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

UNIVERSITAT POLITÈCNICA DE CATALUNYA

BACHELOR'S DEGREE IN AEROSPACE TECHONOLOGIES ENGINEERING

BACHELOR'S THESIS - ANNEX II : CONVECTION-DIFFUSION EQUATION

Computational studies of non-viscous and viscous fluid flows

Author:
PÉREZ RICARDO, Carlos

Director:
OLIVA LLENA, Asensio

Co-director:
PÉREZ SEGARRA, Carles-David

June 10^{th}, 2019

Contents

List of Figures

1 Lineal Parallel flow in a duct

1.1 Results

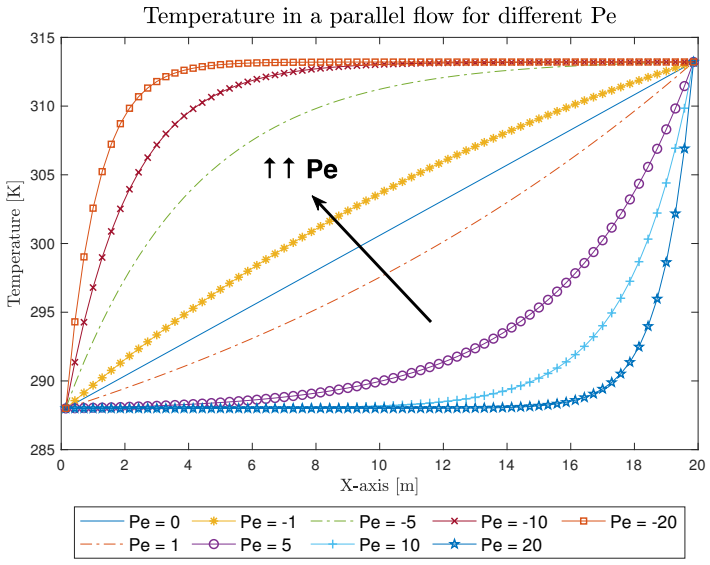

Figure 1.1: Temperature along X-axis for different Péclet numbers

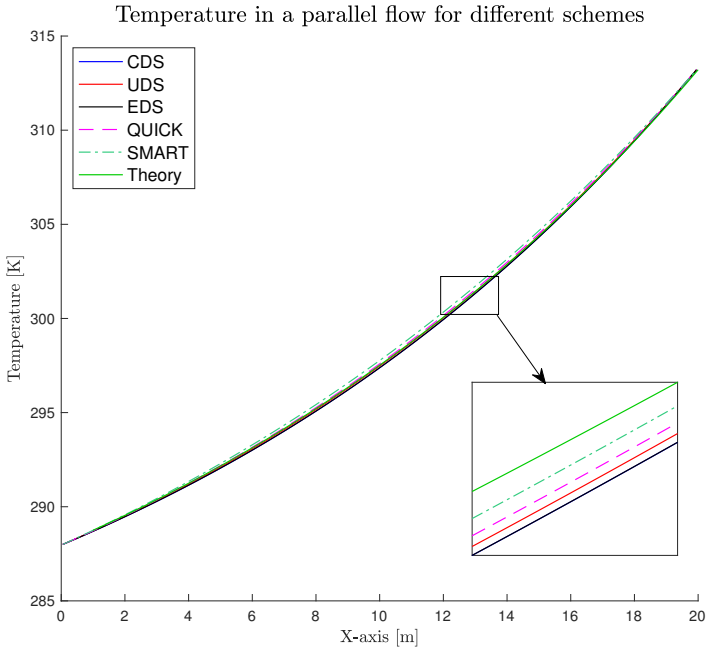

Figure 1.2: CDS , UDS, EDS, QUICK & SMART solutions for Pe = 1

1.2 Code in C++: $CD_$ParallelFlow.cpp

```
#include <iostream>
# include <stdlib.h>
# include <fstream>
# include <math.h>
#include <iomanip>
#include <cstdlib>

//# include <windows.h>

using namespace std;

// CONSTANTES FÍSICAS

const float pi = 3.14159265;

// DATOS DEL PROBLEMA

const int Nx = 200; // number of divisions along dir. X
const float L = 20; // length of the canal in [m]
const float H = 20; // height of the canal in [m]
const float W = 5;  // width of the canal in [m]
const float T_step = 0.0001;
const int dim = 1;
//const float T_step = 0; // Estacionario


const float Mass_0 = 5; // Initial mass flux
const float rho_0 = 0.003; // Initial mass flux
const float Temp_0 = 273+15; // Temp [K]
const float VelX_0 = 50; // entrance's velocity  [m]
const float VelY_0 = 15;
const float mu = 1.2*pow(10,5);
const float Cv = 2.050;
const float lambda = 0.58;
const float u_inf = 50;
const int divT = 2;
float temps [divT+2];


// VARIABLES

    float COOR[Nx+1];
```

```
    float X[Nx+1];
    float Y[Nx+1];
    float V[Nx+1];
    float S[Nx+1][2*dim];
    const float deltaX = L/Nx;
    //const float deltaY = H/Ny;
    float MU [2*dim];

    float Temp [Nx+1][divT+1];
    float Mass [Nx+1][divT+1];
    float rho [Nx+1][divT+1];
    float VelX [Nx+1][divT+1];
    double VelX_teo [Nx+1];
    float CondContornVel [Nx+1];
    float CondContornTemp [Nx+1];

    //float ae [Nx+1][Ny+1];
    //float aw [Nx+1][Ny+1];
    //float an [Nx+1][Ny+1];
    //float as [Nx+1][Ny+1];
    //float ap [Nx+1][Ny+1];
    //float bp [Nx+1][Ny+1];

    // TDMA line by line
    float bp_ast [Nx+1];
    float P[Nx+1];
    float R [Nx+1];

    float criteriConv = pow(10,-7);

    //int north1 = 0; int south1 = 0;
    //int north2 = 0; int south2 = 0;
    int west1 = 0; int west2 = 0;
    int east1 = 0; int east2 = 0;
    int point = 0;

// FUNCTIONS' MENTION

void Geometria2D (float L, float H, int Nx, int Ny);
void IdentifyNodes (int I, int J, int nx, int ny);

void Centroide (float COOR[Nx+1], int Nx, int Ny);
float DegtoRad (float angle);
```

```
float Circulation (float vye, float vxn, float vyw,
float vxs, float deltaY, float deltaX);
float Harmonicmean (float dPE, float dPe, float dEe,
float phiP, float phiE) ;
float phi_UDS (float mflow, float phiP, float phiX);
float phi_HRS (int i, int j, int next1, int next2,
int prev1, float phi[Nx+1], int tipo, float coor[Nx]);
    float xbar (float x, float xu, float xd);
    float phibar (float phi, float phiu, float phid) ;
float GaussSeidel2D (float a_w, float a_e, float a_n,
float a_s, float a_p, float b_p, float phiW, float phiE,
float phiN, float phiS) ;

void MassEquation (int nx, int ny, float T_step);

void ConvDiff (float Lambda[2*dim+1], float S[Nx+1][2*dim],
float phi[Nx+1][divT+1], int tipo, float t_step) ;

void pasartablaafichero1D (float valor [Nx+1][divT+1],
int Nx, int tipo ) ;

// FUNCTIONS

void Geometria2D (float l, float h, int nx, int ny) {
    int i = 0; int j = 0;

         for (i=0; i<=nx; i++) {
            COOR [i] = (i)*deltaX;
            //rho  [(nx+1)*j+i] = rho0;
        }

    Centroide (COOR, nx, ny);
}


void Centroide (float  COOR[Nx+1], int nx, int ny) {

    int i=0; int j=0;

         for (i=0; i<nx; i++) {
            X [i] = (COOR[i]+COOR[i+1])/2;

            V[i] = deltaX*H*W;
            S[i][0] = H*W; // Cara N
```

```
            S[i][1] = H*W; // Cara S
         }

        cout << "1. Malla calculada" << endl;

    /*
    cout << "---------- Coordenadas centroides 2 ----------" << endl;
    for (j=0; j<ny; j++) {
         for (i=0; i<nx; i++) {
            cout << "Punto " << i << " " << j;
            cout << " = Comp X: " << X[i];
            cout << " // Comp Y: " << Y[j] << endl;
            cout << "Sx: " << S[i][j][2] << " Sy: " <<
            S[i][j][0] << " V: " << V[i][j] << endl;
    }
    }
    cout << "------ End of Coordenadas centroides 2 ------" << endl;
    */

}



void IdentifyNodes (int I, int J, int nx, int ny) {

    west1 = I-1;
        west2 = I-2;
    east1 = I+1;
        east2 = I+2;

    if ( I == 0) {
        west1 = -1;
    }
    if ( I == nx-1) {
        east1 = -1;
    }

    if ( I == 0) {
        west2 = -1;
    }
    if ( I == nx-2 ||  I == nx-1) {
        east2 = -1;
    }
```

```
    ////cout << "P: "<<I<<" "<<J<< " / N: " << north << " /
    S: " << south << " / W: " << west << " / E: " << east << endl;
    //cout << "P: "<<I<<" "<<J<< " // N: " << north1 << " /
    NN: "<< north2 << " // S: " << south1 <<" /SS: " << south2;
    //cout << " // W: " << west1 << " / WW: " << west2 << " //
    E: " << east1 << " / EE: " << east2 << endl;
}

void CondInicials (int nx, int ny, float Mass_0, float VelX_0,
float VelY_0, float Temp_0) {

    float xc = L / 2;
    float yc = 0;

    // phi = Mass, Velocity and Temperature
    int i = 0;
    int j = 0;

         for (i=0; i<=nx; i++) {
            Mass[i][0]=Mass_0;
            rho[i][0] = rho_0;
            Temp[i][0] = Temp_0;
            //VelX[i][0] = VelX_0*(1-i*0.001);
            VelX[i][0] = VelX_0/2;

                CondContornVel[i] = 0;
                CondContornTemp[i] = 0;

            if (i==0){
                CondContornVel[i] = 1;
                CondContornTemp[i] = 1;
                VelX[i][0] = VelX_0;
            }

            if (i==Nx-1) {
                CondContornVel[i] = 1;
                CondContornTemp[i] = 1;
                VelX[i][0] = VelX_0 - 5.93;
                Temp[i][0] = Temp_0 - 18.21;
            }

        }

    cout << "2. Condiciones Iniciales establecidas" << endl;
```

```
}

float Harmonicmean (float dPE, float dPe, float dEe,
float phiP, float phiE) {
    float phi = dPE / (dPe/phiP + dEe/phiE);
    return phi ;
}

float GaussSeidel (float a_w, float a_e, float a_p, float b_p,
float phiW, float phiE)    {
    float phi;

    phi = (a_w*phiW + a_e*phiE + b_p) / a_p;

    return phi;
}


void ConvDiff (float Lambda[2*dim+1], float S[Nx+1][2*dim],
float phi[Nx+1][divT+1], int tipo, float t_step, float CondContorn[Nx+1]) {

    int i =0; int j =0;

    float mflowW = 0; float mflowE = 0;

    float speedX_e = 0; float speedX_w = 0;

    float rho_e = 0; float rho_w = 0;
    float rhoM = 0;

    float Fe = 0; float Fw = 0;

    float dPEx = 0; float dPex = 0; float dEex = 0;

    float new_phi = 0;

    float Dw = 0; float De = 0;

    float phi_UDS_e = 0; float phi_UDS_w = 0;
    float phi_HRS_e = 0; float phi_HRS_w = 0;

    int bucles = 0;
```

```
    // Implicit beta = 1;
int p = 0; int x = 0;

float aw = 0;
float ae = 0;
float ap_0 = 0;
float ap = 0;
float bp = 0;
    float bp_exp = 0;
    float qp = 0;

float TIME = 0;

float no=0;

int loops = 800;

for (p=0; p<divT+1; p++) {

    temps[p] = TIME;
    //cout << TIME << endl;

        // SUPOSEM T en l'instant n+1
            for (i=0; i<Nx; i++) {
                phi[i][p+1]= phi[i][p];
            }

        // PEL PRIMER INSTANT DE TEMPS
        if (p==0){

                for (i=0; i<=Nx; i++) {
                    //T[i][p]=T_0;
                    phi[i][p+1]= phi[i][p];
                    //cout << "Paso inicial"<< endl;
                    //cout << p << " " << i << " : "<< phi[i][j][p+1] << endl;
                    //cout << TIME;
                }

            TIME = TIME + t_step;

        } else {

            x = 0;
            //cout << "Estoy aqui" << endl;
```

```
while (x < loops) {

        for (i=0; i<=Nx; i++) {

            IdentifyNodes(i,j,Nx,Nx);

