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

Study for the computational resolution of conservation equations of mass, momentum and energy. Numerical analysis and turbulence models

Document:

Annexes

Author:

Gerard Toset Alonso

Director/Co-director:

Carlos-David Pérez Segarra / Francesc Xavier Trias Miquel

Degree:

Bachelor in Aerospace Vehicle Engineering

Examination session:

Spring 2023.

BACHELOR FINAL THESIS

Contents

Annex A

Heat conduction code

```
//-----------------------------------------------------------//
//                                                           //
//         Study for the computational resolution of         //
//    conservation equations of mass, momentum and energy    //
//         Numerical analysis and turbulence models          //
//                                                           //
//-----------------------------------------------------------//
// Author: Gerard Toset Alonso                               //
// Date: June, 2023                                          //
// Code: Chapter 3 - Conduction heat transfer                //
//-----------------------------------------------------------//
//                   GENERAL DESCRIPTION                     //
// - SOLVER : Gauss-Seidel                                   //
// - Transient regime                                        //
// - 2D spatial discretization - Cell-centered nodes         //
// - NOTE: This code is used to solve the 4-materials        //
//         problem. By changing boundary conditions, the     //
//         other verification cases can also be solved.      //
//-----------------------------------------------------------//
//                                                           //
// VERIFICATION CASES                                        //
//                                                           //
// CASE 1: Only 1 material with lambda=10e8. All the faces   //
// have convection with external fluid at T_g and alfa_g. No //
// volumetric heat generation                                //
//                                                           //
// CASE 2: Only 1 material. Upper and lower faces are        //
// adiabatic. Left wall has convection with T_esq, alfa_esq  //
// and right wall has convection with T_dret, alfa_dret. No  //
// volumetric heat generation                                //
//                                                           //
// CASE 3: Materials 1 & 4. Upper and lower faces are        //
// adiabatic. Left wall has convection with T_esq, alfa_esq  //
// and right wall has convection with T_dret, alfa_dret. No  //
// volumetric heat generation                                //
//                                                           //
// CASE 4: Materials 1 & 4. Upper and lower faces are        //
// adiabatic. Left wall has convection with T_esq, alfa_esq  //
// and right wall has convection with T_dret, alfa_dret.     //
// Volumetric heat generation at material 1.                 //
//                                                           //
//-----------------------------------------------------------//

```

```
#include <iostream>
#include <cmath>
#include <vector>
#include <fstream>
#include <iomanip>
using namespace std;

//--------------------------------------------------------------//
// ------------------ Constant definition ------------------ //
const double pi = M_PI;
const double beta = 1.0;    // 0:explicit - 1:implicit - 0.5:Crank-Nicolson
const int N = 150;                          // Number of CVs in X direction
const int M = 150;                          // Number of CVs in Y direction
const int nmax = N+2;
const int mmax = M+2;
const double delta = 1e-6;                  // Convergence parameter

const double T_esq = 33.0 + 273.0;
const double T_dret = 10.0 + 273.0;
const double T_inf = 23.0 + 273.0;
const double T_sup = 130.0 + 273.0;
const double alfa_esq = 9.0;
const double alfa_dret = 3.0;
const double alfa_inf = 0.0;
const double alfa_sup = 0.0;
const double q_w = 54.55;

typedef vector<double> vect;

//--------------------------------------------------------------//
// ----------------- Functions definition ------------------ //
void geometria(double L, double H, double& Delta_X, double& Delta_Y, vector<double
    > &r_X, vector<double> &r_Y, vector<double> &X_VC, vector<double> &Y_VC);
void blockoff(double H, vector<double> &r_X, vector<double> &r_Y, vector<vector<
    double> > &MAT, vector<vector<double> > &dens, vector<vector<double> > &cp,
    vector<vector<double> > &lambda, double propietats[4][3]);
void harm_mean(vector<double> &r_X, vector<double> &X_VC, vector<double> &r_Y,
    vector<double> &Y_VC, vector<vector<double> > &lambda, vector<vector<double> >
    &lambda_East, vector<vector<double> > &lambda_West, vector<vector<double> > &
    lambda_North, vector<vector<double> > &lambda_South);
void temperature(vector<vector<double> > &T_nmu, vector<vector<double> > &T_est,
    vector<vector<double> > &aE, vector<vector<double> > &aW, vector<vector<double>
     > &aN, vector<vector<double> > &aS, vector<vector<double> > &aP, vector<vector
    <double> > &bP);
void convergence(vector<vector<double> > &T_est, vector<vector<double> > &T_nmu,
    double& max_dif);
void coeff_a(double W, vector<vector<double> > &dens, vector<vector<double> > &cp,
     double Delta_t, vector<double> &r_X, vector<double> &r_Y, double Delta_X,
    double Delta_Y, vector<vector<double> > &lambda_East, vector<vector<double> > &
    lambda_West, vector<vector<double> > &lambda_North, vector<vector<double> > &
    lambda_South, vector<vector<double> > &aE, vector<vector<double> > &aW, vector<
    vector<double> > &aN, vector<vector<double> > &aS, vector<vector<double> > &aP)
    ;
void coeff_b(vector<vector<double> > &bP, double t, double W, vector<vector<double
    > > &dens, vector<vector<double> > &cp, double Delta_t, vector<double> &r_X,
    vector<double> &r_Y, double Delta_X, double Delta_Y, vector<vector<double> > &
    lambda_East, vector<vector<double> > &lambda_West, vector<vector<double> > &
    lambda_North, vector<vector<double> > &lambda_South, vector<vector<double> > &T
    , vector<vector<double> > &q_VP, vector<vector<double> > &q_VPnmu);
void update(vector<vector<double> > &new_val, vector<vector<double> > &
    previous_val);

//--------------------------------------------------------------//
```

```
// --------------------------- MAIN -------------------------- //
int main (){

        ofstream outg ("geometry.txt");
        ofstream outf ("Temperatures.txt");

        //--------------------------------------------------------//
        // I - DATA INPUT:
        // ---------------------- GEOMETRY ---------------------- //
        double P1 [1][2] = {0.5 , 0.3};
        double P2 [1][2] = {0.5 , 0.7};
        double P3 [1][2] = {1.1 , 0.8};         //m - Material coordinates

        double L = P3[0][0];                     //m - Length
        double H = P3[0][1];                     //m - Height
        double W = 20.0;                         //m - Widht
        double Delta_X;
        double Delta_Y;

        // ------------------ THERMOPYSICAL -------------------- //

        double T_inici=10000;
        double T_0 = 8.0 + 273;                   //Initial temperature map at t=0s
        double max_dif = 1000.0;
        double Delta_t = 0.0;                     //Time step
        double t = 0.0;                           //Time variable

        // Thermo-physical properties of the four materials
        //   RHO (kg/m3)          C_P (J/KgK)      LAMBDA (W/mK)
        //   dens_m1=1500;        cp_m1=750;       lambda_m1=170;
        //   dens_m2=1600;        cp_m2=770;       lambda_m2=140;
        //   dens_m3=1900;        cp_m3=810;       lambda_m3=200;
        //   dens_m4=2500;        cp_m4=930;       lambda_m4=140;

        double propietats [4][3] =  {{1500.0 , 750.0 , 170.0} , {1600.0 , 770.0 ,
            140.0} , {1900.0 , 810.0 , 200.0} , {2500.0 , 930.0 , 140.0}};

        //--------------------------------------------------------//
        // II - PREVIOUS CALCULATIONS AND VECTOR DEFINITIONS:

        // ---------------------- VECTORS ---------------------- //
        // Geometry vectors
        vector<double> X_VC(N+1), Y_VC(M+1);
        vector<double> r_X(N+2), r_Y(M+2);
        double d_PX = 0.0;
        double d_XE = 0.0;                        double d_PE = 0.0;
        double d_XW = 0.0;                        double d_PW = 0.0;
        double d_XN = 0.0;                        double d_PN = 0.0;
        double d_XS = 0.0;                        double d_PS = 0.0;
        double S_W = 0.0;                         double S_E = 0.0;
        double S_N = 0.0;                         double S_S = 0.0;
        double V_P = 0.0;

        //Temperature matrices
        vector<vector<double> > T(M+2,vector<double>(N+2));      // T^n
        vector<vector<double> > T_nmu(M+2,vector<double>(N+2)); // T^n+1
        vector<vector<double> > T_est(M+2,vector<double>(N+2)); // T_est^n+1
        //Discretization coefficients
        vector<vector<double> > aP(M+2,vector<double>(N+2));
        vector<vector<double> > bP(M+2,vector<double>(N+2));
        vector<vector<double> > aE(M+2,vector<double>(N+2));
        vector<vector<double> > aN(M+2,vector<double>(N+2));
        vector<vector<double> > aW(M+2,vector<double>(N+2));
```

```
        vector<vector<double> > aS(M+2,vector<double>(N+2));
        //Thermal conductivity
        vector<vector<double> > lambda_East(M+2,vector<double>(N+2));
        vector<vector<double> > lambda_North(M+2,vector<double>(N+2));
        vector<vector<double> > lambda_West(M+2,vector<double>(N+2));
        vector<vector<double> > lambda_South(M+2,vector<double>(N+2));
        //Volumetric heat generation
        vector<vector<double> > q_VP(M+2,vector<double>(N+2));
        vector<vector<double> > Q_P(M+2,vector<double>(N+2));     //q_VP^n
        vector<vector<double> > q_VPnmu(M+2,vector<double>(N+2)); //q_VP^n+1
        //Thermophysical properties
        vector<vector<double> > dens(M+2,vector<double>(N+2));
        vector<vector<double> > lambda(M+2,vector<double>(N+2));
        vector<vector<double> > cp(M+2,vector<double>(N+2));
        vector<vector<double> > MAT(M+2,vector<double>(N+2));

        // ----------------- PREVIOUS CALCULATIONS ------------------ //
        // Null-value initialization
        for (int f=0; f<M+2; f++){
                for (int c=0; c<N+2; c++){
                        T[f][c] = 0.0;
                        T_est[f][c] = 0.0;
                        T_nmu[f][c] = 0.0;
                        aP[f][c] = 0.0;
                        bP[f][c] = 0.0;
                        aE[f][c] = 0.0;
                        aW[f][c] = 0.0;
                        aN[f][c] = 0.0;
                        aS[f][c] = 0.0;
                        lambda_West[f][c] = 0.0;
                        lambda_East[f][c] = 0.0;
                        lambda_North[f][c] = 0.0;
                        lambda_South[f][c] = 0.0;
                        q_VP[f][c] = 0.0;
                        Q_P[f][c] = 0.0;
                        q_VPnmu[f][c] = 0.0;
                 }
        }

        // Mesh generation
        geometria( L, H, Delta_X, Delta_Y, r_X, r_Y, X_VC, Y_VC);
        // Block_off method
        blockoff(H, r_X, r_Y, MAT, dens, cp, lambda, propietats);
        // Harmonic Mean
        harm_mean(r_X, X_VC, r_Y, Y_VC, lambda, lambda_East, lambda_West,
            lambda_North, lambda_South);
        Delta_t = 1;
        // Constant coefficient evaluation
        coeff_a(W, dens, cp, Delta_t, r_X, r_Y, Delta_X, Delta_Y, lambda_East,
            lambda_West, lambda_North, lambda_South, aE, aW, aN, aS, aP);

        //-------------------------------------------------------//
        // III - ARBITRARY INITIAL MAP at t=0:
        for (int f=0; f<M+2; f++){
            int c=0;
            T[f][c] = T_inf;
        }
        for (int f=0; f<M+2; f++){
            for (int c=1; c<N+2; c++){
                    T_est[f][c] = T_inici;
                    T[f][c] = T_0;
            }
        }
```

```
        //----------------------------------------------------------//
        // IV - TEMPORAL COMPUTATION:
        while (t<5000){
                // q_VP^n update from the previous time instant
                update(q_VP, q_VPnmu);
                // New volumetric heat generation computation q_VP^n+1=f(x,y,T_est
                    ^n+1)
                for (int f=0; f<M+2; f++){
                    for (int c=0; c<N+2; c++){
                        q_VPnmu[f][c] = 0.0;
                    }
                }
                // bP evaluation
                coeff_b(bP, t, W, dens, cp, Delta_t, r_X, r_Y, Delta_X, Delta_Y,
                    lambda_East, lambda_West, lambda_North, lambda_South, T, q_VP,
                    q_VPnmu);

                while (max_dif>delta){
                    //----------------------------------------------------------//
                    // V - TEMPERATURE COMPUTATION:
                        temperature( T_nmu, T_est, aE, aW, aN, aS, aP, bP);

                    //----------------------------------------------------------//
                    // VI - CONVERGENCE EVALUATION:
                        convergence(T_est, T_nmu, max_dif);

                    //----------------------------------------------------------//
                    // VII - DATA UPDATE:
                    update(T_est, T_nmu);
                }

                max_dif = 10000.0;
                t = t + Delta_t;

                // DATA UPDATE FOR THE NEXT TIME INSTANT
                update(T_est, T_nmu);
                update(T, T_nmu);

                cout<<t<<endl;

        }

        double T_final [M+2][N+2];

        for (int f=0; f<M+2; f++){
                for(int c=0; c<N+2; c++){
                        T_final[f][c]=T_nmu[M+1-f][c];
                }
        }

        //----------------------------------------------------------//
        // ------------------- OUTPUT FILES -------------------- //
        for (int f=0; f<M+2; f++){
                outg<<r_X[f]<<"\t";
        }
        outg<<endl;
        for (int c=0; c<N+2; c++){
                outg<<r_Y[c]<<"\t";
        }
        outg<<endl;

        for (int c=0; c<N+2; c++){
```

```
			for (int f=0; f<M+2; f++){
				outf<<T_final[c][f]<<"\t";
			}
			outf<<endl;
		}
		//-------------------------------------------------------//
		//-------------------------------------------------------//

}


//----------------------------------------------------------------------//
// -------------------------- MAIN FUNCTIONS ------------------------- //
void geometria(double L, double H, double& Delta_X, double& Delta_Y, vector<double
    > &r_X, vector<double> &r_Y, vector<double> &X_VC, vector<double> &Y_VC)
{
		//-----------------------------------------------------------//
		// A - CV position
		Delta_X = L/N;			Delta_Y = H/M;

		for (int c=0; c<N+1; c++){
			X_VC[c] = Delta_X*c;
		}
		for (int f=0; f<M+1; f++){
			Y_VC[f] = Delta_Y*f;
		}
		//-----------------------------------------------------------//
		// B - Nodes position
		r_X[0] = 0.0;				r_Y[0] = 0.0;
		for (int c=1; c<N+1; c++){
			r_X[c] = ( X_VC[c-1]+X_VC[c] )/2;
		}
		for (int f=1; f<M+1; f++){
			r_Y[f] = ( Y_VC[f-1]+Y_VC[f] )/2;
		}
		r_X[N+1] = L;			r_Y[M+1] = H;
		//-----------------------------------------------------------//
}

void blockoff(double H, vector<double> &r_X, vector<double> &r_Y, vector<vector<
    double> > &MAT, vector<vector<double> > &dens, vector<vector<double> > &cp,
    vector<vector<double> > &lambda, double propietats[4][3])
{
		//-----------------------------------------------------------//
		for (int f=0; f<M+2; f++){
			for(int c=0; c<N+2; c++){
				if(r_X[c]<0.5){  // MATERIALS 1 - 3
					if(r_Y[f]>(H-0.3)){ // Material 3
					    MAT[f][c] = 1;
					    dens[f][c] = propietats[0][0];
					    cp[f][c] = propietats[0][1];
					    lambda[f][c] = propietats[0][2];
					}
					else{   // Material 1
					    MAT[f][c] = 3;
					    dens[f][c] = propietats[2][0];
					    cp[f][c] = propietats[2][1];
					    lambda[f][c] = propietats[2][2];
					}
				}
				else{    // MATERIALS 2 - 4
					if(r_Y[f]>(H-0.7)){ // Material 4
					    MAT[f][c] = 2;
```

```
                                    dens[f][c] = propietats[1][0];
                                    cp[f][c] = propietats[1][1];
                                    lambda[f][c] = propietats[1][2];
                                }
                                else{   // Material 2
                                    MAT[f][c] = 4;
                                    dens[f][c] = propietats[3][0];
                                    cp[f][c] = propietats[3][1];
                                    lambda[f][c] = propietats[3][2];
                                }
                        }
                }
        }
        //-----------------------------------------------------------//

}

void harm_mean(vector<double> &r_X, vector<double> &X_VC, vector<double> &r_Y,
    vector<double> &Y_VC, vector<vector<double> > &lambda, vector<vector<double> >
    &lambda_East, vector<vector<double> > &lambda_West, vector<vector<double> > &
    lambda_North, vector<vector<double> > &lambda_South)
{
        int f, c;
        double d_PX = 0.0;
        double d_XE = 0.0;                      double d_PE = 0.0;
        double d_XW = 0.0;                      double d_PW = 0.0;
        double d_XN = 0.0;                      double d_PN = 0.0;
        double d_XS = 0.0;                      double d_PS = 0.0;
        //------------------------------------------------------------//
        //      HARMONIC MEAN COMPUTATION FOR EACH FACE OF THE CV       //
        //-------------  EAST  -------------//
        for (f=0; f<M+2; f++){
            for (c=0; c<N+2; c++){
                if ( c == 0 || c == (N+1) ){
                    lambda_East[f][c] = lambda[f][c];
                }
                else{
                    d_PX = X_VC[c] - r_X[c];
                    d_XE = r_X[c+1] - X_VC[c];
                    d_PE = d_PX + d_XE;
                    lambda_East[f][c] = d_PE / (  ( d_PX/lambda[f][c] )+( d_XE/
                        lambda[f][c+1] )  );
                }
            }
        }
    //-------------  SOUTH  ------------//
    for (c=0; c<N+2; c++){
        for (f=0; f<M+2; f++){
                if (f == 0 || f == M+1){
                        lambda_South[f][c] = lambda[f][c];
                        }
                        else{
                                d_PX = Y_VC[f] - r_Y[f];
                                d_XS = r_Y[f+1] - Y_VC[f];
                                d_PS = d_PX + d_XS;
                                lambda_South[f][c] = d_PS / (  ( d_PX/lambda[f][c]
                                    )+( d_XS/lambda[f+1][c] )  );
                        }
                }
    }
    //-------------  WEST  -------------//
    for (f=0; f<M+2; f++){
        for (c=0; c<N+2; c++){
```

```
                if ( c == 0 || c == (N+1) ){
                        lambda_West[f][c] = lambda[f][c];
                        }
                        else{
                                d_PX = r_X[c] - X_VC[c-1];
                                d_XW = X_VC[c] - r_X[c-1];
                                d_PW = d_PX + d_XW;
                                lambda_West[f][c] = d_PW / (  ( d_PX/lambda[f][c]
                                    )+( d_XW/lambda[f][c-1] )  );
                        }
                }
    }
    //-------------  NORTH  ------------//
    for (c=0; c<N+2; c++){
        for (f=0; f<M+2; f++){
                if (f == 0 || f == M+1){
                        lambda_North[f][c] = lambda[f][c];
                        }
                        else{
                                d_PX = r_Y[f] - Y_VC[f-1];
                                d_XN = Y_VC[f] - r_Y[f-1];
                                d_PN = d_PX + d_XN;
                                lambda_North[f][c] = d_PN / (  ( d_PX/lambda[f][c]
                                     )+( d_XN/lambda[f-1][c] )  );
                        }
                }
    }
}

void coeff_a(double W, vector<vector<double> > &dens, vector<vector<double> > &cp,
     double Delta_t, vector<double> &r_X, vector<double> &r_Y, double Delta_X,
    double Delta_Y, vector<vector<double> > &lambda_East, vector<vector<double> > &
    lambda_West, vector<vector<double> > &lambda_North, vector<vector<double> > &
    lambda_South, vector<vector<double> > &aE, vector<vector<double> > &aW, vector<
    vector<double> > &aN, vector<vector<double> > &aS, vector<vector<double> > &aP)
{
        int f, c;
        double d_PE = 0.0;                      double d_PW = 0.0;
        double d_PN = 0.0;                      double d_PS = 0.0;
        double S_E = 0.0;                       double S_W = 0.0;
        double S_N = 0.0;                       double S_S = 0.0;
        double V_P = 0.0;

        // LEFT WALL:  c=0 f=0 a M+1
        for (f=0; f<M+2; f++){
            d_PE = r_X[1]-r_X[0];
            S_E = Delta_Y*W;
            aW[f][0] = 0;
            aE[f][0] = (lambda_East[f][0]*S_E*beta)/d_PE;
            aS[f][0] = 0;
            aN[f][0] = 0;
            aP[f][0] = alfa_esq*S_E*beta + (lambda_East[f][0]*S_E*beta)/d_PE;
        }
        // RIGHT WALL: c=N+1 f=0 a M+1
        for (f=0; f<M+2; f++){
            aW[f][N+1] = 0;
            aE[f][N+1] = 0;
            aS[f][N+1] = 0;
            aN[f][N+1] = 0;
            aP[f][N+1] = 1;
        }
        // UPPER WALL: c=0 a N+1 f=0
        for (c=0; c<N+2; c++){
```

```
            d_PN = r_Y[1]-r_Y[0];
            S_N = Delta_X*W;
            aW[0][c] = 0;
            aE[0][c] = 0;
            aS[0][c] = 0;
            aN[0][c] = (lambda_North[0][c]*S_N*beta)/d_PN;
            aP[0][c] = (lambda_North[0][c]*S_N*beta)/d_PN;
        }
        // LOWER WALL: c=0 a N+1 f=M+1
        for (c=0; c<N+2; c++){
            aW[M+1][c] = 0;
            aE[M+1][c] = 0;
            aS[M+1][c] = 0;
            aN[M+1][c] = 0;
            aP[M+1][c] = 1;
        }
        // INNER NODES: f=1 a M - c=1 a N
        for (f=1; f<M+1; f++){
                for (c=1; c<N+1; c++){
                d_PW = r_X[c]-r_X[c-1];
                d_PE = r_X[c+1]-r_X[c];
                d_PS = r_Y[f+1]-r_Y[f];
                d_PN = r_Y[f]-r_Y[f-1];

                S_W = Delta_Y*W;
                S_E = Delta_Y*W;
                S_N = Delta_X*W;
                S_S = Delta_Y*W;
                V_P = (S_E*S_N)/W;

                aW[f][c] = (lambda_West[f][c]*S_W*beta)/d_PW;
                aE[f][c] = (lambda_East[f][c]*S_E*beta)/d_PE;
                aS[f][c] = (lambda_South[f][c]*S_S*beta)/d_PS;
                aN[f][c] = (lambda_North[f][c]*S_N*beta)/d_PN;
                aP[f][c] = aW[f][c]+aE[f][c]+aS[f][c]+aN[f][c]+((dens[f][c]*V_P*cp
                    [f][c])/Delta_t);
                }
        }
        //-----------------------------------------------------------//
}

void coeff_b(vector<vector<double> > &bP, double t, double W, vector<vector<double
    > > &dens, vector<vector<double> > &cp, double Delta_t, vector<double> &r_X,
    vector<double> &r_Y, double Delta_X, double Delta_Y, vector<vector<double> > &
    lambda_East, vector<vector<double> > &lambda_West, vector<vector<double> > &
    lambda_North, vector<vector<double> > &lambda_South, vector<vector<double> > &T
    , vector<vector<double> > &q_VP, vector<vector<double> > &q_VPnmu)
{
        int f, c;
        double d_PE = 0.0;                      double d_PW = 0.0;
        double d_PN = 0.0;                      double d_PS = 0.0;
        double S_E = 0.0;                       double S_W = 0.0;
        double S_N = 0.0;                       double S_S = 0.0;
        double V_P = 0.0;
        double Q_P[mmax][nmax];

        // LEFT WALL:           c=0                     f=0 a M+1
        for (f=0; f<M+2; f++){
                d_PE = r_X[1]-r_X[0];
                S_E = Delta_Y*W;
                Q_P[f][0] = alfa_esq*(T_esq - T[f][0])*S_E + (lambda_East[f][0]*
                    S_E*(T[f][1] - T[f][0]))/d_PE;

