshSizes=[200 100];

%Initial properties
%---------------------
initProp=1;

%Boundary conditions
%----------------------
inletProp = [0 2];
outletProp = 0;
leftProp = 0;
rightProp = 0;
upperProp = 0;

%Time inputs
%---------------------
timeStep=5.0e2;
lastTime=1.0e6;
refTime=5.0e3;

%Material properties
%---------------------
rhogamma=1000000;
rho=1000000;
cp=4;
k=170;

%Iterative solver parameters
%-----------------------------
maxIter=1e4;
maxDiff=1e-4;


%Postprocessor Options
PostProcess = 1;    % 0 for no plots
```

B.3 Uniform Mesh Generation

```
%UNIFOR MESH & VELOCITY FIELD GENERATION
%-----------------------------------------------

classdef UniformMesh < handle
  properties (SetAccess=private)
    nodeX, nodeY, faceX, faceY, domain, U, V, Uf, Vf
  end
  methods

    function obj = UniformMesh(domainPoints,meshSizes)

      [domainLengths] = DomainLength(domainPoints);

      dim=1;
      [obj.nodeX,obj.faceX]=facesZVB(domainLengths(dim),...
          meshSizes(dim),domainPoints([1],[1]));

      dim=2;
      [obj.nodeY,obj.faceY]=facesZVB(domainLengths(dim),...
          meshSizes(dim),domainPoints([2],[1]));

            U   = zeros(numel(obj.nodeX),numel(obj.nodeY));
            V   = zeros(numel(obj.nodeX),numel(obj.nodeY));
            Uf  = zeros(numel(obj.faceX),numel(obj.faceY));
            Vf  = zeros(numel(obj.faceX),numel(obj.faceY));

            for indPX=1:numel(obj.nodeX)
                for indPY=1:numel(obj.nodeY)

                    x = obj.nodeX(indPX);
                    y = obj.nodeY(indPY);

                    obj.U(indPX,indPY) =  2*y*(1-x^2);
                    obj.V(indPX,indPY) =  2*x*(1-y^2);


                end
            end

            for indPX=2:(numel(obj.faceX))
                for indPY=1:(numel(obj.faceY))

                    xf = obj.faceX(indPX);
                    yf = obj.faceY(indPY);

                    obj.Uf(indPX,indPY) =  2*yf*(1-xf^2);
                    obj.Vf(indPX,indPY) =  -2*xf*(1-yf^2);

                end
            end

            %No slip boundary Condition

            obj.Uf(2:end,end)=0;    %TopBoundary
```

```
            obj.Vf(2:end,end)=0;
            obj.Uf(1,2:end)=0;      %LeftBoundary
            obj.Vf(1,2:end)=0;
            obj.Uf(end,2:end)=0;    %RightBoundary
            obj.Vf(end,2:end)=0;

    end



    function [s]=surfX(obj)
      s=obj.faceX(2:end)-obj.faceX(1:end-1);
    end
    function [s]=surfY(obj)
      s=obj.faceY(2:end)-obj.faceY(1:end-1);
    end
  end
end

%Domain Length
function [domainLengths] = DomainLength (domainPoints)

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

end

%facesZeroVolumeBoundaries
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 Physical Properties

```
% PHYSICAL PROPERTIES DOMAIN FILLING
%------------------------------------

classdef PhysProp < handle
  properties (SetAccess=private)
    rhogamma, cp, k, rho
  end
  methods
    function obj=PhysProp(mesh,rhogamma,cp,k,rho)

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

      obj.rhogamma=zeros(sizeX,sizeY);
```

```
      obj.cp=zeros(sizeX,sizeY);
      obj.k=zeros(sizeX,sizeY);
      obj.rho = zeros(sizeX,sizeY);

      %for one material

      obj.rhogamma(:,:)=rhogamma;
      obj.rho(:,:)=rho;
      obj.cp(:,:)=cp;
      obj.k(:,:)=k;

    end
  end
end
```