            dPEx = deltaX;  dPex = deltaX/2;  dEex = deltaX/2;

            speedX_e = Harmonicmean(dPEx, dPex, dEex,
            VelX[i][p+1], VelX[east1][p+1]);
            speedX_w = Harmonicmean(dPEx, dPex, dEex,
            VelX[i][p+1], VelX[west1][p+1]);

            rho_e = Harmonicmean(dPEx, dPex, dEex,
            rho[i][p+1], rho[east1][p+1]);
            rho_w = Harmonicmean(dPEx, dPex, dEex,
            rho[i][p+1], rho[west1][p+1]);

            //rho_e = rho_0; rho_w = 0;

            Fe = rho_e*speedX_e;
            Fw = rho_w*speedX_w;

            De = Lambda[0] / deltaX; Dw = Lambda[1] / deltaX;

            if (tipo == 1){
                De = 0; Dw = 0;
            }
            if (tipo == 2) {
                De = Lambda[0] / deltaX; Dw = Lambda[1] / deltaX;
                //cout << De << "  " << Dw << endl;
            }

            // CENTRAL DIFFERENCE SCHEME

            ae = De - Fe;
            aw = Dw + Fw;

            // UPWIND SCHEME

            //ae = De + phi_UDS (-Fe, phi[i][p+1],  phi[east1][p+1]
            //aw = Dw + phi_UDS (Fw, phi[i][p+1],  phi[west1][p+1])
```

```
if (east1 == -1){
    ae = 0;
    Fe = rho[i][p+1]*VelX[i][p+1]; Fe = 0;
}


if (west1 == -1){
    aw = 0;
    Fw = rho[i][p+1]*VelX[i][p+1]; Fw = 0;
}

ap = ae + aw + (Fe - Fw);

    //cout << phi_UDS (-Fe, phi[i][p+1],
    phi[east1][p+1]) << "   " <<  phi_UDS (Fw, phi[i][p+1],
    phi[west1][p+1])<< endl;
    //cout << ae << " " << aw << " " << ap << endl;

bp = 0;

//cout << mflowS << endl;
//cout << mflowS << " " << mflowN << " " <<
mflowE  << " " << mflowW << " " << endl;

//T[i][p+1] = TDMA(aw,ae,ap,bp, T[i-1][p+1], T[i+1][p+1],i)
new_phi = GaussSeidel(aw,ae,ap,bp,
phi[west1][p+1], phi[east1][p+1]);

if (x==2 && tipo == 4){
    cout <<i<<" "<< j<< " : "<< ae << " " <<
    aw << " " << ap << endl;
    cout <<i<<" "<< j<< " : "<< Fe << " " <<
    Fw << " "<< De << " " <<Dw<< endl;
    cout << new_phi << endl;
}

if ( abs(new_phi - phi[i][p+1]) > criteriConv &&
CondContorn[i] == 0 ) {
    phi[i][p+1] = new_phi;
}
//cout << new_phi<<endl;
//if (x == loops-1 && tipo == 2){
    //cout << p << " " << i << " : "<<
    phi[i][p+1] << endl;
```

```
                        //}
                        //T[i][p+1]= T[i][p];

                    }

                x++;

            }

        }

        TIME = TIME + t_step;

    }

}

float phi_UDS (float mflow, float phiP, float phiX){

    int fe;
    float phi=0;

    if (mflow>0) {
        fe = 0;
    } else {
        fe = 1;
    }

    phi = fe * (phiX-phiP)+phiP;

    return phi;

}


float phibar (float phi, float phiu, float phid) {

    if (phid == phiu) {
        return 1;
    } else {
        float phibar = (phi-phiu)/(phid-phiu);
        return phibar;
    }
}
```

```
float xbar (float x, float xu, float xd) {

    if (xu == xd) {
        return 1;
    }else {
        float xbar = (x-xu)/(xd-xu);
        return xbar;
    }

}

float phi_HRS (int i, int j, int next1, int next2, int prev1,
float phi[Nx+1], int tipo, float coor[Nx+1]) {

    float xbarF = 0;
    float xbarC = 0;
    float phibarC = 0;
    float phibarF = 0;

    int nextX1 = 0; int nextX2 = 0; int prevX1 = 0;
    int nextY1 = 0; int nextY2 = 0; int prevY1 = 0;

    int I = 0;

    float phiu = 0; float phid = 0;
    float phiF = 0;

    if (tipo == 1){ // En direccio x , y no varia
        nextX1 = next1; nextX2 = next2; prevX1 = prev1;
        nextY1 = j; nextY2 = j; prevY1 = j;

        I = i;

    }

    if (tipo ==2){ // En direccio y , x no varia
        nextY1 = next1; nextY2 = next2; prevY1 = prev1;
        nextX1 = i; nextX2 = i; prevX1 = i;

        I = j;

    }
```

```
//cout << nextX1 << " " << nextX2 << " " << prevX1;
//cout << "  //  " << nextY1 << " " << " " << prevY1 << endl;

// SMART
if (next1 == -1){
    /*xbarF = xbar( coor[I], 0 , coor[prev1]);
    xbarC = xbar( 0 , 0, coor[prev1]);

    phibarC = phibar(0, 0, phi[prevX1][prevY1]);

    phiu = 0; phid = phi[prevX1][prevY1];

    phibarF = xbarF + xbarF*(xbarF-1)/(xbarC*(xbarC-1))*(phibarC-xbarC);
    */

    phiF = (phi[i] + phi[prevX1]) / 2;

} else if (next2 == -1){
    /*xbarF = xbar( coor[I] , coor[next1], coor[prev1]);
    xbarC = xbar( coor[next1] , coor[next1], coor[prev1]);

    phibarC = phibar(phi[nextX1][nextY1],
    phi[nextX1][nextX1], phi[prevX1][prevY1]);

    phiu = 0; phid = phi[prevX1][prevY1];

    phibarF = xbarF + xbarF*(xbarF-1)/(xbarC*(xbarC-1))*(phibarC-xbarC);*/

    phiF = (phi[i] + phi[nextX1] + phi[prevX1]) / 3;

    //cout << phiF << endl;

} else if (prev1 == -1) {
    /*xbarF = xbar( coor[I] , coor[next2], 0);
    xbarC = xbar( coor[next1] , coor[next2], 0);

    phibarC = phibar(phi[nextX1][nextY1], phi[nextX2][nextY2], 0);

    phiu = phi[nextX2][nextY2]; phid = 0;

    //cout << phibarC << " " << xbarC << endl;
    // FALLA EN phibarF, dona nan, suposo per quan xc (xc-1) = 0;
    //denominador es igual a 0
    phibarF = xbarF + xbarF*(xbarF-1)/(xbarC*(xbarC-1))*(phibarC-xbarC);
```

```
        cout << " --> " << phibarF << endl;

        phiF = phiu + (phid - phiu)*phibarF;*/

        phiF = (phi[i] + phi[nextX1] + phi[nextX2]) / 3;


    } else {

        xbarF = xbar( coor[I], coor[next2] , coor[prev1]);
        xbarC = xbar( coor[next1] , coor[next2], coor[prev1]);

        phibarC = phibar(phi[nextX1], phi[nextX2], phi[prevX1]);

        phiu = phi[nextX2]; phid = phi[prevX1];

        phibarF = xbarF + xbarF*(xbarF-1)/(xbarC*(xbarC-1))*(phibarC-xbarC);

        phiF = phiu + (phid - phiu)*phibarF;

    }

    // Hem de passar phibarF a phi

    // cout << phiF <<" "<< phibarF << endl;

    return phiF;
}
```

```
// MAIN CODE

int main() {

    int i=0; int j=0;

    Geometria2D (L, H, Nx, Nx);
    //Limits (L, H, Nx, Ny);
```

```
   // pasartablaaficheroMalla (CosFluid, Nx, 1);
CondInicials (Nx, Nx, Mass_0, VelX_0, VelY_0, Temp_0);
//MassEquation (Nx,Ny,T_step);


// Lambda in Momentum Eq = Viscosity MU
float  Gamma_Mom [2*dim+1];
for (i=0;i<(2*dim+1);i++) {
    Gamma_Mom[i] = mu;
}

// DENSITY
//ConvDiff (0, S, rho, 1, T_step) ;

cout << "    -----     DENSIDAD CALCULADA"<< endl;


// VELOCITY
ConvDiff (Gamma_Mom, S, VelX, 2, T_step, CondContornVel) ;

float VelXfinal = VelX[Nx-1][divT];

cout << VelXfinal << endl;

float Peclet = rho_0 * VelX_0 * L / mu;

double Error [Nx+1];

cout << "Peclet" << Peclet << endl;

for (i=0; i<=Nx; i++) {
    VelX_teo[i] = ( exp( ( COOR[i] *Peclet / L ) )-1) / ( exp(Peclet) -1 )
    * (VelXfinal - VelX_0) + VelX_0;
    Error[i] =   abs(VelX_teo[i]-VelX[i][0])/VelX_teo[i] * 100;
    //cout << i << " " << VelX[i][0] << "  /  " << VelX_teo[i] << endl;
    //cout << i << " " << Error[i] << endl;
}
pasartablaafichero1D(VelX,Nx,2);

float  Gamma_Energy [2*dim+1];
for (i=0;i<(2*dim+1);i++) {
    Gamma_Energy[i] = lambda/Cv;
}
```

```
    ConvDiff (Gamma_Energy, S, Temp, 4, T_step, CondContornTemp);

    pasartablaafichero1D(Temp,Nx,4);

    //ConvDiff (Lambda[2*dim+1], S[Nx+1][Ny+1][2*dim+1],
    //dpx[2*dim+1], phi[Nx+1][Ny+1], tipo);
    //for (k=0;k<iterations;k++) {
        //ConvDiff (0, S, deltas, rho, 1);
        //ConvDiff (Lambda, S, deltas, VelX, 2); // eix X
        //ConvDiff (Lambda, S, deltas, VelY, 3); // eix Y


    cout << "--- FIN DEL PROGRAMA ---" << endl;


    return 0;
}


void pasartablaafichero1D (float valor [Nx+1][divT+1],
int Nx, int tipo ) {

     ofstream prod;
        if (tipo == 4){
            prod.open("ConvDiff1D_TEMP2.dat");
        } if (tipo == 2) {
            prod.open("ConvDiff1D_VELX2.dat");
        }
        prod << "# X    Y" << endl;

       if (prod.is_open()) {
            int i = 0; int j =0;
                for (i=0; i<Nx; i++) {

             //prod << "Coordenadas: ";
             //prod << X2[i][0] <<" " << X2[i][1] << " ";
             //prod << " Cos (1) Fluid (0): " << CosFluid[i] <<endl;
             //prod << X2[i][0] << " " << X2[i][1] << " " <<
             //CosFluid[i] <<endl;

             prod << fixed << setprecision(5) << X[i] << "   ";
             prod << fixed << setprecision(5) << valor[i][divT] << endl;
```

```
            }

     prod.close();
    }
}
```