```

```
                bP[f][0] = alfa_esq*S_E*beta*T_esq + (1-beta)*Q_P[f][0];
        }
        // RIGHT WALL:           c=N+1           f=0 a M+1
        for (f=0; f<M+2; f++){
                bP[f][N+1] = (8 + 0.005*t)+273;
        }
        // UPPER WALL:           c=0 a N+1       f=0
        for (c=0; c<N+2; c++){
                d_PN = r_Y[1]-r_Y[0];
                S_N = Delta_X*W;
                Q_P[0][c] = -q_w*S_N - (lambda_North[0][c]*S_N*(T[1][c] - T[0][c])
                    )/d_PN;

                bP[0][c] = -q_w*S_N*beta + (1-beta)*Q_P[0][c];
        }
        // LOWER WALL:           c=0 a N+1       f=M+1
        for (c=0; c<N+2; c++){
                bP[M+1][c] = T_inf;
        }
        // INNER NODES:          f=1 a M  -  c=1 a N
        for (f=1; f<M+1; f++){
                for (c=1; c<N+1; c++){
                        d_PW = r_X[c]-r_X[c-1];
                d_PE = r_X[c+1]-r_X[c];
                d_PS = r_Y[f+1]-r_Y[f];
                d_PN = r_Y[f]-r_Y[f-1];

                        S_W = Delta_Y*W;                S_E = Delta_Y*W;
                S_N = Delta_X*W;                S_S = Delta_Y*W;
                V_P = (S_E*S_N)/W;
                Q_P[f][c] = - (lambda_North[f][c]*S_N*(T[f][c] - T[f-1][c]))/d_PN
                    - (lambda_West[f][c]*S_W*(T[f][c] - T[f][c-1]))/d_PW + q_VP[f][
                    c]*V_P + (lambda_South[f][c]*S_S*(T[f+1][c] - T[f][c]))/d_PS +
                    (lambda_East[f][c]*S_E*(T[f][c+1] - T[f][c]))/d_PE ;

                bP[f][c] = q_VPnmu[f][c]*V_P*beta + ((dens[f][c]*V_P*cp[f][c])/
                    Delta_t)*T[f][c] + (1-beta)*Q_P[f][c];
                }
        }
}

void temperature(vector<vector<double> > &T_nmu, vector<vector<double> > &T_est,
    vector<vector<double> > &aE, vector<vector<double> > &aW, vector<vector<double>
     > &aN, vector<vector<double> > &aS, vector<vector<double> > &aP, vector<vector
    <double> > &bP)
{
        int c, f;

        for (c=1; c<N+1; c++){
                f=0;
                T_nmu[f][c] = ( (aE[f][c]*T_est[f][c+1]) + (aW[f][c]*T_est[f][c
                    -1]) + (aN[f][c]*T_est[f+1][c]) + (bP[f][c]) ) / (aP[f][c]);
        }
        for (c=1; c<N+1; c++){
                f=M+1;
                T_nmu[f][c] = ( (aE[f][c]*T_est[f][c+1]) + (aW[f][c]*T_est[f][c
                    -1]) + (aS[f][c]*T_est[f-1][c]) + (bP[f][c]) ) / (aP[f][c]);
        }
        for (f=1; f<M+1; f++){
                c=0;
                T_nmu[f][c] = ( (aE[f][c]*T_est[f][c+1]) + (aN[f][c]*T_est[f+1][c
                    ]) + (aS[f][c]*T_est[f-1][c]) + (bP[f][c]) ) / (aP[f][c]);
        }
```

```
        for (f=1; f<M+1; f++){
                c=N+1;
                T_nmu[f][c] = ( (aW[f][c]*T_est[f][c-1]) + (aN[f][c]*T_est[f+1][c
                    ]) + (aS[f][c]*T_est[f-1][c]) + (bP[f][c]) ) / (aP[f][c]);
        }
        for (f=1; f<M+1; f++){
                for (c=1; c<M+1; c++){
                        T_nmu[f][c] = ( (aE[f][c]*T_est[f][c+1]) + (aW[f][c]*T_est
                            [f][c-1]) + (aN[f][c]*T_est[f+1][c]) + (aS[f][c]*T_est[
                            f-1][c]) + (bP[f][c]) ) / (aP[f][c]);
                }
        }
        // Arbitary temperature value assignation for the corner nodes
        // UPPER LEFT  LOWER LEFT  UPPER RIGHT LOWER RIGHT
        T_nmu[0][0] = T_nmu[1][0];
        T_nmu[M+1][0] = T_nmu[M][0];
        T_nmu[0][N+1] = T_nmu[1][N+1];
        T_nmu[M+1][N+1] = T_nmu[M][N+1];
}

void convergence(vector<vector<double> > &T_est, vector<vector<double> > &T_nmu,
    double& max_dif)
{
        int f, c;
        double dif, maxim = 0.0;
        // Maximum difference between estimated and next computed temperature
        for (f=0; f<M+2; f++){
                for (c=0; c<N+2; c++){
                        dif = T_est[f][c]-T_nmu[f][c];
                        if (dif>=0){
                                if (dif>=maxim){
                        maxim = dif;
                                }
                                else{
                                        maxim = maxim;
                                }
                        }
                        else{
                                dif = dif * -1;
                                if (dif>=maxim){
                                maxim = dif;
                                }
                                else{
                                        maxim = maxim;
                                }
                        }
                }
        }
        max_dif = maxim;
}

void update(vector<vector<double> > &new_val, vector<vector<double> > &
    previous_val)
{
        int f, c;
        for (f=0; f<M+2; f++){
                for (c=0; c<N+2; c++){
                        new_val[f][c] = previous_val[f][c];
                }
        }
}
```

Annex B

Convection-Diffusion code - CDS, UDS, and EDS

```
//-----------------------------------------------------------//
//                                                           //
//        Study for the computational resolution of          //
//    conservation equations of mass, momentum and energy    //
//        Numerical analysis and turbulence models           //
//                                                           //
//-----------------------------------------------------------//
// Author: Gerard Toset Alonso                               //
// Date: June, 2023                                          //
// Code: Chapter 4 - Convection-Diffusion CDS, UDS, EDS      //
//-----------------------------------------------------------//
//                  GENERAL DESCRIPTION                      //
// - SOLVER : Gauss-Seidel                                   //
// - 2D spatial discretization - Cell-centered nodes         //
// - NOTE: This code is used to solve the general convection //
//        diffusion equation. By changing CASE and SCHEME    //
//        this code can provide solutions for the four       //
//        benchmark problems with CDS, UDS and EDS.          //
//-----------------------------------------------------------//

#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>
#include <math.h>
#include <iomanip>

using namespace std;

//-----------------------------------------------------------//
// ----------------- Constant definition ----------------- //
const int N = 100;           // Number of CVs in X direction
const int M = 100;           // Number of CVs in Y direction
const int nmax = N+2;
const int mmax = M+2;
const int L = 1;
const int H = 1;
const int W = 1;