B.5 Boundary Conditions

```
%BOUNDARY CONDITIONS for defined PROPERTY
%-----------------------------------------
classdef BoundCond < handle
    properties (SetAccess = private)
        inletProp, outletProp, leftProp, rightProp, upperProp
    end

    methods
        function obj = BoundCond(inletProp, outletProp, leftProp, rightProp, ...
            upperProp)
            obj.inletProp = inletProp;
            obj.outletProp= outletProp;
            obj.leftProp  = leftProp;
            obj.rightProp = rightProp;
            obj.upperProp = upperProp;
        end
    end
end
```

B.6 Equation Coefficients Compute

```
classdef Coefficients < handle
  properties (SetAccess=private)
    ap, ae, aw, an, as, ap0, b, Fe
  end
  methods
    function obj = Coefficients(mesh)
      obj.ap=zeros(numel(mesh.nodeX),numel(mesh.nodeY));
      obj.ap0=zeros(size(obj.ap));
      obj.ae=zeros(size(obj.ap));
      obj.aw=zeros(size(obj.ap));
      obj.an=zeros(size(obj.ap));
```

```
      obj.as=zeros(size(obj.ap));
      obj.b=zeros(size(obj.ap));
      obj.Fe=zeros(size(obj.ap));

    end


    %INNER MATRIX COEFFICIENTS
    %----------------------------
    function innerAfor(obj,physProp,mesh,timeStep,Prop)

      sizeX=size(obj.ap,1);
      sizeY=size(obj.ap,2);

      for indPX=2:sizeX-1
        for indPY=2:sizeY-1

          Se= (mesh.faceY(indPY)-mesh.faceY(indPY-1));
          Sw= Se;
          Sn= (mesh.faceX(indPX)-mesh.faceX(indPX-1));
          Ss= Sn;

          obj.Fe(indPX,indPY) = Se*physProp.rho(indPX+1,indPY)*mesh.Uf(indPX,indPY);
          Fw = Sw*physProp.rho(indPX-1,indPY)*mesh.Uf(indPX - 1,indPY);
          Fn = Sn*physProp.rho(indPX,indPY+1)*mesh.Vf(indPX,indPY);
          Fs = Ss*physProp.rho(indPX,indPY-1)*mesh.Vf(indPX,indPY - 1);


          De = ((physProp.rho(indPX+1,indPY)/physProp.rhogamma(indPX,indPY))*Se)/...
               (mesh.nodeX(indPX+1) - mesh.nodeX(indPX));
          Dw = ((physProp.rho(indPX-1,indPY)/physProp.rhogamma(indPX,indPY))*Sw)/...
               (mesh.nodeX(indPX) - mesh.nodeX(indPX-1));
          Dn=  ((physProp.rho(indPX,indPY+1)/physProp.rhogamma(indPX,indPY))*Sn)/...
               (mesh.nodeY(indPY+1) - mesh.nodeY(indPY));
          Ds=  ((physProp.rho(indPX,indPY-1)/physProp.rhogamma(indPX,indPY))*Ss)/...
               (mesh.nodeY(indPY) - mesh.nodeY(indPY-1));

          %Peclet number
          Pe = obj.Fe/De;
          Pw = Fw/Dw;
          Pn = Fn/Dn;
          Ps = Fs/Ds;

          %NUMERICAL SCHEME POWERLAW
          Ae =1;% max(0, (1-0.1*abs(Pe))^5);
          Aw =1;% max(0, (1-0.1*abs(Pw))^5);
          An =1;% max(0, (1-0.1*abs(Pn))^5);
          As =1;% max(0, (1-0.1*abs(Ps))^5);