2 Diagonal Flow

2.1 Results

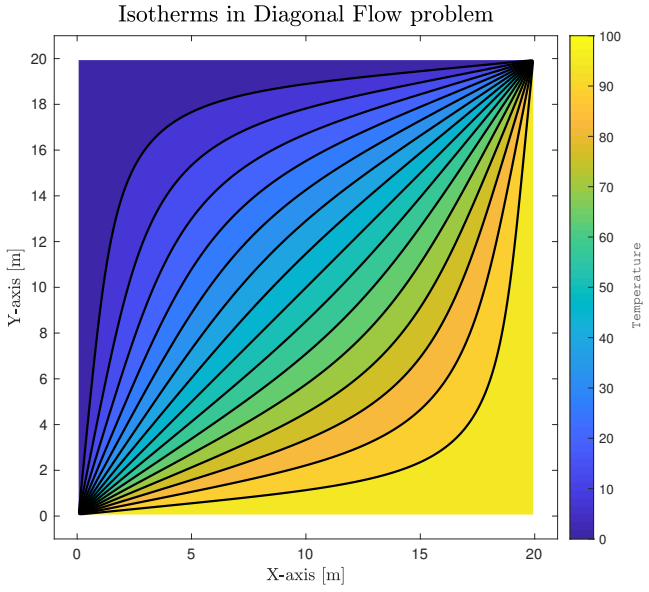

Figure 2.1: Isotherms in a Diagonal Flow (Pe = 0.6)

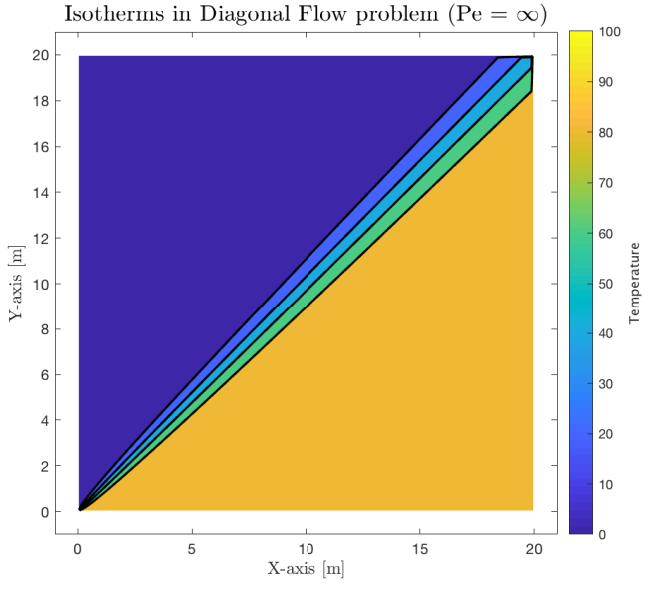

Figure 2.2: Isotherms in a Diagonal Flow ($Pe = \infty$)