```

```
const double pi = M_PI;
const double delta_t = 0.001;   // Time step

//-----------------------------------------------------------//
// ----------------- Functions definition ------------------- //
void mesh_generation(double &delta_X, double &delta_Y, int CASE, vector<double> &x_VC, vector<double> &y_VC, vector<double> &x_N, vector<double> &y_N, vector<vector<double> > &V_VC);
void velocity_field(int CASE, double v_0, vector<vector<double> > &vel_X, vector<vector<double> > &vel_Y, vector<double> &x_N, vector<double> &y_N);
void update_value(vector<vector<double> > &matriu, double value);
void update_matrix(vector<vector<double> > &new_matrix, vector<vector<double> > &previous_matrix);
void coeff_a(int scheme, int CASE, double rho, double delta_X, double delta_Y, double Gamma, vector<vector<double> > &vel_X, vector<vector<double> > &vel_Y, vector<vector<double> > &V_VC, vector<vector<double> > &aE, vector<vector<double> > &aW, vector<vector<double> > &aN, vector<vector<double> > &aS, vector<vector<double> > &aP);
void coeff_b(double rho, vector<vector<double> > &V_VC, vector<vector<double> > &Fi_0, vector<vector<double> > &bP);
void resolution(int CASE, vector<double> &x_N, vector<vector<double> > &Fi, vector<vector<double> > &Fi_est, vector<vector<double> > &aP, vector<vector<double> > &aE, vector<vector<double> > &aW, vector<vector<double> > &aN, vector<vector<double> > &aS, vector<vector<double> > &bP);
void convergence(vector<vector<double> > &Fi, vector<vector<double> > &Fi_est, double &maxim);

ofstream outg ("geom_S_E_Peinf.txt");
ofstream outf ("fi_S_E_Peinf.txt");

//-----------------------------------------------------------//
// -------------------------- MAIN ------------------------- //
int main(){

        //-------------------------------------------------------//
        // I - DATA INPUT:
        // -------------------- NUMERICAL ---------------------- //
        double t = 0.0;
        double t_final = 10;
        double maxim;
        double delta = 10e-6;          // Convergence parameter
        double v_0 = 1.0;
        double Gamma = 1.0;            // Diffusion parameter
        double rho = 1000000.0;
        double Fi_initial = 1.0;
        double delta_X = L/double(N);
        double delta_Y = H/double(M);
        int i, j;                      // Geometry parameters

        int CASE;
        // CASE 1 = PARALLEL FLOW
        // CASE 2 = PERPENDICULAR FLOW
        // CASE 3 = DIAGONAL FLOW
        // CASE 4 = SMITH-HUTTON
        int scheme;
        // scheme 1 = CDS
        // scheme 2 = EDS
        // scheme 3 = UDS
        //-------------------------------------------------------//
        // II - PREVIOUS CALCULATIONS AND VECTOR DEFINITIONS:
        // ---------------------- VECTORS ---------------------- //
        // Geometry vectors
        vector<double> x_VC(N+1), y_VC(M+1);
```

```
	vector<double> x_N(nmax), y_N(mmax);
	vector<vector<double> > V_VC(nmax, vector<double> (mmax));
	// Velocity vectors
	vector<vector<double> > vel_X(nmax, vector<double> (mmax)), vel_Y(nmax, vector<double> (mmax));
	// Discretization coefficients
	vector<vector<double> > aP(nmax, vector<double> (mmax)), aE(nmax, vector<double> (mmax)), aW(nmax, vector<double> (mmax)), aN(nmax, vector<double> (mmax)), aS(nmax, vector<double> (mmax)), bP(nmax, vector<double> (mmax));
	// Generic variable vectors
	vector<vector<double> > Fi_0(nmax, vector<double> (mmax)); // Generic variable t^n
	vector<vector<double> > Fi(nmax, vector<double> (mmax)); // Generic variable t^n+1
	vector<vector<double> > Fi_est(nmax, vector<double> (mmax)); // Generic variable ESTIMATED
	// ----------------- PREVIOUS CALCULATIONS ------------------ //
	// Null-value initialization
	double null_val = 0.0;
	update_value(aP, null_val);
	update_value(aE, null_val);
	update_value(aW, null_val);
	update_value(aN, null_val);
	update_value(aS, null_val);
	update_value(bP, null_val);

	// CASE and SCHEME choice
	cout<<"CASE:␣";
	cin>>CASE;
	cout<<"SCHEME:␣";
	cin>>scheme;

	// Mesh generation as a function of the CASE
	mesh_generation(delta_X, delta_Y, CASE, x_VC, y_VC, x_N, y_N, V_VC);
	// Velocity field as a function of the CASE
	velocity_field(CASE, v_0, vel_X, vel_Y, x_N, y_N);
	// Generic variable initial map
	update_value(Fi, Fi_initial);
	// Constant coefficient evaluation as a function of the CASE and SCHEME
	coeff_a(scheme, CASE, rho, delta_X, delta_Y, Gamma, vel_X, vel_Y, V_VC, aE, aW, aN, aS, aP);

	//-------------------------------------------------------//
	// III - TEMPORAL COMPUTATION:
	while(t<t_final){

		// Generic variable update from the previous time instant
		update_matrix(Fi_0, Fi);
		maxim = 1e4;

		// Gauss-Seidel solver with convercence parameter
		while(maxim>delta){
			// Estimated generic variable update
			update_matrix(Fi_est, Fi);
			// bP evaluation
			coeff_b(rho, V_VC, Fi_0, bP);
			// Generic variable solver as a function of the CASE
			resolution(CASE, x_N, Fi, Fi_est, aP, aE, aW, aN, aS, bP);
			// Convergence evaluation
			convergence(Fi, Fi_est, maxim);
		}
		cout<<t<<endl;
```

```
			t = t + delta_t;
		}
		//------------------------------------------------------//
		// ------------------ OUTPUT FILES -------------------- //
		for(i=0; i<N+2; i++){
			for(j=0; j<M+2; j++){
				cout<<Fi[i][j]<<"   ";
			}
			cout<<endl;
		}

		for (i=0; i<N+2; i++){
			outg<<x_N[i]<<"\t";
		}
		outg<<endl;
		for (j=0; j<M+2; j++){
			outg<<y_N[j]<<"\t";
		}
		outg<<endl;

		for (i=0; i<N+2; i++){
			for (j=0; j<M+2; j++){
				outf<<Fi[j][i]<<"\t";
			}
			outf<<endl;
		}
		//------------------------------------------------------//
		//------------------------------------------------------//
}

//----------------------------------------------------------------------//
// -------------------------- MAIN FUNCTIONS ------------------------- //
void mesh_generation(double &delta_X, double &delta_Y, int CASE, vector<double> &x_VC, vector<double> &y_VC, vector<double> &x_N, vector<double> &y_N, vector<vector<double> > &V_VC)
{
		int i, j;
		// For CASE 1 - 2 - 3, geometry is the same since the three problems are a square cavity
		// For CASE 4, geometry changes since the problem becomes a rectangular cavity

		switch (CASE){
			case 1:
			case 2:
			case 3:
			//----------------------------------------------------------//
			// A - CV position
			for(i=0; i<N+1; i++){
				x_VC[i] = i*delta_X;
			}
			for(j=0; j<M+1; j++){
				y_VC[j] = j*delta_Y;
			}
			//----------------------------------------------------------//
			//----------------------------------------------------------//
			// B - Nodes position
			x_N[0] = 0.0;   x_N[N+1]=L;
			y_N[0] = 0.0;   y_N[M+1]=H;
			for(i=1; i<N+1; i++){
				x_N[i] = 0.5*(x_VC[i]+x_VC[i-1]);
			}
			for(j=1; j<M+1; j++){
```

```
				y_N[j] = 0.5*(y_VC[j]+y_VC[j-1]);
			}
			//------------------------------------------------------------//
			break;

			case 4:
			delta_X = 2*L/double(N);
			//------------------------------------------------------------//
			// A - CV position
			for(i=0; i<N+1; i++){
				x_VC[i] = (i*delta_X)-L;
			}
			for(j=0; j<M+1; j++){
				y_VC[j] = j*delta_Y;
			}
			//------------------------------------------------------------//
			//------------------------------------------------------------//
			// B - Nodes position
			x_N[0] = -L;		x_N[N+1]=L;
			y_N[0] = 0.0;		y_N[M+1]=H;
			for(i=1; i<N+1; i++){
				x_N[i] = 0.5*(x_VC[i]+x_VC[i-1]);
			}
			for(j=1; j<M+1; j++){
				y_N[j] = 0.5*(y_VC[j]+y_VC[j-1]);
			}
			//------------------------------------------------------------//
			break;
	}

	for(i=0; i<N+2; i++){
		for (j=0; j<M+2; j++){
			V_VC[i][j] = 0.0;
		}
	}
	//------------------------------------------------------------//
	// C - CV volume
	for(i=1; i<N+1; i++){
		for(j=1; j<M+1; j++){
			V_VC[i][j] = delta_X*delta_Y*W;
		}
	}
	//------------------------------------------------------------//
}

void velocity_field(int CASE, double v_0, vector<vector<double> > &vel_X, vector<
    vector<double> > &vel_Y, vector<double> &x_N, vector<double> &y_N)
{

	int i, j;
	double alpha = 0.0;

	switch (CASE){
		case 1:
			alpha = 0.0*pi/180.0;
			for (i=0; i<N+2; i++){
				for (j=0; j<M+2; j++){
					vel_X[i][j] = v_0*cos(alpha);
					vel_Y[i][j] = v_0*sin(alpha);
				}
			}
		break;
		case 2:
```

```
				alpha = 90.0*pi/180.0;
				for (i=0; i<N+2; i++){
					for (j=0; j<M+2; j++){
						vel_X[i][j] = v_0*cos(alpha);
						vel_Y[i][j] = v_0*sin(alpha);
					}
				}
			break;
			case 3:
				alpha = 45.0*pi/180.0;
				for (i=0; i<N+2; i++){
					for (j=0; j<M+2; j++){
						vel_X[i][j] = v_0*cos(alpha);
						vel_Y[i][j] = v_0*sin(alpha);
					}
				}
			break;
			case 4:
				double x;
				for (i=0; i<N+2; i++){
					for (j=0; j<M+2; j++){
						vel_X[i][j] = 2*y_N[j]*(1-pow(x_N[i],2));
						vel_Y[i][j] = -2*x_N[i]*(1-pow(y_N[j],2));
					}
				}
			break;
	}

}

void update_value(vector<vector<double> > &matriu, double value)
{
	int i, j;

	for(i=0; i<N+2; i++){
		for(j=0; j<M+2; j++){
			matriu[i][j] = value;
		}
	}

}

void update_matrix(vector<vector<double> > &new_matrix, vector<vector<double> > &previous_matrix)
{
	int i, j;

	for(i=0; i<N+2; i++){
		for(j=0; j<M+2; j++){
			new_matrix[i][j] = previous_matrix[i][j];
		}
	}
}

void coeff_a(int scheme, int CASE, double rho, double delta_X, double delta_Y, double Gamma, vector<vector<double> > &vel_X, vector<vector<double> > &vel_Y, vector<vector<double> > &V_VC, vector<vector<double> > &aE, vector<vector<double> > &aW, vector<vector<double> > &aN, vector<vector<double> > &aS, vector<vector<double> > &aP)
{
	int i, j;
	double Se, Sw, Sn, Ss;
	// mass fluxes
```

```
        vector<vector<double> > me(nmax, vector<double> (mmax)), mw(nmax, vector<double> (mmax)), mn(nmax, vector<double> (mmax)), ms(nmax, vector<double> (mmax));
        // velocities
        vector<vector<double> > vE(nmax, vector<double> (mmax)), vW(nmax, vector<double> (mmax)), vN(nmax, vector<double> (mmax)), vS(nmax, vector<double> (mmax));
        // diffusive terms
        vector<vector<double> > De(nmax, vector<double> (mmax)), Dw(nmax, vector<double> (mmax)), Dn(nmax, vector<double> (mmax)), Ds(nmax, vector<double> (mmax));
        // f coefficients
        vector<vector<double> > fe(nmax, vector<double> (mmax)), fw(nmax, vector<double> (mmax)), fn(nmax, vector<double> (mmax)), fs(nmax, vector<double> (mmax));
        // f coefficients for EDS
        vector<vector<double> > fe_EDS(nmax, vector<double> (mmax)), fw_EDS(nmax, vector<double> (mmax)), fn_EDS(nmax, vector<double> (mmax)), fs_EDS(nmax, vector<double> (mmax));
        // Peclet numbers
        vector<vector<double> > PEe(nmax, vector<double> (mmax)), PEw(nmax, vector<double> (mmax)), PEn(nmax, vector<double> (mmax)), PEs(nmax, vector<double> (mmax));

        // MASS FLUXES as a function of the CASE
        switch (CASE){
                case 1:
                case 2:
                case 3:
                        for(i=0; i<N+2; i++){
                                for(j=0; j<M+2; j++){
                                        vE[i][j] = vel_X[i][j];  vW[i][j] = vel_X[i][j];
                                        vN[i][j] = vel_Y[i][j];  vS[i][j] = vel_Y[i][j];
                                }
                        }
                break;

                case 4:
                        // Velocity harmonic mean is applied for the CASE 4
                        for(j=0; j<M+1; j++){
                            vE[0][j] = vel_X[0][j];  vW[0][j] = vel_X[0][j];
                                vE[N+1][j] = vel_X[N+1][j];  vW[N+1][j] = vel_X[N+1][j];
                                for (i=1; i<N; i++){
                                        vE[i][j]= delta_X/(((0.5*delta_X)/vel_X[i][j])+((0.5*delta_X)/vel_X[i+1][j]));
                                        vW[i][j]= delta_X/(((0.5*delta_X)/vel_X[i-1][j])+((0.5*delta_X)/vel_X[i][j]));

                                }
                        }
                        for (i=0; i<=N+1; i++){
                                vN[i][0] = vel_Y[i][0];  vS[i][0] = vel_Y[i][0];
                                vN[i][M+1] = vel_Y[i][N+1];  vS[i][N+1] = vel_Y[i][N+1];
                            for(j=1; j<M; j++){
                                   vN[i][j]= delta_Y/(((0.5*delta_Y)/vel_Y[i][j])+((0.5*delta_Y)/vel_Y[i][j+1]));
                                   vS[i][j]= delta_Y/(((0.5*delta_Y)/vel_Y[i][j-1])+((0.5*delta_Y)/vel_Y[i][j]));
                                }
```

```
                }
            break;
    }
    Se = delta_Y*W;
    Sw = delta_Y*W;
    Sn = delta_X*W;
    Ss = delta_X*W;

    for(i=0; i<N+2; i++){
            for(j=0; j<M+2; j++){
                    me[i][j] = rho*Se*vE[i][j];  mw[i][j] = -rho*Sw*vW[i][j];
                    mn[i][j] = rho*Se*vN[i][j];  ms[i][j] = -rho*Sw*vS[i][j];
            }
    }


    // DIFFUSIVE TERMS
    for(i=1; i<N+1; i++){
            for(j=1; j<M+1; j++){
                    De[i][j] = Gamma*Se/delta_X;  Dw[i][j] = Gamma*Sw/delta_X;
                    Dn[i][j] = Gamma*Sn/delta_Y;  Ds[i][j] = Gamma*Ss/delta_Y;
                    if(i==N){
                            De[i][j] = Gamma*Se/(0.5*delta_X);
                    }
                    if(i==1){
                            Dw[i][j] = Gamma*Se/(0.5*delta_X);
                    }
                    if(j==M){
                            Dn[i][j] = Gamma*Sn/(0.5*delta_Y);
                    }
                    if(j==1){
                            Ds[i][j] = Gamma*Ss/(0.5*delta_Y);
                    }
            }
    }

    // DISCRETIZATION COEFFICIENTS as a function of the SCHEME
    switch (scheme){
            case 1:
                    //CDS
                    for(i=1; i<N+1; i++){
                            for(j=1; j<M+1; j++){
                                    fe[i][j] = 0.5*delta_X/delta_X;  fw[i][j]
                                        = 0.5*delta_X/delta_X;
                                    fn[i][j] = 0.5*delta_Y/delta_Y;  fs[i][j]
                                        = 0.5*delta_Y/delta_Y;
                                    if(i==N){
                                            fe[i][j] = 1.0;
                                    }
                                    if(i==1){
                                            fw[i][j] = 1.0;
                                    }
                                    if(j==M){
                                            fn[i][j] = 1.0;
                                    }
                                    if(j==1){
                                            fs[i][j] = 1.0;
                                    }
                            }
                    }

                    for(i=1; i<N+1; i++){
                            for(j=1; j<M+2; j++){
```

```
                    aE[i][j] = De[i][j] - me[i][j]*fe[i][j];        aW[i][j] = Dw[i][j] - mw[i][j]*fw[i][j];
                    aN[i][j] = Dn[i][j] - mn[i][j]*fn[i][j];        aS[i][j] = Ds[i][j] - ms[i][j]*fs[i][j];
                }
            }

        break;
        case 2:
            //EDS
            for(i=1; i<N+1; i++){
                for(j=1; j<M+1; j++){
                    fe[i][j] = 0.5*delta_X/delta_X;  fw[i][j] = 0.5*delta_X/delta_X;
                    fn[i][j] = 0.5*delta_Y/delta_Y;  fs[i][j] = 0.5*delta_Y/delta_Y;
                    if(i==N){
                        fe[i][j] = 1.0;
                    }
                    if(i==1){
                        fw[i][j] = 1.0;
                    }
                    if(j==M){
                        fn[i][j] = 1.0;
                    }
                    if(j==1){
                        fs[i][j] = 1.0;
                    }
                }
            }
            for(i=1; i<N+1; i++){
                for(j=1; j<M+1; j++){
                    PEe[i][j] = rho*vE[i][j]*delta_X/Gamma;
                    PEw[i][j] = rho*vW[i][j]*delta_X/Gamma;
                    PEn[i][j] = rho*vN[i][j]*delta_Y/Gamma;
                    PEs[i][j] = rho*vS[i][j]*delta_Y/Gamma;
                    if(i==N){
                        PEe[i][j] = rho*vE[i][j]*0.5*delta_X/Gamma;
                    }
                    if(i==1){
                        PEw[i][j] = rho*vW[i][j]*0.5*delta_X/Gamma;
                    }
                    if(j==M){
                        PEn[i][j] = rho*vN[i][j]*0.5*delta_Y/Gamma;
                    }
                    if(j==1){
                        PEs[i][j] = rho*vS[i][j]*0.5*delta_Y/Gamma;
                    }

                    fe_EDS[i][j] = ( exp(PEe[i][j]*fe[i][j])-1 )/( exp(PEe[i][j]) -1 );            fw_EDS[i][j] = ( exp(PEw[i][j]*fw[i][j])-1 )/( exp(PEw[i][j]) -1 );
                    fn_EDS[i][j] = ( exp(PEn[i][j]*fn[i][j])-1 )/( exp(PEn[i][j]) -1 );            fs_EDS[i][j] = ( exp(
```

```
PEs[i][j]*fs[i][j])-1 )/( exp(PEs[i][j]) -1 );
                        }
                }

                for(i=1; i<N+1; i++){
                        for(j=1; j<M+2; j++){
                                aE[i][j] = De[i][j] - me[i][j]*fe[i][j];        aW[i][j] = Dw[i][j] - mw[i][j]*fw[i][j];
                                aN[i][j] = Dn[i][j] - mn[i][j]*fn[i][j];        aS[i][j] = Ds[i][j] - ms[i][j]*fs[i][j];
                        }
                }

        break;
        case 3:
                //UDS
                for(i=1; i<N+1; i++){
                        for(j=1; j<M+1; j++){
                                aE[i][j] = De[i][j] - 0.5*(me[i][j] - abs(me[i][j]));
                                aW[i][j] = Dw[i][j] - 0.5*(mw[i][j] - abs(mw[i][j]));
                                aN[i][j] = Dn[i][j] - 0.5*(mn[i][j] - abs(mn[i][j]));
                                aS[i][j] = Ds[i][j] - 0.5*(ms[i][j] - abs(ms[i][j]));
                        }
                }
        break;
    }

    // aP coefficient evaluation
    for(i=1; i<N+1; i++){
        for(j=1; j<M+1; j++){
                aP[i][j] = aE[i][j] + aW[i][j] + aN[i][j] + aS[i][j] + rho*V_VC[i][j]/delta_t;
        }
    }

}

void coeff_b(double rho, vector<vector<double> > &V_VC, vector<vector<double> > &Fi_0, vector<vector<double> > &bP)
{
    int i, j;

    for(i=1; i<N+1; i++){
        for(j=1; j<M+1; j++){
                bP[i][j] = rho*V_VC[i][j]*Fi_0[i][j]/delta_t;
        }
    }
}

void resolution(int CASE, vector<double> &x_N, vector<vector<double> > &Fi, vector<vector<double> > &Fi_est, vector<vector<double> > &aP, vector<vector<double> > &aE, vector<vector<double> > &aW, vector<vector<double> > &aN, vector<vector<double> > &aS, vector<vector<double> > &bP)
{
    int i, j;
    double Fi_in = 1.0, Fi_out = 0.0;
```

```

        // INNNER NODES
        for(i=1; i<N+1; i++){
                for(j=1; j<M+1; j++){
                                Fi[i][j] = ( aE[i][j]*Fi_est[i+1][j] + aW[i][j]*
                                    Fi_est[i-1][j] + aN[i][j]*Fi_est[i][j+1] + aS[i
                                    ][j]*Fi_est[i][j-1] + bP[i][j] ) / aP[i][j];
                }
        }

        // BOUNDARY CONDITIONS as a function of the CASE: FI for wall nodes are
            already known
        switch (CASE){
                case 1:
                case 2:
                        for(i=1; i<N+1; i++){
                                j=0;
                                Fi[i][j] = Fi[i][j+1];
                                j=M+1;
                                Fi[i][j] = Fi[i][j-1];
                        }
                        for(j=1; j<M+1; j++){
                                i=0;
                                Fi[i][j] =      Fi_in;
                                i=N+1;
                                Fi[i][j] =      Fi_out;
                        }
                break;

                case 3:
                        for(i=1; i<N+1; i++){
                                j=0;
                                Fi[i][j] = Fi_in;
                                j=M+1;
                                Fi[i][j] = Fi_out;
                        }
                        for(j=1; j<M+1; j++){
                                i=0;
                                Fi[i][j] =      Fi_out;
                                i=N+1;
                                Fi[i][j] =      Fi_in;
                        }
                break;

                case 4:
                        int half = (N+2)/2;
                        for(i=1; i<=half; i++){
                                j=0;
                                Fi[i][j] =      1+tanh(10*(2*x_N[i] + 1));
                                j=M+1;
                                Fi[i][j] = 1-tanh(10);
                        }
                        for(i=half+1; i<N+1; i++){
                                j=0;
                                Fi[i][j] =      Fi[i][j+1];
                                j=M+1;
                                Fi[i][j] = 1-tanh(10);
                        }
                        for(j=1; j<M+1; j++){
                                i=0;
                                Fi[i][j] = 1-tanh(10);
                                i=N+1;
                                Fi[i][j] = 1-tanh(10);
```

```
                        }
                break;

        }

        // Arbitary temperature value assignation for the corner nodes
        Fi[0][0] = Fi[1][0];                        Fi[0][M+1] = Fi[1][M+1];
        Fi[N+1][0] = Fi[N][0];                      Fi[N+1][M+1] = Fi[N][M+1];

}

void convergence(vector<vector<double> > &Fi, vector<vector<double> > &Fi_est,
    double &maxim)
{
        int i, j;
        double diff, max_aux = 0.0;

        // Maximum difference between estimated and next computed temperature
        for(i=1; i<N+1; i++){
                for(j=1; j<M+1; j++){
                        diff = abs(Fi[i][j] - Fi_est[i][j]);

                        if(diff>=max_aux){
                                max_aux = diff;
                        }

                }
        }

        maxim = max_aux;

}
```