          obj.ae(indPX,indPY)= De*Ae + max(-obj.Fe(indPX,indPY),0);
          obj.aw(indPX,indPY)= Dw*Aw + max(Fw,0);
          obj.an(indPX,indPY)= Dn*An + max(-Fn,0);
          obj.as(indPX,indPY)= Ds*As + max(Fs,0);

          obj.ap0(indPX,indPY)=Se*Sn*Prop.T(indPX,indPY)/timeStep;

          obj.ap(indPX,indPY) = obj.ap0(indPX,indPY)+obj.ae(indPX,indPY)+...
              obj.as(indPX,indPY)+obj.an(indPX,indPY)+obj.aw(indPX,indPY);
```

```

          %Same as function newInnerB
          obj.b(indPX,indPY) = obj.ap0(indPX,indPY)*Prop.T(indPX,indPY);



        end
      end
    end

    %INNER MATRIX TIME DEPENDENT COEFFICIENTS
    %--------------------------------------------
    %-Density = ct
    %-Velocity field = ct
    function innerAforTime(obj,mesh,timeStep,Prop)

        sizeX=size(obj.ap,1);
        sizeY=size(obj.ap,2);

         for indPX=2:sizeX-1
            for indPY=2:sizeY-1

                Se= (mesh.faceY(indPY)-mesh.faceY(indPY-1));
                Sw= Se;
                Sn= (mesh.faceX(indPX)-mesh.faceX(indPX-1));
                Ss= Sn;

                obj.ap0(indPX,indPY)=(Se*Sn*Prop.T(indPX,indPY))/timeStep;

                obj.ap(indPX,indPY) = obj.ap0(indPX,indPY)+obj.ae(indPX,indPY)+...
                    obj.as(indPX,indPY)+obj.an(indPX,indPY)+obj.aw(indPX,indPY);

                obj.b(indPX,indPY) = obj.ap0(indPX,indPY)*Prop.T(indPX,indPY);


            end

            if mesh.nodeX(indPX) > 0
               Prop.T(indPX , 1) =  Prop.T(indPX,2);
            end
            if indPX==sizeX-1
                Prop.T(indPX+1 , 1) =  Prop.T(indPX+1,2);
            end

         end

    end

    function newInnerB(obj,Prop)
      obj.b(2:end-1,2:end-1)=obj.ap0(2:end-1,2:end-1).*Prop.T(2:end-1,2:end-1);
    end


    %BOUNDARY COEFFICIENTS
    %-------------------------------------------------
    function topBoundary(obj,upperProp,Prop)

      Prop.T(2:end-1,end) = upperProp;
```

```

    end

    function bottomBoundary(obj,outletProp,inletProp,Prop,mesh)

        sizeX=size(obj.ap,1);

        for indPX=1:sizeX

            if mesh.nodeX(indPX) < −0.5 && mesh.nodeX(indPX) ≥ −1
                Prop.T(indPX , 1) =  inletProp(1);
            end
            if mesh.nodeX(indPX) > −0.5 && mesh.nodeX(indPX) ≤ 0
                Prop.T(indPX , 1) = inletProp(2);
            end
            if(outletProp==0)
                if mesh.nodeX(indPX) > 0
                   Prop.T(indPX , 1) =  Prop.T(indPX,2);
                end
            end
        end

    end

    function leftBoundary(obj,leftProp,Prop)

      Prop.T(1,2:end) = leftProp;

    end

    function rightBoundary(obj, rightProp, Prop)
      Prop.T(end,2:end) = rightProp;
    end

  end




end
```