2.2 Code in C++: CD_DiagonalFlow.cpp

```
#include <iostream>
# include <stdlib.h>
# include <fstream>
# include <math.h>
#include <iomanip>
#include <cstdlib>

//# include <windows.h>

using namespace std;

// CONSTANTES FÍSICAS

const float pi = 3.14159265;

// DATOS DEL PROBLEMA

const int Nx = 20; // number of divisions along dir. X
const int Ny = 20;
const float L = 10; // length of the canal in [m]
const float H = 20; // height of the canal in [m]
const float W = 5;  // width of the canal in [m]
const int dim = 2;
const float T_step = 0; // Estacionario


const float Mass_0 = 5; // Initial mass flux
const float rho_0 = 1.125; // Initial mass flux
const float Temp_0 = 273+15; // Temp [K]
const float VelX_0 = 50; // entrance's velocity  [m]
const float VelY_0 = 15;
const float mu = 1.2*pow(10,3);
const float u_inf = 50;
const int divT = 2;
float temps [divT+2];


// VARIABLES

    const int nnode = Nx*Ny-1;
    float COOR[Nx+1][Ny+1][dim];
    float X[Nx+1];
```

```
    float Y[Ny+1];
    float V[Nx+1][Ny+1];
    float S[Nx+1][Ny+1][2*dim];
    const float deltaX = L/Nx;
    const float deltaY = H/Ny;
    float deltas[2*dim];
    float MU [2*dim+1];

    float Temp [Nx+1][Ny+1];
    float Mass [Nx+1][Ny+1];
    float rho [Nx+1][Ny+1];
    float VelX [Nx+1][Ny+1];
    double VelX_teo [Nx+1][Ny+1];
    float VelY [Nx+1][Ny+1];
    double VelY_teo [Nx+1][Ny+1];
    float CondContorn [Nx+1][Ny+1];

    // TDMA line by line
    float bp_ast [Nx+1];
    float P[Nx+1];
    float R [Nx+1];

    float criteriConv = pow(10,-7);

    int north1 = 0; int south1 = 0;
    int north2 = 0; int south2 = 0;
    int west1 = 0; int west2 = 0;
    int east1 = 0; int east2 = 0;
    int point = 0;

// FUNCTIONS' MENTION

void Geometria2D (float L, float H, int Nx, int Ny);
void IdentifyNodes (int I, int J, int nx, int ny);

void Centroide (float COOR[Nx+1][Ny+1][dim], int Nx, int Ny);
float DegtoRad (float angle);

float Circulation (float vye, float vxn, float vyw, float vxs,
float deltaY, float deltaX);
float Harmonicmean (float dPE, float dPe, float dEe, float phiP,
float phiE) ;
float phi_UDS (float mflow, float phiP, float phiX);
float phi_HRS (int i, int j, int next1, int next2, int prev1,
```

```
float phi[Nx+1], int tipo, float coor[Nx]);
    float xbar (float x, float xu, float xd);
    float phibar (float phi, float phiu, float phid) ;
float GaussSeidel2D (float a_w, float a_e, float a_n, float a_s,
float a_p, float b_p, float phiW, float phiE, float phiN, float phiS) ;

void MassEquation (int nx, int ny, float T_step);

void ConvDiff (float Lambda[2*dim+1], float S[Nx+1][2*dim],
float phi[Nx+1][Ny+1], int tipo, float t_step);

// FUNCTIONS

void Geometria2D (float l, float h, int nx, int ny) {
    int i = 0; int j = 0;

   for (j=0; j<=ny; j++) {
         for (i=0; i<=nx; i++) {
            COOR [i][j][0] = (i)*deltaX;
            COOR [i][j][1] = (j)*deltaY ;
            //rho  [(nx+1)*j+i] = rho0;
        }
    }

    Centroide (COOR, nx, ny);
}


void Centroide (float  COOR[Nx+1][Ny+1][dim], int nx, int ny) {

    int i=0; int j=0;

   // MANERA DE HACERLO

            deltas [0] = deltaY; // Cara N
            deltas [1] = deltaY; // Cara S
            deltas [2] = deltaX; // Cara W
            deltas [3] = deltaX; // Cara E

   for (j=0; j<ny; j++) {
         for (i=0; i<nx; i++) {
            X [i] = (COOR[i][j][0]+COOR[i+1][j][0])/2;
            Y [j] = (COOR[i][j][1]+COOR[i][j+1][1])/2;
```

```
            V[i][j] = deltaX*deltaY*W;
            S[i][j][0] = deltaX*W; // Cara N
            S[i][j][1] = deltaX*W; // Cara S
            S[i][j][2] = deltaY*W; // Cara W
            S[i][j][3] = deltaY*W; // Cara E

        }
    }

    cout << "1. Malla calculada" << endl;

    /*
    cout << "---------- Coordenadas centroides 2 ----------" << endl;
    for (j=0; j<ny; j++) {
        for (i=0; i<nx; i++) {
            cout << "Punto " << i << " " << j;
            cout << " = Comp X: " << X[i];
            cout << " // Comp Y: " << Y[j] << endl;
            cout << "Sx: " << S[i][j][2] << " Sy: " << S[i][j][0] <<
            " V: " << V[i][j] << endl;
    }
    }
    cout << "------ End of Coordenadas centroides 2 ------" << endl;
    */

}



void IdentifyNodes (int I, int J, int nx, int ny) {

    north1 = J+1;
        north2 = J+2;
    south1 = J-1;
        south2 = J+2;
    west1 = I-1;
        west2 = I-2;
    east1 = I+1;
        east2 = I+2;

    if ( J == ny-1) {
        north1 = -1;
    }
    if ( J == 0 ) {
```

```
        south1 = -1;
    }
    if ( I == 0) {
        west1 = -1;
    }
    if ( I == nx-1) {
        east1 = -1;
    }


    if ( J == ny-2 ||  J == ny-1) {
        north2 = -1;
    }
    if ( J == 0 ) {
        south2 = -1;
    }
    if ( I == 0) {
        west2 = -1;
    }
    if ( I == nx-2 ||  I == nx-1) {
        east2 = -1;
    }

    ////cout << "P: "<<I<<" "<<J<< " / N: " << north << " / S: " << south
    << " / W: " << west << " / E: " << east << endl;
    //cout << "P: "<<I<<" "<<J<< " // N: " << north1 << " /NN: "<< north2
    << " // S: " << south1 <<" /SS: " << south2;
    //cout << " // W: " << west1 << " / WW: " << west2 << " // E: "
    << east1 << " / EE: " << east2 << endl;
}

void CondInicials (int nx, int ny, float Mass_0, float VelX_0,
float VelY_0, float Temp_0) {

    float xc = L / 2;
    float yc = 0;

    // phi = Mass, Velocity and Temperature
    int i = 0;
    int j = 0;

    for (j=0; j<ny; j++) {
         for (i=0; i<nx; i++) {
            Mass[i][j]=Mass_0;
```

```
            rho[i][j] = rho_0;
            Temp[i][j]=Temp_0;
            VelX[i][j] = VelX_0;
            VelY[i][j]=0;

            CondContorn[i][j] = 0;

            if (i==0){
                CondContorn[i][j] = 1;
                VelX[i][j] = VelX_0;
            }

            if (i==Nx-1) {
                CondContorn[i][j] = 1;
                VelX[i][j] = VelX_0 - 8.93;
            }

        }
    }

    cout << "2. Condiciones Iniciales establecidas" << endl;

}


float Harmonicmean (float dPE, float dPe, float dEe, float phiP,
float phiE) {
    float phi = dPE / (dPe/phiP + dEe/phiE);
    return phi ;
}

float GaussSeidel (float a_w, float a_e, float a_p, float b_p,
float phiW, float phiE)    {
    float phi;

    phi = (a_w*phiW + a_e*phiE + b_p) / a_p;

    return phi;
}

float GaussSeidel2D (float a_w, float a_e, float a_n, float a_s,
float a_p, float b_p, float phiW, float phiE, float phiN, float phiS)    {
    float phi;
```

```
    phi = (a_w*phiW + a_e*phiE + a_n*phiN + a_s*phiS + b_p) / a_p;

    return phi;
}


void ConvDiff (float Lambda[2*dim+1], float S[Nx+1][Ny+1][2*dim],
float phi[Nx+1][Ny+1], int tipo, float t_step) {

    int i =0; int j =0;

    float mflowW = 0; float mflowE = 0;

    float speedX_e = 0; float speedX_w = 0; float speedY_n = 0;
    float speedY_s = 0;

    float rho_e = 0; float rho_w = 0; float rho_s = 0; float rho_n = 0;
    float rhoM = 0;

    float Fe = 0; float Fw = 0; float Fs = 0; float Fn = 0;

    float dPEx = 0; float dPex = 0; float dEex = 0; float dPNy = 0;
    float dPny = 0; float dNny = 0;

    float new_phi = 0;

    float Dw = 0; float De = 0; float Dn = 0; float Ds = 0;

    float phi_UDS_e = 0; float phi_UDS_w = 0; float phi_UDS_n = 0;
    float phi_UDS_s = 0;
    float phi_HRS_e = 0; float phi_HRS_w = 0; float phi_HRS_n = 0;
    float phi_HRS_s = 0;

    int bucles = 0;

    int p = 0; int x = 0;

    float aw = 0;
    float ae = 0;
    float an = 0;
    float as = 0;
    float ap_0 = 0;