Annex C

Convection-Diffusion code - HRS

```
//--------------------------------------------------------------//
//                                                              //
//          Study for the computational resolution of           //
//     conservation equations of mass, momentum and energy      //
//          Numerical analysis and turbulence models            //
//                                                              //
//--------------------------------------------------------------//
// Author: Gerard Toset Alonso                                  //
// Date: June, 2023                                             //
// Code: Chapter 4 - Convection-Diffusion equation - HRS        //
//--------------------------------------------------------------//
//                      GENERAL DESCRIPTION                     //
// - SOLVER : Gauss-Seidel                                      //
// - 2D spatial discretization - Cell-centered nodes            //
// - HIGH RESOLUTION SCHEMES : This code applies a correction //
// factor for bP coefficient such as:                           //
// bP = (rhoP*Fi_0*vP)/deltaT +- |mf(Fi_fHRS* - Fi_fUDS*)|      //
// for f=e,w,n,s                                                //
// - HRS matrices : HRS[N+2][M+2][8]. To avoid a huge amount  //
// of matrices, the third dimension stands for:                 //
// HRSe, HRSw, HRSn, HRSs, UDSe, UDSw, UDSn, UDSs               //
// - NOTE : THIS CODE IS ONLY USED FOR SOLVING SUDS and QUICK //
//--------------------------------------------------------------//

#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>
#include <math.h>
#include <iomanip>

using namespace std;

//-------------------------------------------------------------//
// ------------------- Constant definition ------------------ //
const int N = 100;      // Number of CVs in X direction
const int M = 100;      // Number of CVs in Y direction
const int nmax = N+2;
const int mmax = M+2;
const int L = 1;
const int H = 1;
const int W = 1;
```

```
const double pi = M_PI;
const double delta_t = 0.01;            // Time step

//-----------------------------------------------------------//
// ----------------- Functions definition ------------------ //
void mesh_generation(double &delta_X, double &delta_Y, int CASE, vector<double> &
    x_VC, vector<double> &y_VC, vector<double> &x_N, vector<double> &y_N);
void velocity_field(int CASE, double v_0, vector<vector<double> > &vel_X, vector<
    vector<double> > &vel_Y, vector<double> &x_N, vector<double> &y_N);
void update_value(vector<vector<double> > &matriu, double value);
void update_matrix(vector<vector<double> > &new_matrix, vector<vector<double> > &
    previous_matrix);
void coeff_a(int scheme, int CASE, double rho, double delta_X, double delta_Y,
    double Gamma, vector<vector<double> > &vel_X, vector<vector<double> > &vel_Y,
    vector<vector<double> > &aE, vector<vector<double> > &aW, vector<vector<double>
     > &aN, vector<vector<double> > &aS, vector<vector<double> > &aP, vector<vector
    <double> > &me, vector<vector<double> > &mw, vector<vector<double> > &mn,
    vector<vector<double> > &ms );
void HRS_matrix(int scheme, vector<vector<vector<double> > > &HRS, vector<double>
    &x_VC, vector<double> &y_VC, vector<double> &x_N, vector<double> &y_N, vector<
    vector<double> > &Fi_0, vector<vector<double> > &me, vector<vector<double> > &
    mw, vector<vector<double> > &mn, vector<vector<double> > &ms );
void coeff_b(vector<vector<vector<double> > > &HRS, double rho, vector<vector<
    double> > &Fi_0, vector<vector<double> > &bP, vector<vector<double> > &me,
    vector<vector<double> > &mw, vector<vector<double> > &mn, vector<vector<double>
     > &ms );
void resolution(int CASE, vector<double> &x_N, vector<vector<double> > &Fi, vector
    <vector<double> > &Fi_est, vector<vector<double> > &aP, vector<vector<double> > >
     &aE, vector<vector<double> > &aW, vector<vector<double> > &aN, vector<vector<
    double> > &aS, vector<vector<double> > &bP);
void convergence(vector<vector<double> > &Fi, vector<vector<double> > &Fi_est,
    double &maxim);

ofstream outg ("geom_S_quick_Peinf.txt");
ofstream outf ("fi_S_quick_Peinf.txt");

//-----------------------------------------------------------//
// ------------------------- MAIN ------------------------- //
int main(){

        //-----------------------------------------------------//
        // I - DATA INPUT:
        // -------------------- NUMERICAL ---------------------- //
        double t = 0.0;
        double t_final = 100;
        double maxim;
        double delta = 10e-6;  // Convergence parameter
        double v_0 = 1.0;
        double Gamma = 1.0;    // Diffusive parameter
        double rho = 1000000.0;
        double Fi_initial = 1.0;
        double delta_X = L/double(N);
        double delta_Y = H/double(M);
        int i, j;        // Geometry parameters

        int CASE;
        // CASE 4 = SMITH-HUTTON
        int scheme;
        // scheme 4 = SUDS
        // scheme 5 = QUICK
        //-----------------------------------------------------//
        // II - PREVIOUS CALCULATIONS AND VECTOR DEFINITIONS:
        // --------------------- VECTORS ---------------------- //
```

```
        // Geometry vectors
        vector<double> x_VC(N+1), y_VC(M+1);
        vector<double> x_N(nmax), y_N(mmax);
        // Velocity vectors
        vector<vector<double> > vel_X(nmax, vector<double> (mmax)), vel_Y(nmax,
            vector<double> (mmax));
        // Discretization coefficients
        vector<vector<double> > aP(nmax, vector<double> (mmax)), aE(nmax, vector<
            double> (mmax)), aW(nmax, vector<double> (mmax)), aN(nmax, vector<
            double> (mmax)), aS(nmax, vector<double> (mmax)), bP(nmax, vector<
            double> (mmax));
        // Generic variable vectors
        vector<vector<double> > Fi_0(nmax, vector<double> (mmax)); // Generic
            variable t^n
        vector<vector<double> > Fi(nmax, vector<double> (mmax));  // Generic
            variable t^n+1
        vector<vector<double> > Fi_est(nmax, vector<double> (mmax));  // Generic
            variable ESTIMATED
        // HRS matrix
        vector<vector<vector<double> > > HRS(nmax, vector<vector<double> >(mmax,
            vector<double>(4)));
        // Mass fluxes vectors
        vector<vector<double> > me(nmax, vector<double> (mmax)), mw(nmax, vector<
            double> (mmax)), mn(nmax, vector<double> (mmax)), ms(nmax, vector<
            double> (mmax));

        // ----------------- PREVIOUS CALCULATIONS ------------------ //
        // Null-value initialization
        double null_val = 0.0;
        update_value(aP, null_val);
        update_value(aE, null_val);
        update_value(aW, null_val);
        update_value(aN, null_val);
        update_value(aS, null_val);
        update_value(bP, null_val);

        // CASE and SCHEME choice
        cout<<"CASE: ";
        cin>>CASE;
        cout<<"SCHEME: ";
        cin>>scheme;

        // Mesh generation as a function of the CASE
        mesh_generation(delta_X, delta_Y, CASE, x_VC, y_VC, x_N, y_N);
        // Velocity field as a function of the CASE
        velocity_field(CASE, v_0, vel_X, vel_Y, x_N, y_N);

        // Generic variable initial map
        update_value(Fi, Fi_initial);

        // Constant coefficient evaluation as a function of the CASE and SCHEME
        coeff_a(scheme, CASE, rho, delta_X, delta_Y, Gamma, vel_X, vel_Y, aE, aW,
            aN, aS, aP, me, mw, mn, ms);

        //-------------------------------------------------------//
        // III - TEMPORAL COMPUTATION:
        while(t<t_final){

                // Generic variable update from the previous time instant
                update_matrix(Fi_0, Fi);
                maxim = 1e4;

                // HRS matrix evaluation
```

```
			HRS_matrix(scheme, HRS, x_VC, y_VC, x_N, y_N, Fi_0, me, mw, mn, ms);

			// Gauss-Seidel solver with convercence parameter
			while(maxim>delta){

				// Estimated generic variable update
				update_matrix(Fi_est, Fi);

				// bP evaluation
				coeff_b(HRS, rho, Fi_0, bP, me, mw, mn, ms );

				// Generic variable solver as a function of the CASE
				resolution(CASE, x_N, Fi, Fi_est, aP, aE, aW, aN, aS, bP);

				// Convergence evaluation
				convergence(Fi, Fi_est, maxim);

			}
			cout<<t<<endl;
			t = t + delta_t;

	}

	//-------------------------------------------------------//
	// ------------------- OUTPUT FILES -------------------- //
	for(i=0; i<N+2; i++){
		for(j=0; j<M+2; j++){
			cout<<Fi[i][j]<<"   ";
		}
		cout<<endl;
	}

	for (i=0; i<N+2; i++){
		outg<<x_N[i]<<"\t";
	}
	outg<<endl;
	for (j=0; j<M+2; j++){
		outg<<y_N[j]<<"\t";
	}
	outg<<endl;

	for (i=0; i<N+2; i++){
		for (j=0; j<M+2; j++){
			outf<<Fi[j][i]<<"\t";
		}
		outf<<endl;
	}
	//-------------------------------------------------------//
	//-------------------------------------------------------//

}

void mesh_generation(double &delta_X, double &delta_Y, int CASE, vector<double> &x_VC, vector<double> &y_VC, vector<double> &x_N, vector<double> &y_N)
{
	int i, j;

	switch (CASE){

		case 4:
		delta_X = 2*L/double(N);
		//-----------------------------------------------------------//
```

```
			// A - CV position
			for(i=0; i<N+1; i++){
				x_VC[i] = (i*delta_X)-L;
			}
			for(j=0; j<M+1; j++){
				y_VC[j] = j*delta_Y;
			}
			//------------------------------------------------------------//
			//------------------------------------------------------------//
			// B - Nodes position
			x_N[0] = -L;		x_N[N+1]=L;
			y_N[0] = 0.0;		y_N[M+1]=H;
			for(i=1; i<N+1; i++){
				x_N[i] = 0.5*(x_VC[i]+x_VC[i-1]);
			}
			for(j=1; j<M+1; j++){
				y_N[j] = 0.5*(y_VC[j]+y_VC[j-1]);
			}
			//------------------------------------------------------------//
			break;
	}
}

void velocity_field(int CASE, double v_0, vector<vector<double> > &vel_X, vector<
    vector<double> > &vel_Y, vector<double> &x_N, vector<double> &y_N)
{

	int i, j;
	double alpha = 0.0;

	switch (CASE){

		case 4:
			double x;
			for (i=0; i<N+2; i++){
				for (j=0; j<M+2; j++){
					vel_X[i][j] = 2*y_N[j]*(1-pow(x_N[i],2));
					vel_Y[i][j] = -2*x_N[i]*(1-pow(y_N[j],2));
				}
			}
		break;
	}
}

void update_value(vector<vector<double> > &matriu, double value)
{
	int i, j;

	for(i=0; i<N+2; i++){
		for(j=0; j<M+2; j++){
			matriu[i][j] = value;
		}
	}
}

void update_matrix(vector<vector<double> > &new_matrix, vector<vector<double> > &
    previous_matrix)
{
	int i, j;

	for(i=0; i<N+2; i++){
		for(j=0; j<M+2; j++){
			new_matrix[i][j] = previous_matrix[i][j];
```

```
                }
        }
}

void coeff_a(int scheme, int CASE, double rho, double delta_X, double delta_Y,
    double Gamma, vector<vector<double> > &vel_X, vector<vector<double> > &vel_Y,
    vector<vector<double> > &aE, vector<vector<double> > &aW, vector<vector<double>
     > &aN, vector<vector<double> > &aS, vector<vector<double> > &aP, vector<vector
    <double> > &me, vector<vector<double> > &mw, vector<vector<double> > &mn,
    vector<vector<double> > &ms )
{
        int i, j;
        double Se, Sw, Sn, Ss;
        // Diffusive terms
        double De, Dw, Dn, Ds;
        // velocities
        vector<vector<double> > vE(nmax, vector<double> (mmax)), vW(nmax, vector<
            double> (mmax)), vN(nmax, vector<double> (mmax)), vS(nmax, vector<
            double> (mmax));                                                   //
            velocities

        // MASS FLUXES as a function of the CASE
        switch (CASE){

                case 4:
                        // Velocity harmonic mean is applied for the CASE 4
                        for(j=0; j<M+1; j++){
                            vE[0][j] = vel_X[0][j];  vW[0][j] = vel_X[0][j];
                                vE[N+1][j] = vel_X[N+1][j];  vW[N+1][j] = vel_X[N
                                    +1][j];
                                for (i=1; i<N; i++){
                                        vE[i][j]= delta_X/(((0.5*delta_X)/vel_X[i
                                            ][j])+((0.5*delta_X)/vel_X[i+1][j]));
                                        vW[i][j]= delta_X/(((0.5*delta_X)/vel_X[i
                                            -1][j])+((0.5*delta_X)/vel_X[i][j]));

                                }
                        }
                        for (i=0; i<=N+1; i++){
                                vN[i][0] = vel_Y[i][0];  vS[i][0] = vel_Y[i][0];
                                vN[i][M+1] = vel_Y[i][N+1];  vS[i][N+1] = vel_Y[i
                                    ][N+1];
                            for(j=1; j<M; j++){
                                   vN[i][j]= delta_Y/(((0.5*delta_Y)/vel_Y[i][j])
                                       +((0.5*delta_Y)/vel_Y[i][j+1]));
                                   vS[i][j]= delta_Y/(((0.5*delta_Y)/vel_Y[i][j
                                       -1])+((0.5*delta_Y)/vel_Y[i][j]));
                                }
                        }
                break;
        }
        Se = delta_Y*W;  Sw = delta_Y*W;
        Sn = delta_X*W;  Ss = delta_X*W;

        for(i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
                        me[i][j] = rho*Se*vE[i][j];  mw[i][j] = -rho*Sw*vW[i][j];
                        mn[i][j] = rho*Se*vN[i][j];  ms[i][j] = -rho*Sw*vS[i][j];
                }
        }

        // DISCRETIZATION COEFFICIENTS as a function of the SCHEME
        switch (scheme){
```

```
                case 4:
                case 5:
                        for(i=1; i<N+1; i++){
                                for(j=1; j<M+1; j++){
                                        //DIFFUSIVE TERMS
                                        De = Gamma*Se/delta_X;
                                        Dw = Gamma*Sw/delta_X;
                                        Dn = Gamma*Sn/delta_Y;
                                        Ds = Gamma*Ss/delta_Y;
                                        if(i==N){
                                                De = Gamma*Se/(0.5*delta_X);
                                        }
                                        if(i==1){
                                                Dw = Gamma*Se/(0.5*delta_X);
                                        }
                                        if(j==M){
                                                Dn = Gamma*Sn/(0.5*delta_Y);
                                        }
                                        if(j==1){
                                                Ds = Gamma*Ss/(0.5*delta_Y);
                                        }
                                        aE[i][j] = De - ((me[i][j] - abs(me[i][j])
                                            )/(2));
                                        aW[i][j] = Dw - ((mw[i][j] - abs(mw[i][j])
                                            )/(2));
                                        aN[i][j] = Dn - ((mn[i][j] - abs(mn[i][j])
                                            )/(2));
                                        aS[i][j] = Ds - ((ms[i][j] - abs(ms[i][j])
                                            )/(2));
                                }
                        }
                break;
        }

        // aP coefficient evaluation
        for(i=1; i<N+1; i++){
                for(j=1; j<M+1; j++){
                        aP[i][j] = aE[i][j] + aW[i][j] + aN[i][j] + aS[i][j] + rho
                            *0.0002/delta_t;
                }
        }
}

void HRS_matrix(int scheme, vector<vector<vector<double> > > &HRS, vector<double>
    &x_VC, vector<double> &y_VC, vector<double> &x_N, vector<double> &y_N, vector<
    vector<double> > &Fi_0, vector<vector<double> > &me, vector<vector<double> > &
    mw, vector<vector<double> > &mn, vector<vector<double> > &ms )
{
        // -------------- REMEMBER -------------- //
        // HRS[i][j][0] = HRSe
        // HRS[i][j][1] = HRSw
        // HRS[i][j][2] = HRSn
        // HRS[i][j][3] = HRSs
        // HRS[i][j][4] = UDSe
        // HRS[i][j][5] = UDSw
        // HRS[i][j][6] = UDSn
        // HRS[i][j][7] = UDSs
        // -------------------------------------- //

        // U-C-D scheme
        // U = most upstream node
        // C = first upstream node
        // D = downstream node