B.7 Solver Function

```
% ITERATIVE SOLVER METHODS
%-----------------------------------------

classdef Solver < handle

    properties (SetAccess = private)

    end

    methods

```

```

        function obj=Solver(coef, Prop)



            %POINT-BY-POINT SOLVER
            %--------------------------------------------------
            % - Option 1
            sizeX=size(coef.ap,1);
            sizeY=size(coef.ap,2);


            for indPX=2:sizeX-1
                for indPY=2:sizeY-1
                    Prop.T(indPX,indPY) = ...
                        (coef.ae(indPX,indPY)*Prop.T0(indPX+1,indPY)+ ...
                                          coef.aw(indPX,indPY)*Prop.T0(indPX-1,indPY)+ ...
                                              ...
                                          coef.an(indPX,indPY)*Prop.T0(indPX,indPY+1)+ ...
                                              ...
                                          coef.as(indPX,indPY)*Prop.T0(indPX,indPY-1)+ ...
                                              ...
                                          coef.b(indPX,indPY))/(coef.ap(indPX,indPY)) ...
                                                ;
                end
            end

            % - Option 2:
%             Prop.T(2:sizeX-1,2:sizeY-1) = ...
    (coef.ae(2:sizeX-1,2:sizeY-1).*Prop.T0((2:sizeX-1)+1,2:sizeY-1)+ ...
%                                                   ...
    coef.aw(2:sizeX-1,2:sizeY-1).*Prop.T0((2:sizeX-1)-1,(2:sizeY-1))+ ...
%                                                   ...
    coef.an(2:sizeX-1,2:sizeY-1).*Prop.T0((2:sizeX-1),(2:sizeY-1)+1)+ ...
%                                                   ...
    coef.as(2:sizeX-1,2:sizeY-1).*Prop.T0((2:sizeX-1),(2:sizeY-1)-1)+ ...
%                                                   ...
    coef.b(2:sizeX-1,2:sizeY-1))./(coef.ap(2:sizeX-1,2:sizeY-1))   ;

             %LINE-BY-LINE SOLVER
             %----------------------------------------------



        end



    end

end
```

B.8 Field ϕ Properties

```
% PROPERTIES TO BE SAVED FOR THE POST PROCESSINGe
%———————————————————————————————————————————————————

classdef Properties < handle
    properties  (SetAccess = public)

        T, T0

    end

    methods
        function obj = Properties(mesh, initProp)

            obj.T = zeros(numel(mesh.nodeX),numel(mesh.nodeY))+initProp;
            obj.T0 = zeros(numel(mesh.nodeX),numel(mesh.nodeY))+initProp;

        end
    end
end
```

B.9 Core of the code

```
%TRANSIENT 2—D CONVECTION—DIFFUSSION EQUATION
%——————————————————————————————————————————————————

classdef TransientConvectionDiffusion2D < handle

    properties (SetAccess=public)
        mesh, physProp,boundCond ,timeStep, refTime , coef, Prop,Pref, err
    end

    %Prop = property to compute

    methods

        function  obj = TransientConvectionDiffusion2D(mesh, physProp, boundCond, ...
            timeStep, initProp, refTime)
            obj.mesh=mesh;
            obj.physProp=physProp;
            obj.boundCond=boundCond;
            obj.timeStep=timeStep;
            obj.refTime = refTime;

            obj.Prop = Properties(mesh, initProp);
            obj.coef = Coefficients(obj.mesh);


        end