    float ap = 0;
    float bp = 0;
```

```
        float bp_exp = 0;
        float qp = 0;

    //float TIME = 0;

    float no=0;

    int loops = 1000;

            x = 0;

            while (x < loops) {

                for (j=0; j<Ny; j++) {
                for (i=0; i<Nx; i++) {

                    IdentifyNodes(i,j,Nx,Nx);

                    dPEx = deltaX;  dPex = deltaX/2;  dEex = deltaX/2;
                    dPNy = deltaY;  dPny = deltaX/2;  dNny = deltaX/2;

                    speedX_e = Harmonicmean(dPEx, dPex, dEex,
                    VelX[i][j], VelX[east1][j]);
                    speedX_w = Harmonicmean(dPEx, dPex, dEex,
                    VelX[i][j], VelX[west1][j]);
                    speedY_n = Harmonicmean(dPNy, dPny, dNny,
                    VelX[i][j], VelX[i][north1]);
                    speedY_s = Harmonicmean(dPNy, dPny, dNny,
                    VelX[i][j], VelX[i][south1]);

                    rho_e = Harmonicmean(dPEx, dPex, dEex,
                    rho[i][j], rho[east1][j]);
                    rho_w = Harmonicmean(dPEx, dPex, dEex,
                    rho[i][j], rho[west1][j]);
                    rho_n = Harmonicmean(dPNy, dPny, dNny,
                    rho[i][j], rho[i][north1]);
                    rho_s = Harmonicmean(dPNy, dPny, dNny,
                    rho[i][j], rho[i][south1]);

                    Fe = rho_e*speedX_e; Fw = rho_w*speedX_w;
                    Fn = rho_n*speedY_n; Fs = rho_s*speedY_s;

                        // CENTRAL DIFFERENCE SCHEME
```

```
        //ae = De - Fe/2;
        //aw = Dw + Fw/2;

    // UPWIND SCHEME

        ae = De + phi_UDS (-Fe, phi[i][j], phi[east1][j]);
        aw = Dw + phi_UDS (Fw, phi[i][j], phi[west1][j]);
        ae = Dn + phi_UDS (-Fn, phi[i][j], phi[i][north1]);
        aw = Ds + phi_UDS (Fs, phi[i][j], phi[i][south1]);

        De = Lambda[0] / deltaX; Dw = Lambda[1] / deltaX;
        Dn = Lambda[0] / deltaY; Ds = Lambda[1] / deltaY;

    if (tipo == 1){
        De = 0; Dw = 0;
    }

    if (east1 == -1){
        ae = 0;
        Fe = rho[i][j]*VelX[i][j];
    }


    if (west1 == -1){
        aw = 0;
        Fw = rho[i][j]*VelX[i][j];
    }

    if (north1 == -1) {
        an = 0;
        Fn = rho[i][j]*VelY[i][j];
    }

    if (south1 == -1) {
        as = 0;
        Fs = rho[i][j]*VelY[i][j];
    }

    if (tipo == 2) { // Momentum en X
        an = 0; as = 0;
        Fn = 0; Fs = 0;
    }

    if (tipo == 3) {  // Momentum en Y
```

```
                        ae = 0; aw = 0;
                        Fe = 0; Fw = 0;
                    }

                    ap = ae + aw + an + as + (Fe - Fw) + (Fn - Fs);

                    bp = 0;


                    //T[i][p+1] = TDMA(aw,ae,ap,bp, T[i-1][p+1], T[i+1][p+1],i)
                    new_phi = GaussSeidel2D(aw,ae,an,as,ap,bp, phi[west1][j],
                    phi[east1][j], phi[i][north1], phi[i][south1]);

                    if ( abs(new_phi - phi[i][j]) > criteriConv &&
                    CondContorn[i][j] == 0) {
                        phi[i][j] = new_phi;
                    }

                }
            }
             x++;
            }

}

float phi_UDS (float mflow, float phiP, float phiX){

    int fe;
    float phi=0;

    if (mflow>0) {
        fe = 0;
    } else {
        fe = 1;
    }

    phi = fe * (phiX-phiP)+phiP;

    return phi;

}


float phibar (float phi, float phiu, float phid) {
```

```
    if (phid == phiu) {
        return 1;
    } else {
        float phibar = (phi-phiu)/(phid-phiu);
        return phibar;
    }
}

float xbar (float x, float xu, float xd) {

    if (xu == xd) {
        return 1;
    }else {
        float xbar = (x-xu)/(xd-xu);
        return xbar;
    }

}

float phi_HRS (int i, int j, int next1, int next2, int prev1,
float phi[Nx+1], int tipo, float coor[Nx+1]) {

    float xbarF = 0;
    float xbarC = 0;
    float phibarC = 0;
    float phibarF = 0;

    int nextX1 = 0; int nextX2 = 0; int prevX1 = 0;
    int nextY1 = 0; int nextY2 = 0; int prevY1 = 0;

    int I = 0;

    float phiu = 0; float phid = 0;
    float phiF = 0;

    if (tipo == 1){ // En direccio x , y no varia
        nextX1 = next1; nextX2 = next2; prevX1 = prev1;
        nextY1 = j; nextY2 = j; prevY1 = j;

        I = i;

    }
```

```
if (tipo ==2){ // En direccio y , x no varia
    nextY1 = next1; nextY2 = next2; prevY1 = prev1;
    nextX1 = i; nextX2 = i; prevX1 = i;

    I = j;

}

//cout << nextX1 << " " << nextX2 << " " << prevX1;
//cout << "  //  " << nextY1 << " " << " " << prevY1 << endl;

// SMART
if (next1 == -1){
    /*xbarF = xbar( coor[I], 0 , coor[prev1]);
    xbarC = xbar( 0 , 0, coor[prev1]);

    phibarC = phibar(0, 0, phi[prevX1][prevY1]);

    phiu = 0; phid = phi[prevX1][prevY1];

    phibarF = xbarF + xbarF*(xbarF-1)/(xbarC*(xbarC-1))*(phibarC-xbarC);
    */

    phiF = (phi[i] + phi[prevX1]) / 2;

} else if (next2 == -1){
    /*xbarF = xbar( coor[I] , coor[next1], coor[prev1]);
    xbarC = xbar( coor[next1] , coor[next1], coor[prev1]);

    phibarC = phibar(phi[nextX1][nextY1], phi[nextX1][nextX1],
    phi[prevX1][prevY1]);

    phiu = 0; phid = phi[prevX1][prevY1];

    phibarF = xbarF + xbarF*(xbarF-1)/(xbarC*(xbarC-1))*(phibarC-xbarC);*/

    phiF = (phi[i] + phi[nextX1] + phi[prevX1]) / 3;

    //cout << phiF << endl;

} else if (prev1 == -1) {
    /*xbarF = xbar( coor[I] , coor[next2], 0);
    xbarC = xbar( coor[next1] , coor[next2], 0);
```

```
        phibarC = phibar(phi[nextX1][nextY1], phi[nextX2][nextY2], 0);

        phiu = phi[nextX2][nextY2]; phid = 0;

        //cout << phibarC << " " << xbarC << endl;
        // FALLA EN phibarF, dona nan, suposo per quan xc (xc-1) = 0;
        denominador es igual a 0
        phibarF = xbarF + xbarF*(xbarF-1)/(xbarC*(xbarC-1))*(phibarC-xbarC);

        cout << " --> " << phibarF << endl;

        phiF = phiu + (phid - phiu)*phibarF;*/

        phiF = (phi[i] + phi[nextX1] + phi[nextX2]) / 3;


    } else {

        xbarF = xbar( coor[I], coor[next2] , coor[prev1]);
        xbarC = xbar( coor[next1] , coor[next2], coor[prev1]);

        phibarC = phibar(phi[nextX1], phi[nextX2], phi[prevX1]);

        phiu = phi[nextX2]; phid = phi[prevX1];

        phibarF = xbarF + xbarF*(xbarF-1)/(xbarC*(xbarC-1))*(phibarC-xbarC);

        phiF = phiu + (phid - phiu)*phibarF;

    }

    // Hem de passar phibarF a phi

    // cout << phiF <<" "<< phibarF << endl;

    return phiF;
}
```

```
// MAIN CODE

int main() {

    int i=0; int j=0;

    Geometria2D (L, H, Nx, Nx);
        Limits (L, H, Nx, Ny);
        pasartablaaficheroMalla (CosFluid, Nx, 1);
    CondInicials (Nx, Nx, Mass_0, VelX_0, VelY_0, Temp_0);

        float  Lambda [2*dim+1];
            for (i=0;i<(2*dim+1);i++) {
                Lambda[i] = mu;
            }


    // VELOCITY
    ConvDiff (Lambda, S, VelX, 2, T_step) ;

    float VelXfinal = VelX[Nx-1][0];
    cout << VelXfinal << endl;
    float Peclet = rho_0 * VelX_0 * L / mu;
    double Error [Nx+1];
    cout << Peclet << endl;

        for (j=0; j<Ny; j++) {
            for (i=0; i<Nx; i++) {
                cout << i << " " << j << " : " << VelX[i][j] << endl;
            }
        }

    ConvDiff (Lambda[2*dim+1], S[Nx+1][Ny+1][2*dim+1], dpx[2*dim+1],
    phi[Nx+1][Ny+1], tipo);
        //ConvDiff (0, S, deltas, rho, 1);
        //ConvDiff (Lambda, S, deltas, VelX, 2); // eix X
        //ConvDiff (Lambda, S, deltas, VelY, 3); // eix Y

    cout << "--- FIN DEL PROGRAMA ---" << endl;


    return 0;
}
```