```

```
    int i, j, k;
    double xU, xC, xD, xf;            // Positions
    double fiU, fiC, fiD;             // Phi values
    double fiC_ad, xC_ad, xf_ad;      // Dimensionless positions and phi values

    // Dimensionless evaluations as a function of the SCHEME
    switch(scheme){
        case 4:
        for (i=0; i<N+2; i++){
            for(j=0; j<M+2; j++){
                if(i>0 && i<N+1 && j>0 && j<M+1){
                    // EAST FACES
                    if( i==N ){
                        HRS[i][j][0] = Fi_0[i+1][j];
                        HRS[i][j][4] = Fi_0[i+1][j];
                    }
                    else{
                        if (me[i][j]>=0){
                            xU = x_N[i-1];   xC = x_N[i];   xD = x_N[i+1];
                            xf = x_VC[i];
                            fiU = Fi_0[i-1][j];   fiC = Fi_0[i][j];       fiD = Fi_0[i+1][j];
                        }
                        else{
                            xU = x_N[i+2];   xC = x_N[i+1];   xD = x_N[i];
                            xf = x_VC[i];
                            fiU = Fi_0[i+2][j];   fiC = Fi_0[i+1][j];     fiD = Fi_0[i][j];
                        }
                        //Dimensionless computation
                        if (fiD == fiU){                   // Avoid 0/0 from the first iteration since Phi_0(t=0) = 1.0 for all the fluid domain
                            fiC_ad = 0.0;
                        }
                        else{
                            fiC_ad = (fiC-fiU)/(fiD-fiU);
                        }
                        xC_ad = (xC-xU)/(xD-xU);
                        xf_ad = (xf-xU)/(xD-xU);
                        //Dimension computation
                        HRS[i][j][0] = fiU + (fiD-fiU)*fiC_ad*(xf_ad/xC_ad);
                        HRS[i][j][4] = fiU + (fiD-fiU)*fiC_ad;
                    }
                    // WEST FACES
                    if( i==1 ){
                        HRS[i][j][1] = Fi_0[i-1][j];
                        HRS[i][j][5] = Fi_0[i-1][j];
                    }
                    else{
                        if (mw[i][j]>=0){
                            xU = x_N[i-2];   xC = x_N[i-1];   xD = x_N[i];
                            xf = x_VC[i-1];
```

```
                                        fiU = Fi_0[i-2][j];    fiC = Fi_0[i-1][j];      fiD = Fi_0[i][j];
                                }
                                else{
                                        xU = x_N[i+1];  xC = x_N[i];  xD = x_N[i-1];
                                        xf = x_VC[i-1];
                                        fiU = Fi_0[i+1][j];    fiC = Fi_0[i][j];       fiD = Fi_0[i-1][j];
                                }
                                //Dimensionless computation
                                if (fiD == fiU){                    // Avoid 0/0 from the first iteration since Phi_0(t=0) = 1.0 for all the fluid domain
                                        fiC_ad = 0.0;
                                }
                                else{
                                        fiC_ad = (fiC-fiU)/(fiD-fiU);
                                }
                                xC_ad = (xC-xU)/(xD-xU);
                                xf_ad = (xf-xU)/(xD-xU);
                                //Dimension computation
                                HRS[i][j][1] = fiU + (fiD-fiU)*fiC_ad*(xf_ad/xC_ad);
                                HRS[i][j][5] = fiU + (fiD-fiU)*fiC_ad;
                        }

                        // NORTH FACES
                        if( j==M){
                                HRS[i][j][2] = Fi_0[i][j+1];
                                HRS[i][j][6] = Fi_0[i][j+1];
                        }
                        else{
                                if (mn[i][j]>=0){
                                        xU = y_N[j-1];  xC = y_N[j];  xD = y_N[j+1];
                                        xf = y_VC[j];
                                        fiU = Fi_0[i][j-1];    fiC = Fi_0[i][j];       fiD = Fi_0[i][j+1];
                                }
                                else{
                                        xU = y_N[j+2];  xC = y_N[j+1];  xD = y_N[j];
                                        xf = y_VC[j];
                                        fiU = Fi_0[i][j+2];    fiC = Fi_0[i][j+1];      fiD = Fi_0[i][j];
                                }
                                //Dimensionless computation
                                if (fiD == fiU){                    // Avoid 0/0 from the first iteration since Phi_0(t=0) = 1.0 for all the fluid domain
                                        fiC_ad = 0.0;
                                }
```

```
						else{
							fiC_ad = (fiC-fiU)/(fiD-fiU);
						}
						xC_ad = (xC-xU)/(xD-xU);
						xf_ad = (xf-xU)/(xD-xU);
						//Dimension computation
						HRS[i][j][2] = fiU + (fiD-fiU)*fiC_ad*(xf_ad/xC_ad);
						HRS[i][j][6] = fiU + (fiD-fiU)*fiC_ad;
					}

					// SOUTH FACES
					if( j==1 ){
						HRS[i][j][3] = Fi_0[i][j-1];
						HRS[i][j][7] = Fi_0[i][j-1];
					}
					else{
						if (ms[i][j]>=0){
							xU = y_N[j-2];		xC = y_N[j-1];		xD = y_N[j];
							xf = y_VC[j-1];
							fiU = Fi_0[i][j-2];	fiC = Fi_0[i][j-1];		fiD = Fi_0[i][j];
						}
						else{
							xU = y_N[j+1];		xC = y_N[j];		xD = y_N[j-1];
							xf = y_VC[j-1];
							fiU = Fi_0[i][j+1];	fiC = Fi_0[i][j];		fiD = Fi_0[i][j-1];
						}
						//Dimensionless computation
						if (fiC == fiU){			// Avoid 0/0 from the first iteration since Phi_0(t=0) = 1.0 for all the fluid domain
							fiC_ad = 0.0;
						}
						else{
							fiC_ad = (fiC-fiU)/(fiD-fiU);
						}
						xC_ad = (xC-xU)/(xD-xU);
						xf_ad = (xf-xU)/(xD-xU);
						//Dimension computation
						HRS[i][j][3] = fiU + (fiD-fiU)*fiC_ad*(xf_ad/xC_ad);
						HRS[i][j][7] = fiU + (fiD-fiU)*fiC_ad;
					}
				}

				else{
					for(k=0; k<8; k++){
						for(i=0; i<N+2; i++){
							for(j=0; j<M+2; j++){
```

```
                                        HRS[i][j][k] =
                                            0.0;
                                    }
                                }
                            }
                        }

                    }
                }
                break;
                case 5:
                    for (i=0; i<N+2; i++){
                        for(j=0; j<M+2; j++){
                            if(i>0 && i<N+1 && j>0 && j<M+1){
                                // EAST FACES
                                if( i==N ){
                                    HRS[i][j][0] = Fi_0[i+1][j
                                        ];
                                    HRS[i][j][4] = Fi_0[i+1][j
                                        ];
                                }
                                else{
                                    if (me[i][j]>=0){
                                        xU = x_N[i-1];
                                                xC =
                                            x_N[i];
                                                xD =
                                            x_N[i+1];
                                        xf = x_VC[i];
                                        fiU = Fi_0[i-1][j
                                            ];      fiC =
                                            Fi_0[i][j];
                                                fiD =
                                            Fi_0[i+1][j];
                                    }
                                    else{
                                        xU = x_N[i+2];
                                                xC =
                                            x_N[i+1];
                                                xD =
                                            x_N[i];
                                        xf = x_VC[i];
                                        fiU = Fi_0[i+2][j
                                            ];      fiC =
                                            Fi_0[i+1][j];
                                                fiD = Fi_0
                                            [i][j];
                                    }
                                    //Dimensionless
                                        computation
                                    if (fiD == fiU){
                                            //Avoid 0/0
                                        from the first
                                        iteration since Phi_0(t
                                        =0) = 1.0 for all the
                                        fluid domain
                                        fiC_ad = 0.0;
                                    }
                                    else{
                                        fiC_ad = (fiC-fiU)
                                            /(fiD-fiU);
                                    }
                                    xC_ad = (xC-xU)/(xD-xU);
```

```
                xf_ad = (xf-xU)/(xD-xU);
                //Dimension computation
                HRS[i][j][0] = fiU + (fiD-
                    fiU)*( xf_ad + ((xf_ad
                    *(xf_ad-1))/(xC_ad*(
                    xC_ad-1)))*(fiC_ad -
                    xC_ad) );
                HRS[i][j][4] = fiU + (fiD-
                    fiU)*fiC_ad;
            }
            // WEST FACES
            if( i==1 ){
                HRS[i][j][1] = Fi_0[i-1][j
                    ];
                HRS[i][j][5] = Fi_0[i-1][j
                    ];
            }
            else{
                if (mw[i][j]>=0){
                    xU = x_N[i-2];
                            xC =
                        x_N[i-1];
                            xD =
                        x_N[i];
                    xf = x_VC[i-1];
                    fiU = Fi_0[i-2][j
                        ];      fiC =
                        Fi_0[i-1][j];
                            fiD = Fi_0
                        [i][j];
                }
                else{
                    xU = x_N[i+1];
                            xC =
                        x_N[i];
                            xD =
                        x_N[i-1];
                    xf = x_VC[i-1];
                    fiU = Fi_0[i+1][j
                        ];      fiC =
                        Fi_0[i][j];
                            fiD =
                        Fi_0[i-1][j];
                }
                //Dimensionless
                    computation
                if (fiD == fiU){
                        //Avoid 0/0
                    from the first
                    iteration since Phi_0(t
                    =0) = 1.0 for all the
                    fluid domain
                    fiC_ad = 0.0;
                }
                else{
                    fiC_ad = (fiC-fiU)
                        /(fiD-fiU);
                }
                xC_ad = (xC-xU)/(xD-xU);
                xf_ad = (xf-xU)/(xD-xU);
                //Dimension computation
                HRS[i][j][1] = fiU + (fiD-
                    fiU)*( xf_ad + ((xf_ad
```

```
*(xf_ad-1))/(xC_ad*(xC_ad-1)))*(fiC_ad - xC_ad) );
                HRS[i][j][5] = fiU + (fiD-fiU)*fiC_ad;
        }

        // NORTH FACES
        if( j==M){
                HRS[i][j][2] = Fi_0[i][j+1];
                HRS[i][j][6] = Fi_0[i][j+1];
        }
        else{
                if (mn[i][j]>=0){
                        xU = y_N[j-1];          xC = y_N[j];            xD = y_N[j+1];
                        xf = y_VC[j];
                        fiU = Fi_0[i][j-1];     fiC = Fi_0[i][j];               fiD = Fi_0[i][j+1];
                }
                else{
                        xU = y_N[j+2];          xC = y_N[j+1];          xD = y_N[j];
                        xf = y_VC[j];
                        fiU = Fi_0[i][j+2];     fiC = Fi_0[i][j+1];             fiD = Fi_0[i][j];
                }
                //Dimensionless computation
                if (fiD == fiU){                //Avoid 0/0 from the first iteration since Phi_0(t=0) = 1.0 for all the fluid domain
                        fiC_ad = 0.0;
                }
                else{
                        fiC_ad = (fiC-fiU)/(fiD-fiU);
                }
                xC_ad = (xC-xU)/(xD-xU);
                xf_ad = (xf-xU)/(xD-xU);
                //Dimension computation
                HRS[i][j][2] = fiU + (fiD-fiU)*( xf_ad + ((xf_ad*(xf_ad-1))/(xC_ad*(xC_ad-1)))*(fiC_ad - xC_ad) );
```

```
                    HRS[i][j][6] = fiU + (fiD-fiU)*fiC_ad;
                }

                // SOUTH FACES
                if( j==1 ){
                    HRS[i][j][3] = Fi_0[i][j-1];
                    HRS[i][j][7] = Fi_0[i][j-1];
                }
                else{
                    if (ms[i][j]>=0){
                        xU = y_N[j-2];      xC = y_N[j-1];      xD = y_N[j];
                        xf = y_VC[j-1];
                        fiU = Fi_0[i][j-2];     fiC = Fi_0[i][j-1];     fiD = Fi_0[i][j];
                    }
                    else{
                        xU = y_N[j+1];      xC = y_N[j];      xD = y_N[j-1];
                        xf = y_VC[j-1];
                        fiU = Fi_0[i][j+1];     fiC = Fi_0[i][j];     fiD = Fi_0[i][j-1];
                    }
                    //Dimensionless computation
                    if (fiC == fiU){            //Avoid 0/0 from the first iteration since Phi_0(t=0) = 1.0 for all the fluid domain
                        fiC_ad = 0.0;
                    }
                    else{
                        fiC_ad = (fiC-fiU)/(fiD-fiU);
                    }
                    xC_ad = (xC-xU)/(xD-xU);
                    xf_ad = (xf-xU)/(xD-xU);
                    //Dimension computation
                    HRS[i][j][3] = fiU + (fiD-fiU)*( xf_ad + ((xf_ad*(xf_ad-1))/(xC_ad*(xC_ad-1)))*(fiC_ad - xC_ad) );
                    HRS[i][j][7] = fiU + (fiD-fiU)*fiC_ad;
                }
```

```
                                                    }

                                                    else{
                                                            for(k=0; k<8; k++){
                                                                    for(i=0; i<N+2; i++){
                                                                            for(j=0; j<M+2; j++){
                                                                                    HRS[i][j][k] = 0.0;
                                                                            }
                                                                    }
                                                            }
                                                    }

                                            }
                                    }
                            break;
                    }


}

void coeff_b(vector<vector<vector<double> > > &HRS, double rho, vector<vector<double> > &Fi_0, vector<vector<double> > &bP, vector<vector<double> > &me, vector<vector<double> > &mw, vector<vector<double> > &mn, vector<vector<double> > &ms )
{
        int i, j;

        for(i=1; i<N+1; i++){
                for(j=1; j<M+1; j++){
                        bP[i][j] = rho*0.0002*Fi_0[i][j]/delta_t - me[i][j]*(HRS[i][j][0] - HRS[i][j][4]) + mw[i][j]*(HRS[i][j][1] - HRS[i][j][5]) - mn[i][j]*(HRS[i][j][2] - HRS[i][j][6]) + ms[i][j]*(HRS[i][j][3] - HRS[i][j][7]);
                }
        }
}

void resolution(int CASE, vector<double> &x_N, vector<vector<double> > &Fi, vector<vector<double> > &Fi_est, vector<vector<double> > &aP, vector<vector<double> > &aE, vector<vector<double> > &aW, vector<vector<double> > &aN, vector<vector<double> > &aS, vector<vector<double> > &bP)
{
        int i, j;
        double Fi_in = 1.0, Fi_out = 0.0;

        // INNNER NODES
        for(i=1; i<N+1; i++){
                for(j=1; j<M+1; j++){
                                Fi[i][j] = ( aE[i][j]*Fi_est[i+1][j] + aW[i][j]*Fi_est[i-1][j] + aN[i][j]*Fi_est[i][j+1] + aS[i][j]*Fi_est[i][j-1] + bP[i][j] ) / aP[i][j];
                                //Fi[i][j] = 0.0;
                }
        }

        // BOUNDARY CONDITIONS as a function of the CASE: FI for wall nodes are already known
        switch (CASE){