```

```
        %Main algorithm
        function [PropReqPoints,timeReqPoints]=solveTime(obj,lastTime, reqPoints, ...
            maxIter, maxDiff,PostProcess)



            %CONSTANT COEFFICIENTS
            %------------------------------------------------------------------------
            %Inner matrix
            obj.coef.innerAfor(obj.physProp,obj.mesh,obj.timeStep, obj.Prop);


            %Boundaries
            obj.coef.topBoundary(obj.boundCond.upperProp,obj.Prop);
            obj.coef.leftBoundary(obj.boundCond.leftProp, obj.Prop);
            obj.coef.bottomBoundary(obj.boundCond.outletProp,...
                                    obj.boundCond.inletProp,obj.Prop,obj.mesh);
            obj.coef.rightBoundary(obj.boundCond.rightProp,obj.Prop);

            time=0.0;

            %REQUESTED POINTS RESULTS
            %-----------------
            numTotalValues=min(lastTime/max(1,obj.timeStep),1e4)+1;

            timeReqPoints=zeros(1,numTotalValues);
            PropReqPoints=zeros(numTotalValues,size(reqPoints,1));

            %Matlab Interpolation Function FOR reqPoints
            PropReqPoints(1,:)=interp2(obj.mesh.nodeX,obj.mesh.nodeY,...
            obj.Prop.T',reqPoints(:,1),reqPoints(:,2))';

            %MAIN ALGORITHM
            %-------------------------------------------------------------------------
            obj.Prop.T0 = obj.Prop.T;

            cnt=1;
            t1=toc;
            fprintf(' Time: %f\nsolveTime: :Iterate: \n',t1);

            %CORE OF THE CODE
            %----------------------------------------------------------------
            tic;t1=toc;
            saveRefTime=true;

            while time<lastTime

                % TIMING OF ITERATIONS FOR LAST TIME STEP
                %-----------------
                time=time+obj.timeStep;
%                 if mod(round(time),showStep)==0 &&...
%                     abs(time-round(time))<0.5*obj.timeStep
%
%                     t2=toc;
%                     timePerIte=(t2-t1)/showStep;
%                     fprintf('Current Time: %6.f TpI: %.3es ETC: %5.fs\n',...
%                     time,timePerIte,timePerIte*(lastTime-time));
%                     t1=t2;
```

```
%                   end

                %DOMAIN CONVERGENCE
                %------------------
                it = 0;
                obj.err = zeros (size(obj.coef.ap,1)-1,size(obj.coef.ap,2)-1)+1;
                a=max(obj.err);
                stop=0;

                while (max(a) > maxDiff)  || stop == 1

                    it = it +1;

                    %SOLVER
                    %------------------
                    obj.Prop.T0 = obj.Prop.T;
                    Solver(obj.coef, obj.Prop);

                    %COEFFICIENTS
                    %-------------------
                    %Inner matrix
                    obj.coef.innerAforTime(obj.mesh,obj.timeStep,obj.Prop);


                    % CONVERGENCE CHECK
                    %------------------
                    obj.err = abs(obj.Prop.T0-obj.Prop.T);
                    a=max(obj.err);

                    if it > maxIter
                        error("Can not reach convergence of the results, check ...
                            Input Data")
                        stop=1;
                    end
                end

                %Ensures time steps count and saves times
                if (time+obj.timeStep)>cnt
                  cnt=cnt+1;
                  timeReqPoints(cnt)=time;


                  PropReqPoints(cnt,:)=interp2(obj.mesh.nodeX,obj.mesh.nodeY,...
                    obj.Prop.T',reqPoints(:,1),reqPoints(:,2))';
                end
                %Saves temperature at reference time
                if saveRefTime && (time+obj.timeStep)>obj.refTime
                    fprintf('Saving refTemp: %f\n',time);
                    obj.Pref=obj.Prop.T;
                    saveRefTime=false;
                end

            end

            if PostProcess ==1

                postprocess(obj.Prop, obj.mesh);