3 Smith-Hutton problem

3.1 Results

Using Central Difference scheme (CDS):

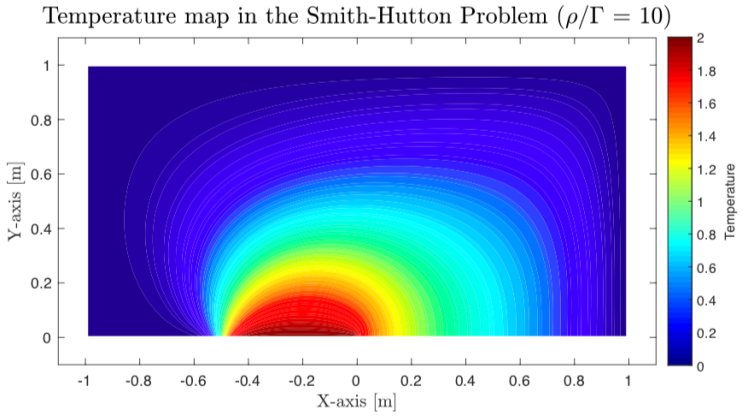

Figure 3.1: CDS - Solution of Smith-Hutton problem for Pe = 10

Using Upwind Difference scheme (UDS):

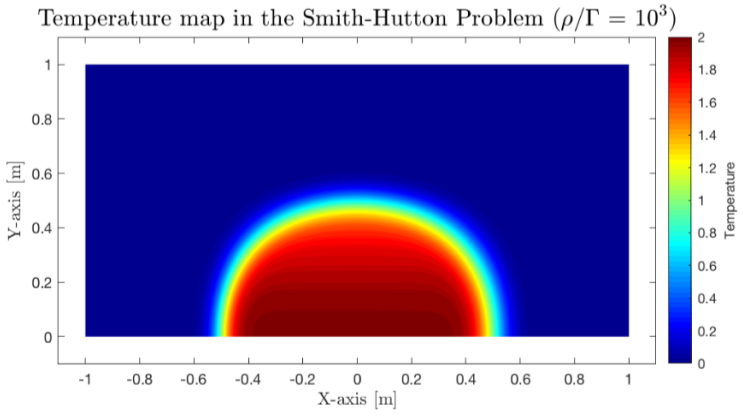

Figure 3.2: UDS - Solution of Smith-Hutton problem for Pe = 10^3

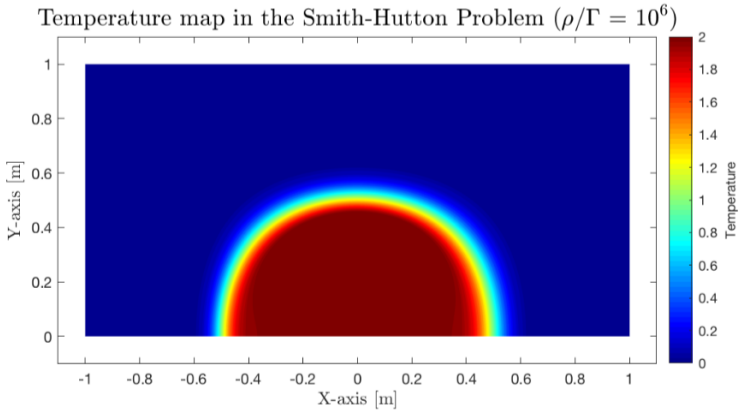

Figure 3.3: UDS - Solution of Smith-Hutton problem for Pe = 10^6

Using Quadratic Upwind Interpolation for convective kinematic (QUICK):