```

```
				case 4:
						int half = (N+2)/2;
						for(i=1; i<=half; i++){
								j=0;
								Fi[i][j] =	1+tanh(10*(2*x_N[i] + 1));
								j=M+1;
								Fi[i][j] = 1-tanh(10);
						}
						for(i=half+1; i<N+1; i++){
								j=0;
								Fi[i][j] =	Fi[i][j+1];
								j=M+1;
								Fi[i][j] = 1-tanh(10);
						}
						for(j=1; j<M+1; j++){
								i=0;
								Fi[i][j] = 1-tanh(10);
								i=N+1;
								Fi[i][j] = 1-tanh(10);
						}
				break;

		}
		// Arbitary temperature value assignation for the corner nodes
		Fi[0][0] = Fi[1][0];				Fi[0][M+1] = Fi[1][M+1];
		Fi[N+1][0] = Fi[N][0];				Fi[N+1][M+1] = Fi[N][M+1];

}

void convergence(vector<vector<double> > &Fi, vector<vector<double> > &Fi_est,
	double &maxim)
{
		int i, j;
		double diff, max_aux = 0.0;

		// Maximum difference between estimated and next computed temperature
		for(i=1; i<N+1; i++){
				for(j=1; j<M+1; j++){
						diff = abs(Fi[i][j] - Fi_est[i][j]);

						if(diff>=max_aux){
								max_aux = diff;
						}

				}
		}

		maxim = max_aux;

}
```

Annex D

Lid Driven Cavity code

```
//------------------------------------------------------------//
//                                                            //
//         Study for the computational resolution of          //
//     conservation equations of mass, momentum and energy    //
//         Numerical analysis and turbulence models           //
//                                                            //
//------------------------------------------------------------//
// Author: Gerard Toset Alonso                                //
// Date: June, 2023                                           //
// Code: Chapter 5 - FSM - Lid Driven Cavity                  //
//------------------------------------------------------------//
//                   GENERAL DESCRIPTION                      //
// - SOLVER : Gauss-Seidel                                    //
// - Transient regime                                         //
// - 2D spatial discretization - Cell-centered nodes          //
// - CDS - UDS - SUDS                                         //
//------------------------------------------------------------//

#include <iostream>
#include <math.h>
#include <cmath>
#include <fstream>
#include <ctime>
#include <vector>
using namespace std;

//------------------------------------------------------------//
// ------------------ Constant definitions ----------------- //
const double L=1;
const double H=1;
const double W=1;
const double rho=5000;
const double mu=1;
const double uref=1;

const int N=50;          // Number of CVs in X direction
const int M=50;          // Number of CVs in Y direction
const int nmax = N+2;
const int mmax = M+2;
double delta=1e-6;       // Convergence parameter
double delta_t=0.01;     // Time step
const int SCHEME = 0;    // 1:CDS - 2:UDS - 3:SUDS