```

```

                %---------- 3-D FIELD PLOT ----------%
                o = rot90(obj.Prop.T,-1);
                O=fliplr(o);
                figure(5)
                mesh(obj.mesh.nodeX,rot90(obj.mesh.nodeY,-2),rot90(o,-2));
                colormap(jet);
                title('Field \phi of the Smith-Hutton Problem');
                xlabel('Domain size (x)'), ylabel('Domain size (y)');
                zlabel('Field \phi value');

                % --------- INLET AND OUTLET FIELD PLOT -----------%
                sizeX=size(obj.coef.ap,1);

                for indPX=2:sizeX
                    vals(indPX-1)=obj.Prop.T(indPX,2);
                end

                x=linspace(-1,1,sizeX-1);

                figure(6)
                y= zeros(sizeX-1)+  max(vals);
                p=plot(x,vals, x,y,'--k','LineWidth',1);
                title('Field \phi of the Smith-Hutton Problem at Bottom Boundary');
                xlabel('Domain size (x)'), ylabel('Field \phi Value');

                ylim([min(vals) max(vals)+1]);
                p(1).LineWidth = 2;

            end
        end

    end



end

%-----------------------------------------------------%
%----------------POSTPROCESSOR FUNCTION---------------%
%-----------------------------------------------------%
function postprocess(Prop, mesh)


    o = rot90(Prop.T,-1);
    O=fliplr(o);

    % -------------ISOTHERM MAP --------------%
    figure(1);
    contour(O);
    colormap(jet);
    colorbar;
    title('Field \phi isobars of the Smith-Hutton Problem');
    xlabel('x'), ylabel('y');

    %---------ISOTHERM COLOR MAP ---------%
    figure(2);
    contourf(mesh.nodeX,mesh.nodeY,O);     %mostra les isotermes
```

```
    colormap(jet);
    colorbar;
    title('Field \phi isobars of the Smith—Hutton Problem');
    xlabel('x'), ylabel('y');

    % ——————————————COLOR MAP ———————————————%
    figure(3);
    pcolor(mesh.nodeX,mesh.nodeY,O);
    shading interp;
    colormap(jet);
    colorbar;
    title('Field \phi of the Smith—Hutton Problem');
    xlabel('x'), ylabel('y');



    %———————————  MESH PLOT ————————————————%
    figure(4);
    h = heatmap(rot90(o,—2));  %heatmap
    colormap(jet);
    title('Field \phi of the Smith—Hutton Problem');
    xlabel('Nodes in x direction'), ylabel('Nodes in y direction');









end
```

Appendix C

Smith-Hutton Problem Code

In this Appendix we can see the code developed with Matlab OOP for solving the Smith-Hutton Problem. This results has been obtained using a self developed code[1] with the numerical scheme and solver indicated in the case of study. This code was made in the purpose of studding the transitory state and computational resolution of the Smith-Hutton problem. This code works with the same files of the previous one but some have been modified. The ones modified are shown in this attachments.

C.1 Main Algorithm

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%  SMITH—HUTTON  %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%  Miquel Altadill Llasat  %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear all
more off
tic;
%Load input data
InputData

tic;
mesh=UniformMesh(domainPoints,meshSizes);
fprintf('MeshTime %f\n',toc); tic;

physProp=PhysProp(mesh,rhogamma,cp,k,rho);
fprintf('PhysPropTime %f\n',toc); tic;

boundCond=BoundCond(inletProp, outletProp, leftProp, rightProp, upperProp);
fprintf('BoundCondTime %f\n',toc); tic;

tcd2D=TransientConvectionDiffusion2D(mesh, physProp, boundCond, timeStep, ...
    initProp, refTime);
```

[1] This code can be downloaded with a document that describes the case of study by clicking ”here”.

```
fprintf('CreateTHC2DTime %f\n',toc); tic;

tcd2D.solveTime( maxIter, maxDiff,PostProcess,maxtDiff, reqPoints);

t=toc;
```

C.2 Input Data

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

%Domain lengths
%------------------------
domainPoints=[-1 1; 0 1];        %First  row for X dim
                                 %Second row for Y dim


%Requested points (x: OUTLET)
%--------------------------
reqPoints=[0.1 0; 0.2 0; 0.3 0 ; 0.4 0;  0.5 0; 0.6 0; 0.7 0; 0.8 0; 0.9 0; 1 0]; ...
    %[x ; y] points

%Mesh sizes
%--------------------
meshSizes=[30 15];

%Initial properties
%---------------------
initProp=1;

%Boundary conditions
%----------------------
inletProp = [0 2];
outletProp = 0;
leftProp = 0;
rightProp = 0;
upperProp = 0;

%Time inputs
%---------------------
timeStep=8;
refTime=5.0e3;
maxtDiff=1e-4;

%Material properties
%---------------------
rhogamma=1000;
rho=1000;
cp=4;
k=170;

%Iterative solver parameters
%-----------------------------
maxIter=1e4;
```

```
maxDiff=1e-4;


%Postprocessor Options
PostProcess = 11;     % 0 for no plots
                      % 11 for evolutive plot
```