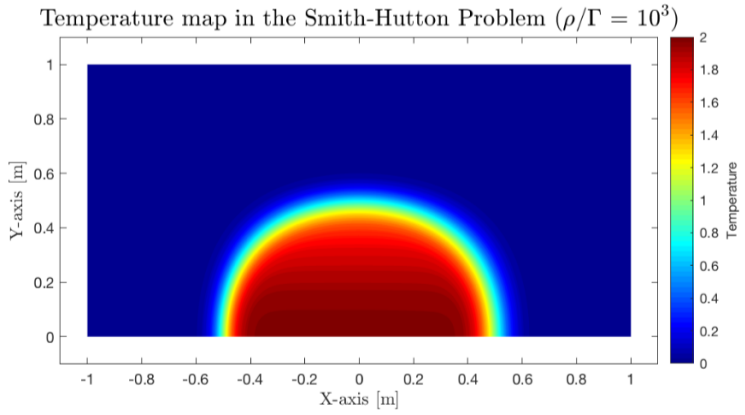

Figure 3.4: QUICK - Solution of Smith-Hutton problem for Pe = 10^3

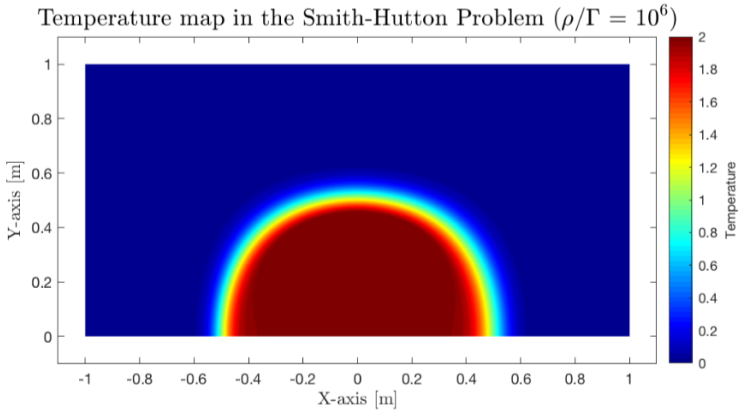

Figure 3.5: QUICK - Solution of Smith-Hutton problem for Pe = 10^6

3.2 Code in C++: CD_SmithHutton.cpp

```
#include <iostream>
# include <stdlib.h>
# include <fstream>
# include <math.h>
#include <iomanip>
#include <cstdlib>

//# include <windows.h>

using namespace std;

// CONSTANTES FÍSICAS

const float pi = 3.14159265;

// DATOS DEL PROBLEMA

const int Nx = 100; // number of divisions along dir. X
const int Ny = 100;
const float L = 2; // length of the canal in [m]
const float H = 1; // height of the canal in [m]
const float W = 1;  // width of the canal in [m]
const int dim = 2;
const float T_step = 0; // Estacionario


const float Mass_0 = 5; // Initial mass flux
const float rho_0 = pow(10,6); // Initial mass flux
const float Temp_0 = 273+15; // Temp [K]
const float VelX_0 = 1; // entrance's velocity  [m]
const float VelY_0 = 0;
const float mu = 1.5*pow(10,3);
const float Cv = 1; //0.0050; 4.186/1.4
const float lambda = 1;
const float u_inf = 50;
const int divT = 2;
float temps [divT+2];

int scheme = 1;


// VARIABLES
```

```
    const int nnode = Nx*Ny-1;
    float COOR[Nx+1][Ny+1][dim+1];
    float X[Nx+1];
    float Y[Ny+1];
    float V[Nx+1][Ny+1];
    float S[Nx+1][Ny+1][2*dim];
    const float deltaX = L/Nx;
    const float deltaY = H/Ny;
    float deltas[2*dim];
    float MU [2*dim+1];

    float Temp [Nx+1][Ny+1];
    float Mass [Nx+1][Ny+1];
    float rho [Nx+1][Ny+1];
    float VelX [Nx+1][Ny+1];
    double VelX_teo [Nx+1][Ny+1];
    float VelY [Nx+1][Ny+1];
    double VelY_teo [Nx+1][Ny+1];
    float CondContornVel [Nx+1][Ny+1];
    float CondContornTemp [Nx+1][Ny+1];

    //float ae [Nx+1][Ny+1];
    //float aw [Nx+1][Ny+1];
    //float an [Nx+1][Ny+1];
    //float as [Nx+1][Ny+1];
    //float ap [Nx+1][Ny+1];
    //float bp [Nx+1][Ny+1];

    // TDMA line by line
    float bp_ast [Nx+1];
    float P[Nx+1];
    float R [Nx+1];

    float criteriConv = pow(10,-7);

    int north1 = 0; int south1 = 0;
    int north2 = 0; int south2 = 0;
    int west1 = 0; int west2 = 0;
    int east1 = 0; int east2 = 0;
    int point = 0;

// FUNCTIONS' MENTION
```

```
void Geometria2D (float L, float H, int Nx, int Ny);
void IdentifyNodes (int I, int J, int nx, int ny);

void Centroide (float COOR[Nx+1][Ny+1][dim+1], int Nx, int Ny);
float DegtoRad (float angle);

float Circulation (float vye, float vxn, float vyw,
float vxs, float deltaY, float deltaX);
float Harmonicmean (float dPE, float dPe, float dEe,
float phiP, float phiE) ;
float phi_UDS (float mflow, float phiP, float phiX);
float phi_HRS (int i, int j, int next1, int next2,
int prev1, int prev2, float phi[Nx+1][Ny+1], float coor[Nx+1], float F);
    float xbar (float x, float xu, float xd);
    float phibar (float phi, float phiu, float phid) ;
float GaussSeidel2D (float a_w, float a_e,
float a_n, float a_s, float a_p, float b_p,
float phiW, float phiE, float phiN, float phiS) ;

void MassEquation (int nx, int ny, float T_step);

void pasartablaafichero2D (float valor [Nx+1][Ny+1],
int Nx, int Ny, int tipo );

void CondInicials (int nx, int ny, float Mass_0,
float VelX_0, float VelY_0, float Temp_0,
float X[Nx+1], float Y[Ny+1]) ;
void ConvDiff (float Lambda[2*dim+1],
float S[Nx+1][2*dim], float phi[Nx+1][Ny+1],
int tipo, float t_step, float CondContorn[Nx+1][Ny+1]);

// FUNCTIONS

void Geometria2D (float l, float h, int nx, int ny) {
    int i = 0; int j = 0;

   for (i=0; i<=nx; i++) {
         for (j=0; j<=ny; j++) {
            COOR [i][j][0] = -L/2+(i)*deltaX; // x
            COOR [i][j][1] = (j)*deltaY; // y
            X [i] = -L/2+(i)*deltaX; // x
            Y [j] = (j)*deltaY; // y
            //rho  [(nx+1)*j+i] = rho0;
        }
```

```
    }
    /*
    cout << "---------- Coordenadas centroides 2 ----------" << endl;
    for (j=0; j<=ny; j++) {
         for (i=0; i<=nx; i++) {
            cout << "Punto " << i << " " << j;
            cout << " = Comp X: " << COOR [i][j][0] ;
            cout << " // Comp Y: " << COOR [i][j][1]  << endl;
            //cout << "Sx: " << S[i][j][2] << " Sy: " <<
            S[i][j][0] << " V: " << V[i][j] << endl;
    }
    }
    cout << "------ End of Coordenadas centroides 2 ------" << endl;
    */

    //Centroide (COOR, nx, ny);
}


void Centroide (float  COOR[Nx+1][Ny+1][dim+1], int nx, int ny) {

    int i=0; int j=0;

   // MANERA DE HACERLO

            deltas [0] = deltaY; // Cara N
            deltas [1] = deltaY; // Cara S
            deltas [2] = deltaX; // Cara W
            deltas [3] = deltaX; // Cara E

    for (i=0; i<nx; i++) {
        for (j=0; j<ny; j++) {
            X [i] = (COOR[i][j][0]+COOR[i+1][j][0])/2;
            Y [j] = (COOR[i][j][1]+COOR[i][j+1][1])/2;

            V[i][j] = deltaX*deltaY*W;
            S[i][j][0] = deltaX*W / H; // Cara N
            S[i][j][1] = deltaX*W / H; // Cara S
            S[i][j][2] = deltaY*W / H; // Cara W
            S[i][j][3] = deltaY*W / H; // Cara E

         }
    }
```

```
    cout << "1. Malla calculada" << endl;

    /*
    cout << "---------- Coordenadas centroides 2 ----------" << endl;
    for (j=0; j<ny; j++) {
         for (i=0; i<nx; i++) {
            cout << "Punto " << i << " " << j;
            cout << " = Comp X: " << X[i];
            cout << " // Comp Y: " << Y[j] << endl;
            //cout << "Sx: " << S[i][j][2] << " Sy: "
            << S[i][j][0] << " V: " << V[i][j] << endl;
    }
    }

    cout << "------ End of Coordenadas centroides 2 ------" << endl;
    */
}



void IdentifyNodes (int I, int J, int nx, int ny) {

    north1 = J+1;
        north2 = J+2;
    south1 = J-1;
        south2 = J+2;
    west1 = I-1;
        west2 = I-2;
    east1 = I+1;
        east2 = I+2;

    if ( J == ny) {
        north1 = -1;
    }
    if ( J == 0 ) {
        south1 = -1;
    }
    if ( I == 0) {
        west1 = -1;
    }
    if ( I == nx) {
        east1 = -1;
    }
```

```
    if ( J == ny-1 ||  J == ny) {
        north2 = -1;
    }
    if ( J == 0 ) {
        south2 = -1;
    }
    if ( I == 0) {
        west2 = -1;
    }
    if ( I == nx-1 ||  I == nx) {
        east2 = -1;
    }

    //cout << "P: "<<I<<" "<<J<< " / N: " << north << " /
    S: " << south << " / W: " << west << " / E: " << east << endl;
    //cout << "P: "<<I<<" "<<J<< " // N: " << north1 <<
    " /NN: "<< north2 << " // S: " << south1 <<" /SS: " << south2;
    //cout << " // W: " << west1 << " / WW: " << west2 <<
    " // E: " << east1 << " / EE: " << east2 << endl;
}

void CondInicials (int nx, int ny, float Mass_0, float VelX_0,
float VelY_0, float Temp_0, float X[Nx+1], float Y[Ny+1]) {

    float xc = L / 2;
    float yc = 0;

    // phi = Mass, Velocity and Temperature
    int i = 0;
    int j = 0;

    float x = 0; float y = 0;

    for (i=0; i<=nx; i++) {
        for (j=0; j<=ny; j++) {
            Mass[i][j]=Mass_0;
            rho[i][j] = rho_0;
            Temp[i][j]= 1;

            x = X[i]/(H);
            y = Y[j]/(H);

            VelX[i][j] = 2 * y * (1 - x*x);
```

```
            VelY[i][j] = -2 * x * (1 - y*y);

            CondContornTemp[i][j] = 0;


            if ( (x >= -1-0.00001 && x-0.00001 <= 0) && j == 0) {
                Temp[i][j] = 1 + tanh (10*( 2*x + 1));
                CondContornTemp[i][j] = 1;
            }

            if ( (i == 0 || j == Ny -1 || i == Nx-1) &&
            CondContornTemp[i][j] == 0 ) {
                Temp[i][j] = 0.0; // 1 - tanh (10);
                CondContornTemp[i][j] = 1;
            }

        }
    }

    cout << "2. Condiciones Iniciales establecidas" << endl;

}

float GaussSeidel (float a_w, float a_e, float a_p,
float b_p, float phiW, float phiE)    {
    float phi;

    phi = (a_w*phiW + a_e*phiE + b_p) / a_p;

    return phi;
}

float GaussSeidel2D (float a_w, float a_e, float a_n,
float a_s, float a_p, float b_p, float phiW, float phiE,
float phiN, float phiS)    {
    float phi;

    phi = (a_w*phiW + a_e*phiE + a_n*phiN + a_s*phiS + b_p) / a_p;

    return phi;
}
```

```
float phi_UDS (float mflow, float phiP, float phiX){

    int fe;
    float phi=0;

    if (mflow>0) {
        fe = 0;
    } else {
        fe = 1;
    }

    phi = fe * (phiX-phiP)+phiP;

    return phi;

}


float phibar (float phi, float phiu, float phid) {

    if (phid == phiu) {
        return 1;
    } else {
        float phibar = (phi-phiu)/(phid-phiu);
        return phibar;
    }
}

float xbar (float x, float xu, float xd) {

    if (xu == xd) {
        return 1;
    }else {
        float xbar = (x-xu)/(xd-xu);
        return xbar;
    }

}

float phi_HRS (int i, int j, int next1, int next2,
int prev1, int prev2, float phi[Nx+1][Ny+1],
float coor[Nx+1], float F) {

    float xbarF = 0;
```

```
float xbarC = 0;
float phibarC = 0;
float phibarF = 0;

int nextX1 = 0; int nextX2 = 0; int prevX1 = 0;
int nextY1 = 0; int nextY2 = 0; int prevY1 = 0;

int I = 0;

float phiu = 0; float phid = 0;
float phiF = 0;

if ( F >=0 ){
    nextX1 = next1; nextX2 = next2; prevX1 = prev1;
    nextY1 = j; nextY2 = j; prevY1 = j;

    I = i;

} else {

    nextY1 = next1; nextY2 = next2; prevY1 = prev1;
    nextX1 = i; nextX2 = i; prevX1 = i;

    I = j;

}

//cout << nextX1 << " " << nextX2 << " " << prevX1;
//cout << "  //  " << nextY1 << " " << " " << prevY1 << endl;

// SMART
if (next1 == -1){
    /*xbarF = xbar( coor[I], 0 , coor[prev1]);
    xbarC = xbar( 0 , 0, coor[prev1]);