```

```
//-----------------------------------------------------------//
// ----------------- Functions definition ------------------ //
void mesh_generation(vector<double> &x_N, vector<double> &xVC, vector<double> &y_N
    , vector<double> &yVC, double& delta_X, double& delta_Y);
void initial_vel_map(vector<vector<double> > &u0, vector<vector<double> > &v0);
void press_map(vector<vector<double> > &press);
void update(vector<vector<double> > &mat1, vector<vector<double> > &mat2);
void R_vel_u(vector<double> &x_N, vector<double> &xVC, vector<double> &y_N, vector
    <double> &yVC, vector<vector<double> > &R_u, vector<vector<double> > &vel_X,
    vector<vector<double> > &vel_Y);
void R_vel_v(vector<double> &x_N, vector<double> &xVC, vector<double> &y_N, vector
    <double> &yVC, vector<vector<double> > &R_v, vector<vector<double> > &vel_X,
    vector<vector<double> > &vel_Y);
void R_initial(vector<vector<double> > &R_u, vector<vector<double> > &R_v);
void interm_vel(double delta_t, vector<vector<double> > &vel_p, vector<vector<
    double> > &vel, vector<vector<double> > &R, vector<vector<double> > &R_PRE);
void resolution(double delta_t, vector<double> &x_N, vector<double> &xVC, vector<
    double> &y_N, vector<double> &yVC, vector<vector<double> > &press, double delta
    , vector<vector<double> > &vP_u, vector<vector<double> > &vP_v);
void maximum(vector<vector<double> > &press, vector<vector<double> > &press_est,
    double& error);
void next_vel_u(vector<double> &x_N, double delta_t, vector<vector<double> > &
    vnext_u, vector<vector<double> > &press, vector<vector<double> > &vP_u);
void next_vel_v(vector<double> &y_N, double delta_t, vector<vector<double> > &
    vnext_v, vector<vector<double> > &press, vector<vector<double> > &vP_v);
void min_time(double delta_X, double delta_Y, double& delta_t, vector<vector<
    double> > &vnext_u, vector<vector<double> > &vnext_v);

ofstream outg ("g_LDC_C_5000_25.txt");
ofstream outu ("u_LDC_C_5000_25.txt");
ofstream outv ("v_LDC_C_5000_25.txt");
ofstream outp ("p_LDC_C_5000_25.txt");

//-----------------------------------------------------------//
//--------------------------- MAIN --------------------------//

int main (){

        //-----------------------------------------------------------//
        // I - DATA INPUT:
        // --------------------- NUMERICAL ---------------------- //
        double delta_X, delta_Y;
        double t, error_s;
        int i,j;

        //-----------------------------------------------------------//
        // II - PREVIOUS CALCULATIONS AND VECTOR DEFINITIONS:
        // ---------------------- VECTORS ---------------------- //
        // Geometry
        vector<double> xVC(M+2), yVC(M+2), x_N(N+2), y_N(N+2);
        // Initial velocity map
        vector<vector<double> > u0(N+2,vector<double>(M+2)), v0(N+2,vector<double
            >(M+2));
        // Velocity for t=n
        vector<vector<double> > vel_X(N+2,vector<double>(M+2)), vel_Y(N+2,vector<
            double>(M+2));
        // Velocity for t=n-1
        vector<vector<double> > vel_X_PRE(N+2,vector<double>(M+2)), vel_Y_PRE(N+2,
            vector<double>(M+2));
        // Velocity for t=n+1
        vector<vector<double> > vnext_u(N+2,vector<double>(M+2)), vnext_v(N+2,
            vector<double>(M+2));
        // Intermediate velocity - Predictor velocity
```

```
    vector<vector<double> > vP_u(N+2,vector<double>(M+2)), vP_v(N+2,vector<double>(M+2));
    // Pressure map for t=n+1
    vector<vector<double> > press(N+2,vector<double>(M+2));
    // Pressure map for t=n
    vector<vector<double> > press_PRE(N+2,vector<double>(M+2));
    // R for u and v, and t=n and t=n-1
    vector<vector<double> > R_u(N+2,vector<double>(M+2)), R_u_PRE(N+2,vector<double>(M+2)), R_v(N+2,vector<double>(M+2)), R_v_PRE(N+2,vector<double>(M+2));

    // ----------------- PREVIOUS CALCULATIONS ------------------ //
    // Mesh generation
    mesh_generation(x_N, xVC, y_N, yVC, delta_X, delta_Y);
    // Initial velocity and pressure maps
    initial_vel_map(u0, v0);
    press_map(press);
    // Initially, R(u for t=n-1) = 0 and R(v for t=n-1) = 0.
    R_initial(R_u_PRE, R_v_PRE);

    //-----------------------------------------------------------//
    // Data update for t=n and t=n-1
    update(u0, vel_X);
    update(u0, vel_X_PRE);
    update(v0, vel_Y);
    update(v0, vel_Y_PRE);
    update(press, press_PRE);

    //-----------------------------------------------------------//
    //             TIME-CONVERGENCE STARTING POINT
    //-----------------------------------------------------------//
    t=0;
    error_s=99999;
    //
    while (error_s>0.001){
        // CFL condition for time step
        min_time(delta_X, delta_Y, delta_t, vnext_u, vnext_v);
        t=t+delta_t;
        //delta_t = 0.0001;

        // R(u for t=n) and R(v for t=n) evaluation
        R_vel_u(x_N, xVC, y_N, yVC, R_u, vel_X, vel_Y);
        R_vel_v(x_N, xVC, y_N, yVC, R_v, vel_X, vel_Y);

        // Intermediate velocity evaluation with real velocity for t=n and R for t=n and t=n-1
        interm_vel(delta_t, vP_u, vel_X, R_u, R_u_PRE);
        interm_vel(delta_t, vP_v, vel_Y, R_v, R_v_PRE);

        // Pressure solver: Gauss-Seidel node-by-node
        resolution(delta_t, x_N, xVC, y_N, yVC, press, delta, vP_u, vP_v);

        // Next velocity evaluation
        next_vel_u(x_N, delta_t, vnext_u, press, vP_u);
        next_vel_v(y_N, delta_t, vnext_v, press, vP_v);

        // Time convergence evaluation
        maximum(press, press_PRE, error_s);

        // Data update for the next time step
        update(vnext_u, vel_X);
        update(vnext_v, vel_Y);
        update(vel_X, vel_X_PRE);
```

```
                update(vel_Y, vel_Y_PRE);
                update(R_u, R_u_PRE);
                update(R_v, R_v_PRE);
                update(press, press_PRE);

                cout<<t;
                cout<<endl;
        }

        //----------------------------------------------//
        // --------------- OUTPUT FILES --------------- //
        for (i=0; i<N+2; i++){
                outg<<x_N[i]<<"\t";
        }
        outg<<endl;
        for (j=0; j<M+2; j++){
                outg<<y_N[j]<<"\t";
        }
        outg<<endl;
        for(i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
                        outu<<vnext_u[i][j]<<"\t";
                }
                outu<<endl;
        }
        for(i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
                        outv<<vnext_v[i][j]<<"\t";
                }
                outv<<endl;
        }
        for(i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
                        outp<<press[i][j]<<"\t";
                }
                outp<<endl;
        }
        //----------------------------------------------//
        //----------------------------------------------//

}

//--------------------------------------------------------------------//
// ------------------------- MAIN FUNCTIONS ------------------------- //
void mesh_generation(vector<double> &x_N, vector<double> &xVC, vector<double> &y_N
    , vector<double> &yVC, double& delta_X, double& delta_Y)
{
    int i, j;
    delta_X=L/N;                        delta_Y=H/M;

    // -------- x-mesh -------- //
    x_N[0]=0;                           xVC[0]=0;
    for (i=1; i<N+1; i++){
        x_N[i]= (i-0.5)*delta_X;
        xVC[i]= i*delta_X;
    }
    x_N[N+1]=L;                         xVC[N+1]=0;

    // -------- y-mesh -------- //
    y_N[0]=0;                       yVC[0]=0;
    for (j=1; j<M+1; j++){
                y_N[j]= (j-0.5)*delta_Y;
                yVC[j]= j*delta_Y;
```

```
        }
    y_N[M+1]=H;                          yVC[M+1]=0;
}

void initial_vel_map(vector<vector<double> > &u0, vector<vector<double> > &v0)
{
        int i, j;
        double u_inicial=0;
        double v_inicial=0;

        for (i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
                        if(j==M and i<N+1){
                                u0[i][M] = uref;
                        }
                        else {
                                u0[i][j] = u_inicial;
                        }
        }
        }

        for (i=0; i<N+2; i++){
                for(j=0; j<M+1; j++){
                        v0[i][j] = v_inicial;
                }
                v0[i][M+1] = 0;
        }
}

void press_map(vector<vector<double> > &press)
{
        int i,j;

        // Initial arbitrary pressure
        for (i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
                        press[i][j] = 0;
                }
        }
}

void update(vector<vector<double> > &mat1, vector<vector<double> > &mat2)
{
        int i,j;

        for (i=0; i<N+2; i++){
                for (j=0; j<M+2; j++){
                        mat2[i][j] = mat1[i][j];
                }
        }
}

void R_initial(vector<vector<double> > &R_u, vector<vector<double> > &R_v)
{
        int i,j;

        for(i=0; i<N+2; i++ ){
                for(j=0; j<M+2; j++){
                        R_u[i][j]=0;
                        R_v[i][j]=0;
                }
        }
}
```

```
void R_vel_u(vector<double> &x_N, vector<double> &xVC, vector<double> &y_N, vector
    <double> &yVC, vector<vector<double> > &R_u, vector<vector<double> > &vel_X,
    vector<vector<double> > &vel_Y)
{
        int i, j;
        double me, mw, mn, ms;          // mass fluxes
        double ue, uw, un, us;          // velocities with CDS - UDS - QUICK
        double Ae, Aw, An, As;          // Areas
        double dEP, dPW, dNP, dPS, vVC; // Distances
        double fe, fw, fn, fs;          // fe = 0 if me>0

        double xU, xC, xD, xf;          // Positions
        double fiU, fiC, fiD;           // Phi values
        double fiC_ad, xC_ad, xf_ad;    // Adimensional values

        for (i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
                        if(i>0 && i<N && j>1 && j<M){
                                Ae = yVC[j+1]-yVC[j];                   Aw = Ae;
                                An = x_N[i+1]-x_N[i];                   As = An;

                                me = rho*(vel_X[i+1][j]+vel_X[i][j])*Ae/2;
                                mw = rho*(vel_X[i][j]+vel_X[i-1][j])*Aw/2;
                                mn = rho*(vel_Y[i][j]*(xVC[i]-x_N[i]) + vel_Y[i
                                    +1][j]*(x_N[i+1]-xVC[i]) );
                                ms = rho*(vel_Y[i][j-1]*(xVC[i]-x_N[i]) + vel_Y[i
                                    +1][j-1]*(x_N[i+1]-xVC[i]) );

                                switch(SCHEME){
                                        case 1:
                                                // CDS
                                                ue = ((x_N[i+1]-xVC[i])/(xVC[i+1]-
                                                    xVC[i]))*(vel_X[i+1][j]-vel_X[i
                                                    ][j])+vel_X[i][j];
                                                uw = ((xVC[i]-x_N[i])/(xVC[i]-xVC[
                                                    i-1]))*(vel_X[i-1][j]-vel_X[i][
                                                    j])+vel_X[i][j];
                                                un = ((yVC[j]-y_N[j])/(y_N[j+1]-
                                                    y_N[j]))*(vel_X[i][j+1]-vel_X[i
                                                    ][j])+vel_X[i][j];
                                                us = ((y_N[j]-yVC[j-1])/(y_N[j]-
                                                    y_N[j-1]))*(vel_X[i][j-1]-vel_X
                                                    [i][j])+vel_X[i][j];
                                        break;
                                        case 2:
                                                // UDS
                                                if(me>=0){
                                                        fe = 0.0;
                                                }
                                                else{
                                                        fe = 1.0;
                                                }
                                                if(mw>=0){
                                                        fw = 1.0;
                                                }
                                                else{
                                                        fw = 0.0;
                                                }
                                                if(mn>=0){
                                                        fn = 0.0;
                                                }
                                                else{
```

```
                                fn = 1.0;
                        }
                        if(ms>=0){
                                fs = 1.0;
                        }
                        else{
                                fs = 0.0;
                        }
                        ue = (fe)*(vel_X[i+1][j]-vel_X[i][j])+vel_X[i][j];
                        uw = (fw)*(vel_X[i-1][j]-vel_X[i][j])+vel_X[i][j];
                        un = (fn)*(vel_X[i][j+1]-vel_X[i][j])+vel_X[i][j];
                        us = (fs)*(vel_X[i][j-1]-vel_X[i][j])+vel_X[i][j];
                break;
                case 3:
                        // SUDS
                        // SUDS + fe=0 o fe=1 for UDS
                        if(me>=0){
                                xU = x_N[i-1];          xC = x_N[i];            xD = x_N[i+1];
                                xf = xVC[i];
                                fiU = vel_X[i-1][j];    fiC = vel_X[i][j];      fiD = vel_X[i+1][j];
                        }
                        else{
                                xU = x_N[i+2];          xC = x_N[i+1];          xD = x_N[i];
                                xf = xVC[i];
                                fiU = vel_X[i+2][j];    fiC = vel_X[i+1][j];    fiD = vel_X[i][j];
                        }
                        if (fiD == fiU){ //Avoid 0/0 from the first iteration since Phi_0(t=0) = 1.0 for all the fluid domain
                                fiC_ad = 0.0;
                        }
                        else{
                                fiC_ad = (fiC-fiU)/(fiD-fiU);
                        }
                        xC_ad = (xC-xU)/(xD-xU);
                        xf_ad = (xf-xU)/(xD-xU);
                        ue = fiU + (fiD-fiU)*fiC_ad*(xf_ad/xC_ad);

                        if(mw>=0){
                                xU = x_N[i-2]; xC = x_N[i-1]; xD = x_N[i];
                                xf = xVC[i-1];
                                fiU = vel_X[i-2][j]; fiC = vel_X[i-1][j]; fiD = vel_X[i][j];
```

```
}
else{
        xU = x_N[i+1]; xC = x_N[i]; xD = x_N[i-1];
        xf = xVC[i-1];
        fiU = vel_X[i+1][j]; fiC = vel_X[i][j]; fiD = vel_X[i-1][j];
}
if (fiD == fiU){ //Avoid 0/0 from the first iteration since Phi_0 (t=0) = 1.0 for all the fluid domain
        fiC_ad = 0.0;
}
else{
        fiC_ad = (fiC-fiU)/(fiD-fiU);
}
xC_ad = (xC-xU)/(xD-xU);
xf_ad = (xf-xU)/(xD-xU);
uw = fiU + (fiD-fiU)*fiC_ad*(xf_ad/xC_ad);

if(mn>=0){
        xU = y_N[j-1]; xC = y_N[j]; xD = y_N[j+1];
        xf = yVC[j];
        fiU = vel_X[i][j-1]; fiC = vel_X[i][j]; fiD = vel_X[i][j+1];
}
else{
        xU = y_N[j+2];          xC = y_N[j+1];          xD = y_N[j];
        xf = yVC[j];
        fiU = vel_X[i][j+2];    fiC = vel_X[i][j+1];    fiD = vel_X[i][j];
}
if (fiD == fiU){ //Avoid 0/0 from the first iteration since Phi_0 (t=0) = 1.0 for all the fluid domain
        fiC_ad = 0.0;
}
else{
        fiC_ad = (fiC-fiU)/(fiD-fiU);
}
xC_ad = (xC-xU)/(xD-xU);
xf_ad = (xf-xU)/(xD-xU);
un = fiU + (fiD-fiU)*fiC_ad*(xf_ad/xC_ad);

if(ms>=0){
        xU = y_N[j-2];          xC = y_N[j-1];          xD = y_N[j];
        xf = yVC[j-1];
```

```
                                        fiU = vel_X[i][j-2];    fiC = vel_X[i][j-1];    fiD = vel_X[i][j];
                                    }
                                    else{
                                        xU = y_N[j+1];          xC = y_N[j];            xD = y_N[j-1];
                                        xf = yVC[j-1];
                                        fiU = vel_X[i][j+1];    fiC = vel_X[i][j];      fiD = vel_X[i][j-1];
                                    }
                                    if (fiC == fiU){ //Avoid 0/0 from the first iteration since Phi_0(t=0) = 1.0 for all the fluid domain
                                        fiC_ad = 0.0;
                                    }
                                    else{
                                        fiC_ad = (fiC-fiU)/(fiD-fiU);
                                    }
                                    xC_ad = (xC-xU)/(xD-xU);
                                    xf_ad = (xf-xU)/(xD-xU);
                                    us = fiU + (fiD-fiU)*fiC_ad*(xf_ad/xC_ad);
                                break;
                            }

                            dEP = xVC[i+1]-xVC[i];                  dPW = xVC[i]-xVC[i-1];
                            dNP = y_N[j+1]-y_N[j];                  dPS = y_N[j]-y_N[j-1];
                            vVC = (yVC[j]-yVC[j-1])*(x_N[i+1]-x_N[i]);

                            R_u[i][j]=(1/vVC)*(-(me*ue-mw*uw+mn*un-ms*us)+mu*((vel_X[i+1][j]-vel_X[i][j])*Ae/dEP-(vel_X[i][j]-vel_X[i-1][j])*Aw/dPW +(vel_X[i][j+1]-vel_X[i][j])*An/dNP-(vel_X[i][j]-vel_X[i][j-1])*As/dPS));
                        }
                        else{
                            R_u[i][j]=0;
                        }
                }
        }
}

void R_vel_v(vector<double> &x_N, vector<double> &xVC, vector<double> &y_N, vector<double> &yVC, vector<vector<double> > &R_v, vector<vector<double> > &vel_X, vector<vector<double> > &vel_Y)
{
        int i, j;
        double me, mw, mn, ms;                          // mass fluxes
        double ve, vw, vn, vs;                          // velocities with CDS - UDS - QUICK
        double Ae, Aw, An, As;                          // Areas
        double dEP, dPW, dNP, dPS, vVC;         // Distances
        double fe, fw, fn, fs;