C.3 Field ϕ Properties

```
% PROPERTIES TO BE SAVED FOR THE POST PROCESSING AND SOLVING
%-----------------------------------------------------------
%INPUTS
%    - initProp:     \phi value at t=0
%    - UniformMesh (OBJECT)
%        - mesh.nodeX
%        - mesh.NodeY
%
%






classdef Properties < handle
    properties  (SetAccess = public)

        T, T0, Tt

    end

    methods
        function obj = Properties(mesh, initProp)

            obj.T = zeros(numel(mesh.nodeX),numel(mesh.nodeY))+initProp;
            obj.T0 = zeros(numel(mesh.nodeX),numel(mesh.nodeY))+initProp;
            obj.Tt = zeros(numel(mesh.nodeX),numel(mesh.nodeY))+initProp;
        end
    end
end
```

C.4 Core of the code

```
%TRANSIENT 2-D CONVECTION-DIFFUSSION EQUATION
%-------------------------------------------------

classdef TransientConvectionDiffusion2D < handle

    properties (SetAccess=public)
```

```
        mesh, physProp,boundCond ,timeStep, refTime , coef, Prop,Pref, err, ...
            tempReqPoints
    end

    %Prop = property to compute

    methods

        function  obj = TransientConvectionDiffusion2D(mesh, physProp, boundCond, ...
            timeStep, initProp, refTime)
            obj.mesh=mesh;
            obj.physProp=physProp;
            obj.boundCond=boundCond;
            obj.timeStep=timeStep;
            obj.refTime = refTime;

            obj.Prop = Properties(mesh, initProp);
            obj.coef = Coefficients(obj.mesh);

            obj.tempReqPoints = zeros(1000,10);


        end


        %Main algorithm
        function solveTime(obj, maxIter, maxDiff,PostProcess,maxtDiff, reqPoints)



            %CONSTANT COEFFICIENTS
            %-----------------------------------------------------------------------
            %Inner matrix
            obj.coef.innerAfor(obj.physProp,obj.mesh,obj.timeStep, obj.Prop);


            %Boundaries
            obj.coef.topBoundary(obj.boundCond.upperProp,obj.Prop);
            obj.coef.leftBoundary(obj.boundCond.leftProp, obj.Prop);
            obj.coef.bottomBoundary(obj.boundCond.outletProp,...
                                    obj.boundCond.inletProp,obj.Prop,obj.mesh);
            obj.coef.rightBoundary(obj.boundCond.rightProp,obj.Prop);



            %MAIN ALGORITHM
            %------------------------------------------------------------------------
            obj.Prop.T0 = obj.Prop.T;
            obj.Prop.Tt = obj.Prop.T;
            Time = zeros;               %Iteration time
            time = zeros;                   %Real time
            Error = zeros;

            obj.err = zeros (size(obj.coef.ap,1)-1,size(obj.coef.ap,2)-1)+1;
            tit = 0;                     %Time iterations

            %CORE OF THE CODE
            %---------------------------------------------------------------
```

```
        diff2=inf;
        tic;

        while diff2 > maxtDiff

            tit = tit +1;          %Time iteration count
            time(tit) = tit*obj.timeStep;

            %INNER COEFFICIENTS
            %-------------------
            obj.coef.innerAforTime(obj.mesh,obj.timeStep,obj.Prop);

            %DOMAIN CONVERGENCE
            %------------------
            it = 0;
            diff1=inf;
            stop=0;

            while (diff1 > maxDiff)  || stop == 1

                it = it +1;

                %SOLVER
                %------------------
                Solver(obj.coef, obj.Prop);

                % CONVERGENCE CHECK
                %------------------
                obj.err = abs(obj.Prop.T0-obj.Prop.T);
                a = max(obj.err);
                diff1 = max(a);

                obj.Prop.T0 = obj.Prop.T;

                if it > maxIter
                    error("Can not reach convergence of the results, check ...
                        Input Data")
                    stop=1;
                end
            end

            d2 = abs(obj.Prop.Tt-obj.Prop.T);
            d2i = max(d2);
            diff2 = max(d2i);%/obj.timeStep;
            obj.Prop.Tt=obj.Prop.T;
            Time(tit) = toc;
            fprintf('Time= %d; Ctime = %d;  Error = %d;  Iterations = %d; ...
                TimeStep = %d\n',...
                        time(tit),Time(tit),diff2,it,tit);

            Error(tit) = diff2;

            %obj.tempReqPoints(tit,:)  = interp2(obj.mesh.nodeX,obj.mesh.nodeY,...
            %obj.Prop.T,reqPoints(:,1),reqPoints(:,2));

            %tempReqPoints(cnt,:)=interp2(obj.mesh.nodeX,obj.mesh.nodeY,...
        %obj.T',reqPoints(:,1),reqPoints(:,2))';