    phibarC = phibar(0, 0, phi[prevX1][prevY1]);

    phiu = 0; phid = phi[prevX1][prevY1];

    phibarF = xbarF + xbarF*(xbarF-1)/(xbarC*(xbarC-1))*
    (phibarC-xbarC);
    */

    phiF = (phi[i][j] + phi[prevX1][prevY1]) / 2;
```

```
} else if (next2 == -1){
    /*xbarF = xbar( coor[I] , coor[next1], coor[prev1]);
    xbarC = xbar( coor[next1] , coor[next1], coor[prev1]);

    phibarC = phibar(phi[nextX1][nextY1],
    phi[nextX1][nextX1], phi[prevX1][prevY1]);

    phiu = 0; phid = phi[prevX1][prevY1];

    phibarF = xbarF + xbarF*(xbarF-1)/(xbarC*(xbarC-1))*(phibarC-xbarC);*/

    phiF = (phi[i][j] + phi[nextX1][nextY1] +
    phi[prevX1][prevY1]) / 3;

    //cout << phiF << endl;

} else if (prev1 == -1) {
    /*xbarF = xbar( coor[I] , coor[next2], 0);
    xbarC = xbar( coor[next1] , coor[next2], 0);

    phibarC = phibar(phi[nextX1][nextY1],
    phi[nextX2][nextY2], 0);

    phiu = phi[nextX2][nextY2]; phid = 0;

    //cout << phibarC << " " << xbarC << endl;
    // FALLA EN phibarF, dona nan, suposo per quan xc (xc-1) = 0;
    //denominador es igual a 0
    phibarF = xbarF + xbarF*(xbarF-1)/
    (xbarC*(xbarC-1))*(phibarC-xbarC);

    cout << " --> " << phibarF << endl;

    phiF = phiu + (phid - phiu)*phibarF;*/

    phiF = (phi[i][j] + phi[nextX1][nextY1]) / 2;


} else {
    xbarF = xbar( coor[I], coor[next2] , coor[prev1]);
    xbarC = xbar( coor[next1] , coor[next2], coor[prev1]);

    phibarC = phibar(phi[nextX1][nextY1], phi[nextX2][nextY2],
```

```
        phi[prevX1][prevY1]);

        phiu = phi[nextX2][nextY2]; phid = phi[prevX1][prevY1];

        phibarF = xbarF + xbarF*(xbarF-1)/(xbarC*(xbarC-1))*
        (phibarC-xbarC);

        phiF = phiu + (phid - phiu)*phibarF;

    }

    return phiF;
}


void ConvDiff (float Lambda[2*dim+1], float S[Nx+1][Ny+1][2*dim],
float phi[Nx+1][Ny+1], int tipo, float t_step,
float CondContorn [Nx+1][Ny+1]) {

    int i =0; int j =0;

    float speedX_e = 0; float speedX_w = 0;
    float speedY_n = 0; float speedY_s = 0;

    float rho_e = 0; float rho_w = 0;
    float rho_s = 0; float rho_n = 0;
    float rhoM = 0;

    float fe = 0; float fw = 0; float fs = 0; float fn = 0;
    float Fe = 0; float Fw = 0; float Fs = 0; float Fn = 0;
    float xe = 0; float ye = 0; float xw = 0; float yw = 0;
    float xn = 0; float yn = 0; float xs = 0; float ys = 0;

    float dPEx = 0; float dPex = 0; float dEex = 0;
    float dPNy = 0; float dPny = 0; float dNny = 0;
    float dPWx = 0; float dPwx = 0; float dWwx = 0;
    float dPSy = 0; float dPsy = 0; float dSsy = 0;
    float dEy = 0; float dSx = 0;
    float dWy = 0; float dNx = 0;

    float phiHRSe = 0; float phiHRSw = 0;
    float phiHRSn = 0; float phiHRSs = 0;
    float phiUDSe = 0; float phiUDSw = 0;
    float phiUDSn = 0; float phiUDSs = 0;
```

```
float new_phi = 0;

float Dw = 0; float De = 0; float Dn = 0; float Ds = 0;

int bucles = 0;

int p = 0; int x = 0;

float aw = 0;
float ae = 0;
float an = 0;
float as = 0;
float ap_0 = 0;
float ap = 0;
float bp = 0;
    float bp_exp = 0;
    float qp = 0;

//float TIME = 0;

float no=0;

int loops = 20000;

        x = 0;

        while (x < loops) {

            for (j=0; j<=Ny; j++) {
            for (i=0; i<=Nx; i++) {

                IdentifyNodes(i,j,Nx,Ny);

                dPEx = deltaX;  dPex = deltaX/2;  dEex = deltaX/2;
                dPNy = deltaY;  dPny = deltaY/2;  dNny = deltaY/2;
                dPWx = deltaX;  dPwx = deltaX/2;  dWwx = deltaX/2;
                dPSy = deltaY;  dPsy = deltaY/2;  dSsy = deltaY/2;

                dSx = deltaX; dNx = deltaX;
                dEy = deltaY; dWy = deltaY;

                xe = (X[i] + X[east1])/(2*H);
                ye = (Y[j] + Y[j])/(2*H);
```

```
xw = (X[i] + X[west1])/(2*H);
yw = (Y[j] + Y[j])/(2*H);
xn = (X[i] + X[i])/(2*H);
yn = (Y[j] + Y[north1])/(2*H);
xs = (X[i] + X[i])/(2*H);
ys = (Y[i] + Y[south1])/(2*H);

if (north1 == -1){
    yn = 0; // Y[j];//+ dPny;
}
if (west1 == -1){
    xw = 0; // X[i];//- dPwx;
}
if (east1 == -1){
    xe = 0; // X[i];//+ dPex;
}
if (south1 == -1){
    ys = 0; // Y[j];//- dPsy;
}

speedX_e = 2 * ye * (1 - xe*xe);
speedX_w = 2 * yw * (1 - xw*xw);
speedY_n = -2 * xn * (1 - yn*yn);
speedY_s = -2 * xn * (1 - yn*yn);

rho_e = rho_0; rho_w = rho_0;
rho_n = rho_0; rho_s = rho_0;

    Fn = rho_0 * speedY_n * dNx; // Cara N
    Fs = rho_0 * speedY_s * dSx; // Cara S
    Fw = rho_0 * speedX_w * dWy; // Cara W
    Fe = rho_0 * speedX_e * dEy; // Cara E

    fn = 0; fe = 0; fn =0; fs = 0;

    if ( - speedY_n >= 0){
        fn = 1;
    }

    if ( + speedY_s >= 0){
        fs = 1;
    }

    if ( - speedX_e >= 0){
```

```
        fe = 1;
    }

    if ( + speedX_w >= 0){
        fw = 1;
    }

    switch {

    case scheme == '1':

    // CENTRAL DIFFERENCE SCHEME

        an = - Lambda[0] / rho_0 * dNx / dPNy + Fn/2;
        as = - Lambda[0] / rho_0 * dSx / dPSy - Fs/2;
        aw = - Lambda[0] / rho_0 * dWy / dPWx - Fw/2;
        ae = - Lambda[0] / rho_0 * dEy / dPEx + Fe/2;

    case scheme == '2':

        // UPWIND DIFFERENCE SCHEME


        an = - Lambda[0] / rho_0 * dNx / dPNy + max(-Fn,0);
        as = - Lambda[0] / rho_0 * dSx / dPSy + max(Fs,0);
        aw = - Lambda[0] / rho_0 * dWy / dPWx + max(-Fe,0);
        ae = - Lambda[0] / rho_0 * dEy / dPEx + max(Fe,0);

    case scheme == '3':

        an = - Lambda[0] / rho_0 * dNx / dPNy - (Fe-abs(Fe))/2;
        as = - Lambda[0] / rho_0 * dSx / dPSy + (Fw+abs(Fe))/2;
        aw = - Lambda[0] / rho_0 * dWy / dPWx - (Fn-abs(Fn))/2;
        ae = - Lambda[0] / rho_0 * dEy / dPEx + (Fe+abs(Fs))/2;

phiHRSe = phi_HRS(i,j,east1,east2,
west1,west2,phi,X,-Fe);
phiHRSw = phi_HRS(i,j,west1,west2,
east1,east2,phi,X,Fw);
phiHRSn = phi_HRS(i,j,north1,north2,
south1,south2,phi,Y,Fn);
phiHRSs = phi_HRS(i,j,south1,south2,
north1,north2,phi,Y,-Fs);
```

```
        bp = -Fe*(phiHRSe - fe*phi[i][j]) +
        Fw*(phiHRSw - fe*phi[i][j]) -Fn*(phiHRSn - fn*phi[i][j]
        Fs*(phiHRSs - fs*phi[i][j]);


    if (east1 == -1){
        bp = ae*0;
        ae = 0;
    }


    if (west1 == -1){
        bp = aw*0;
        aw=0;
    }

    if (north1 == -1) {
        bp = an*0;
        an=0;
    }

    if (south1 == -1) {
        bp = as*0;
        as = 0;
    }

    }
    //ap = Lambda[0]/rho_0*( dEy/dPEx + dWy/dPWx
    + dNx/dPNy + dSx/dPSy ) + Fe/2 + Fn/2 - Fw/2 - Fs/2;
    //ap = ae + aw + an + as -
    Fe/2 - Fn/2 + Fw/2 + Fs/2;
    //ap = ae + aw + an + as -
    (1-fe)*Fe - (1-fn)*Fn + (1-fw)*Fw + (1-fs)*Fs;

    ap = ae + aw + an + as ; //- Fe - Fn + Fw + Fs;

    new_phi = GaussSeidel2D(aw,ae,an,as,ap,bp,
    phi[west1][j], phi[east1][j], phi[i][north1],
    phi[i][south1]);

    if ( abs(new_phi - phi[i][j]) > criteriConv
    && CondContorn[i][j] == 0) {
        phi[i][j] = new_phi;
```

```
                        //cout << "estoy aqui"<< endl;
                    }

                    if ( j == 0 && CondContorn[i][j] == 0) {
                        phi[i][j] = phi[i][j+1];
                    }


                }
            }

             x++;

            }

}



// MAIN CODE

int main() {

    int i=0; int j=0;

    Geometria2D (L, H, Nx, Ny);
    Limits (L, H, Nx, Ny);
       pasartablaaficheroMalla (CosFluid, Nx, 1);
    CondInicials (Nx, Ny, Mass_0, VelX_0, VelY_0, Temp_0, X , Y);
        pasartablaafichero2D (Temp, Nx, Ny, 4);

        //system("pause");

    // Lambda in Momentum Eq = Viscosity MU
    float  Gamma_Mom [2*dim+1];
    for (i=0;i<(2*dim+1);i++) {
        Gamma_Mom[i] = mu;
    }

    pasartablaafichero2D (VelX, Nx, Ny, 2);
    pasartablaafichero2D (VelY, Nx, Ny, 3);

    float  Gamma_Energy [2*dim+1];
    for (i=0;i<(2*dim+1);i++) {
```

```
        Gamma_Energy[i] = lambda/Cv;
    }

    ConvDiff (Gamma_Energy, S, Temp, 4, T_step, CondContornTemp);

    float Peclet_Energy = rho_0 * VelX_0 * H / (lambda/Cv);
    float GammaRho = rho_0/(lambda/Cv);
    float Peclet = rho_0 * VelX_0 * H / mu;

    double Error [Nx+1];

    cout << "Peclet Energy: " << Peclet_Energy << " ,
    rho/gamma: "<< GammaRho << endl;

    /*
        for (j=0; j<=Ny; j++) {
            for (i=0; i<=Nx; i++) {
                cout << i << " " << j <<  " (" <<
                CondContornTemp[i][j] << ") : " << Temp[i][j] << endl;
            }
        }
    */

    cout << "--- FIN DEL PROGRAMA ---" << endl;

    pasartablaafichero2D (Temp, Nx, Ny, 4);

    return 0;
}

void pasartablaafichero2D (float valor [Nx+1][Ny+1],
int Nx, int Ny, int tipo ) {

     ofstream prod;
        if (tipo == 4){
            prod.open("SH-Temp.dat");
        } if (tipo == 2) {
            prod.open("SH-VelX.dat");
        } if (tipo == 3) {
            prod.open("SH-VelY.dat");
        }
        //prod << "# X     Y      Z" << endl;

        float x = 0; float y = 0;
```

```
        if (prod.is_open()) {
             int i = 0; int j =0;
             for (j=0; j<=Ny; j++) {
                 for (i=0; i<=Nx; i++) {



              //prod << "Coordenadas: ";
              //prod << X2[i][0] <<" " << X2[i][1] << " ";
              //prod << " Cos (1) Fluid (0): " << CosFluid[i] <<endl;
              //prod << X2[i][0] << " " << X2[i][1] << " " << CosFluid[i] <<endl;

                 x = X[i]/H; y = Y[j]/H;

              prod << fixed << setprecision(5) << x << "   ";
              prod << fixed << setprecision(5) << y << "   ";
              prod << fixed << setprecision(5) << valor[i][j] << endl;


             }
             }

     prod.close();
    }
}
```