```

```

                %Postproces Evolutive Plot
                if PostProcess == 11
                    o = rot90(obj.Prop.T,-1);
                    O=fliplr(o);
                    figure(3);
                    pcolor(obj.mesh.nodeX,obj.mesh.nodeY,O);
                    shading interp;
                    colormap(jet);
                    colorbar;
                    title('Field \phi of the Smith-Hutton Problem');
                    xlabel('x'), ylabel('y');
                end



            end

            if PostProcess ==1

                postprocess(obj.Prop, obj.mesh);


                %---------- 3-D FIELD PLOT ---------%
                o = rot90(obj.Prop.T,-1);
                O=fliplr(o);
                figure(5)
                mesh(obj.mesh.nodeX,rot90(obj.mesh.nodeY,-2),rot90(o,-2));
                colormap(jet);
                title('Field \phi of the Smith-Hutton Problem');
                xlabel('Domain size (x)'), ylabel('Domain size (y)');
                zlabel('Field \phi value');

                % -------- INLET AND OUTLET FIELD PLOT ----------%
                sizeX=size(obj.coef.ap,1);

                for indPX=2:sizeX
                    vals(indPX-1)=obj.Prop.T(indPX,2);
                end

                x=linspace(-1,1,sizeX-1);

                figure(6)
                y= zeros(sizeX-1)+  max(vals);
                p=plot(x,vals, x,y,'--k','LineWidth',1);
                title('Field \phi of the Smith-Hutton Problem at Bottom Boundary');
                xlabel('Domain size (x)'), ylabel('Field \phi Value');

                ylim([min(vals) max(vals)+1]);
                p(1).LineWidth = 2;

            end
        end

    end



```

```
end

%------------------------------------------------------------------%
%--------------------POSTPROCESSOR FUNCTION------------------------%
%------------------------------------------------------------------%
function postprocess(Prop, mesh)


    o = rot90(Prop.T,-1);
    O=fliplr(o);

    % ----------------ISOTHERM MAP -----------------%
    figure(1);
    contour(O);
    colormap(jet);
    colorbar;
    title('Field \phi isobars of the Smith-Hutton Problem');
    xlabel('x'), ylabel('y');

    %----------ISOTHERM COLOR MAP ------------%
    figure(2);
    contourf(mesh.nodeX,mesh.nodeY,O);     %mostra les isotermes
    colormap(jet);
    colorbar;
    title('Field \phi isobars of the Smith-Hutton Problem');
    xlabel('x'), ylabel('y');

    % -----------------COLOR MAP ------------------%
    figure(3);
    pcolor(mesh.nodeX,mesh.nodeY,O);
    shading interp;
    colormap(jet);
    colorbar;
    title('Field \phi of the Smith-Hutton Problem');
    xlabel('x'), ylabel('y');



    %------------   MESH PLOT -------------------%
    figure(4);
    h = heatmap(rot90(o,-2));  %heatmap
    colormap(jet);
    title('Field \phi of the Smith-Hutton Problem');
    xlabel('Nodes in x direction'), ylabel('Nodes in y direction');








 end
```