```

```
        double xU, xC, xD, xf;                      // Positions
        double fiU, fiC, fiD;                       // Phi values
        double fiC_ad, xC_ad, xf_ad;        // Adimensional values

        for (i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
                        if(i>1 && i<N && j>0 && j<M){
                                Ae = y_N[j+1]-y_N[j];                   Aw = Ae;
                                An = xVC[i]-xVC[i-1];                   As = An;

                                me = rho*(vel_X[i][j]*(yVC[j]-y_N[j])+vel_X[i][j+1]*(y_N[j+1]-yVC[j]));
                                mw = rho*(vel_X[i-1][j]*(yVC[j]-y_N[j])+vel_X[i-1][j+1]*(y_N[j+1]-yVC[j]));
                                mn = rho*(vel_Y[i][j]+vel_Y[i][j+1])*An/2;
                                ms = rho*(vel_Y[i][j-1]+vel_Y[i][j])*As/2;

                                switch(SCHEME){
                                        case 1:
                                                // CDS
                                                ve = ((xVC[i]-x_N[i])/(x_N[i+1]-x_N[i]))*(vel_Y[i+1][j]-vel_Y[i][j])+vel_Y[i][j];
                                                vw = ((x_N[i]-xVC[i-1])/(x_N[i]-x_N[i-1]))*(vel_Y[i-1][j]-vel_Y[i][j])+vel_Y[i][j];
                                                vn = ((y_N[j+1]-yVC[j])/(yVC[j+1]-yVC[j]))*(vel_Y[i][j+1]-vel_Y[i][j])+vel_Y[i][j];
                                                vs = ((yVC[j]-y_N[j])/(yVC[j]-yVC[j-1]))*(vel_Y[i][j-1]-vel_Y[i][j])+vel_Y[i][j];
                                        break;
                                        case 2:
                                                // UDS
                                                if(me>=0){
                                                        fe = 0.0;
                                                }
                                                else{
                                                        fe = 1.0;
                                                }
                                                if(mw>=0){
                                                        fw = 1.0;
                                                }
                                                else{
                                                        fw = 0.0;
                                                }
                                                if(mn>=0){
                                                        fn = 0.0;
                                                }
                                                else{
                                                        fn = 1.0;
                                                }
                                                if(ms>=0){
                                                        fs = 1.0;
                                                }
                                                else{
                                                        fs = 0.0;
                                                }
                                                ve = (fe)*(vel_Y[i+1][j]-vel_Y[i][j])+vel_Y[i][j];
                                                vw = (fw)*(vel_Y[i-1][j]-vel_Y[i][j])+vel_Y[i][j];
```

```
                        vn = (fn)*(vel_Y[i][j+1]-vel_Y[i][j])+vel_Y[i][j];
                        vs = (fs)*(vel_Y[i][j-1]-vel_Y[i][j])+vel_Y[i][j];
                break;
                case 3:
                        // SUDS
                        // SUDS + fe=0 o fe=1 for UDS
                        if(me>=0){
                                fe = 0.0;
                                xU = x_N[i-1];          xC = x_N[i];            xD = x_N[i+1];
                                xf = xVC[i];
                                fiU = vel_Y[i-1][j];    fiC = vel_Y[i][j];      fiD = vel_Y[i+1][j];
                        }
                        else{
                                fe = 1.0;
                                xU = x_N[i+2];          xC = x_N[i+1];          xD = x_N[i];
                                xf = xVC[i];
                                fiU = vel_Y[i+2][j];    fiC = vel_Y[i+1][j];    fiD = vel_Y[i][j];
                        }
                        if (fiD == fiU){                // Avoid 0/0 from the first iteration since Phi_0(t=0) = 1.0 for all the fluid domain
                                fiC_ad = 0.0;
                        }
                        else{
                                fiC_ad = (fiC-fiU)/(fiD-fiU);
                        }
                        xC_ad = (xC-xU)/(xD-xU);
                        xf_ad = (xf-xU)/(xD-xU);
                        ve = fiU + (fiD-fiU)*fiC_ad*(xf_ad/xC_ad);

                        if(mw>=0){
                                fw = 1.0;
                                xU = x_N[i-2];          xC = x_N[i-1];          xD = x_N[i];
                                xf = xVC[i-1];
                                fiU = vel_Y[i-2][j];    fiC = vel_Y[i-1][j];    fiD = vel_Y[i][j];
                        }
                        else{
                                fw = 0.0;
                                xU = x_N[i+1];          xC = x_N[i];            xD
```

```
                        = x_N[i-1];
                    xf = xVC[i-1];
                    fiU = vel_Y[i+1][j];
                        fiC = vel_Y[i][j];
                                        fiD =
                        vel_Y[i-1][j];
            }
            if (fiD == fiU){                    //
                Avoid 0/0 from the first
                iteration since Phi_0(t=0) =
                1.0 for all the fluid domain
                    fiC_ad = 0.0;
            }
            else{
                    fiC_ad = (fiC-fiU)/(fiD-
                        fiU);
            }
            xC_ad = (xC-xU)/(xD-xU);
            xf_ad = (xf-xU)/(xD-xU);
            vw = fiU + (fiD-fiU)*fiC_ad*(xf_ad
                /xC_ad);

            if(mn>=0){
                    fn = 0.0;
                    xU = y_N[j-1];
                                xC = y_N[j];
                                                xD
                        = y_N[j+1];
                    xf = yVC[j];
                    fiU = vel_Y[i][j-1];
                        fiC = vel_Y[i][j];
                                        fiD =
                        vel_Y[i][j+1];
            }
            else{
                    fn = 1.0;
                    xU = y_N[j+2];
                                xC = y_N[j+1];
                                            xD =
                        y_N[j];
                    xf = yVC[j];
                    fiU = vel_Y[i][j+2];
                        fiC = vel_Y[i][j+1];
                            fiD = vel_Y[i][j];
            }
            if (fiD == fiU){                    //
                Avoid 0/0 from the first
                iteration since Phi_0(t=0) =
                1.0 for all the fluid domain
                    fiC_ad = 0.0;
            }
            else{
                    fiC_ad = (fiC-fiU)/(fiD-
                        fiU);
            }
            xC_ad = (xC-xU)/(xD-xU);
            xf_ad = (xf-xU)/(xD-xU);
            vn = fiU + (fiD-fiU)*fiC_ad*(xf_ad
                /xC_ad);

            if(ms>=0){
                    fs = 1.0;
```

```
                            xU = y_N[j-2];      xC = y_N[j-1];      xD = y_N[j];
                            xf = yVC[j-1];
                            fiU = vel_Y[i][j-2];    fiC = vel_Y[i][j-1];    fiD = vel_Y[i][j];
                        }
                        else{
                            fs = 0.0;
                            xU = y_N[j+1];      xC = y_N[j];        xD = y_N[j-1];
                            xf = yVC[j-1];
                            fiU = vel_Y[i][j+1];    fiC = vel_Y[i][j];      fiD = vel_Y[i][j-1];
                        }
                        if (fiC == fiU){            // Avoid 0/0 from the first iteration since Phi_0(t=0) = 1.0 for all the fluid domain
                            fiC_ad = 0.0;
                        }
                        else{
                            fiC_ad = (fiC-fiU)/(fiD-fiU);
                        }
                        xC_ad = (xC-xU)/(xD-xU);
                        xf_ad = (xf-xU)/(xD-xU);
                        vs = fiU + (fiD-fiU)*fiC_ad*(xf_ad/xC_ad);
                    break;
                }

                dEP = x_N[i+1]-x_N[i];                  dPW = x_N[i]-x_N[i-1];
                dNP = yVC[j+1]-yVC[j];                  dPS = yVC[j]-yVC[j-1];
                vVC = (xVC[i]-xVC[i-1])*(y_N[j+1]-y_N[j]);

                R_v[i][j]=(1/vVC)*(-(me*ve-mw*vw+mn*vn-ms*vs)+mu*((vel_Y[i+1][j]-vel_Y[i][j])*Ae/dEP-(vel_Y[i][j]-vel_Y[i-1][j])*Aw/dPW +(vel_Y[i][j+1]-vel_Y[i][j])*An/dNP-(vel_Y[i][j]-vel_Y[i][j-1])*As/dPS));
            }
            else {
                R_v[i][j]=0;
            }
        }
    }
}

void interm_vel(double delta_t, vector<vector<double> > &vel_p, vector<vector<double> > &vel, vector<vector<double> > &R, vector<vector<double> > &R_PRE)
{
    int i, j;
    for (i=0; i<N+2; i++){
        for (j=0; j<M+2; j++){
```

```
                        vel_p[i][j] = vel[i][j]+(delta_t/rho)*(1.5*R[i][j]-0.5*R_PRE[i][j]);
                        //vel_p[i][j] = vel[i][j]+(delta_t/rho)*R[i][j];
                }
        }
}

void resolution(double delta_t, vector<double> &x_N, vector<double> &xVC, vector<double> &y_N, vector<double> &yVC, vector<vector<double> > &press, double delta, vector<vector<double> > &vP_u, vector<vector<double> > &vP_v)
{
        vector<vector<double> > press_est(N+2,vector<double>(M+2));
        double error;
        int i, j;


        double dpe, dpw, dpn, dps;
        double Ae, Aw, An, As;
        double aE, aW, aN, aS, aP, bP;

        // ------------ GAUSS-SEIDEL solver ------------ //
        error=1000;
        while (error>delta){

                update(press, press_est);
                // Internal nodes
                for(i=1; i<N+1; i++){
                        for (j=1; j<M+1; j++){

                                dpe = x_N[i+1]-x_N[i];          dpw = x_N[i]-x_N[i-1];
                                dpn = y_N[j+1]-y_N[j];          dps = y_N[j]-y_N[j-1];
                                Ae = yVC[j]-yVC[j-1];           Aw = Ae;
                                An = xVC[i]-xVC[i-1];           As = An;

                                aE = Ae/dpe;                            aW = Aw/dpw;
                                aN = An/dpn;                            aS = As/dps;
                                aP = aE + aN + aS + aW;
                                bP = (-rho/delta_t)*(vP_u[i][j]*Ae - vP_u[i-1][j]*Aw + vP_v[i][j]*An - vP_v[i][j-1]*As);

                                press[i][j] = (press_est[i+1][j]*aE+press_est[i-1][j]*aW+press_est[i][j+1]*aN+press_est[i][j-1]*aS+bP)/aP;
                        }
                }
                // Upper and Lower nodes
                for (i=1; i<N+1; i++){
                        press[i][0] = press[i][1];
                        press[i][M+1] = press[i][M];
                }
                // Right and left nodes
                for (j=1; j<M+1; j++){
                        press[0][j] = press[1][j];
                        press[N+1][j] = press[N][j];
                }
                // Corner nodes
                press[0][0] = press[1][1];              press[0][M+1] = press[1][M];
```

```
                press[N+1][M+1] = press[N][M];          press[N+1][0] = press[N
                    ][1];

                // Convergence analysis
                maximum(press, press_est, error);

        }
}

void maximum(vector<vector<double> > &press, vector<vector<double> > &press_est,
    double& error)
{
        int i,j;
        double dif;

        error=0;
        for(i=0; i<N+2; i++){
                for (j=0; j<M+2; j++){
                        dif = abs(press[i][j]-press_est[i][j]);
                        if(dif>error){
                                error=dif;
                        }
                        else{
                                error=error;
                        }
                }
        }
}

void next_vel_u(vector<double> &x_N, double delta_t, vector<vector<double> > &
    vnext_u, vector<vector<double> > &press, vector<vector<double> > &vP_u)
{
        int i, j;
        double dpe;

        for (i=0; i<N+2; i++){
                for (j=0; j<M+2; j++){
                        if(i>0 && i<N && j>1 && j<M){
                                dpe = x_N[i+1]-x_N[i];
                                vnext_u[i][j] = vP_u[i][j]-(delta_t/rho)*((press[i
                                    +1][j]-press[i][j])/dpe);
                        }
                        else {
                                vnext_u[i][j] = vP_u[i][j];
                        }
                }
        }
}

void next_vel_v(vector<double> &y_N, double delta_t, vector<vector<double> > &
    vnext_v, vector<vector<double> > &press, vector<vector<double> > &vP_v)
{
        int i,j;
        double dpn;

        for (i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
                        if(i>1 && i<N && j>0 && j<M){
                                dpn = y_N[j+1]-y_N[j];
                                vnext_v[i][j] = vP_v[i][j]-(delta_t/rho)*((press[i
                                    ][j+1]-press[i][j])/dpn);
                        }
                        else{
```

```
                                vnext_v[i][j] = vP_v[i][j];
                        }
                }
        }
}

void min_time(double delta_X, double delta_Y, double& delta_t, vector<vector<
    double> > &vnext_u, vector<vector<double> > &vnext_v)
{
        double t_cX, t_cY, t_dX, t_dY, v_modul;
        int i,j;

        delta_t=999999;

        for(i=1; i<=N-1; i++){
                for(j=1; j<=M-1; j++){
                        v_modul=sqrt(pow(vnext_u[i][j],2)+pow(vnext_v[i][j],2));
                        t_cX=0.35*delta_X/v_modul;
                        t_cY=0.35*delta_Y/v_modul;
                        t_dX=0.2*pow(delta_X, 2)*rho/mu;
                        t_dY=0.2*pow(delta_Y, 2)*rho/mu;
                        if (t_cX<delta_t){
                                delta_t = t_cX;
                        }
                        else{
                                delta_t = delta_t;
                        }
                        if (t_cY<delta_t){
                                delta_t = t_cY;
                        }
                        else {
                                delta_t = delta_t;
                        }
                        if (t_dX<delta_t){
                                delta_t = t_dX;
                        }
                        else {
                                delta_t = delta_t;
                        }
                        if (t_dY<delta_t){
                                delta_t = t_dY;
                        }
                        else {
                                delta_t = delta_t;
                        }
                }
        }
}
```

Annex E

Differential Heated Cavity code

```
//-----------------------------------------------------------//
//                                                           //
//        Study for the computational resolution of          //
//    conservation equations of mass, momentum and energy    //
//        Numerical analysis and turbulence models           //
//                                                           //
//-----------------------------------------------------------//
// Author: Gerard Toset Alonso                               //
// Date: June, 2023                                          //
// Code: Chapter 5 - FSM - Differential Heated Cavity        //
//-----------------------------------------------------------//
//                  GENERAL DESCRIPTION                      //
// - SOLVER : Gauss-Seidel                                   //
// - Transient regime                                        //
// - 2D spatial discretization - Cell-centered nodes         //
// - CDS                                                     //
//-----------------------------------------------------------//

#include <iostream>
#include <math.h>
#include <cmath>
#include <fstream>
#include <ctime>
#include <vector>
using namespace std;

//-----------------------------------------------------------//
// ------------------ Constant definitions ----------------- //
const double L=1;
const double H=1;
const double W=1;
const double rho=1;
const double mu=1;
const double uref=0;

const int N=50;                        // Number of CVs in X direction
const int M=50;                        // Number of CVs in Y direction
const int nmax = N+2;
const int mmax = M+2;
double delta=1e-6;               // Convergence parameter
double delta_t=0.01;             // Time step

const double g = 1;
```

```
const double beta = 1000; //Barometric expansion coefficient. Used to change
    Rayleigh
const double T0 = 1;             // Hot temperature
const double T_var = 1; // Temperature variation from Hot wall to Cold wall
const double cp = 1;
const double lambda = 1;

//-----------------------------------------------------------//
// ----------------- Functions definition ------------------ //
void mesh_generation(vector<double> &x_N, vector<double> &xVC, vector<double> &y_N
    , vector<double> &yVC, double& delta_X, double& delta_Y);
void initial_vel_map(vector<vector<double> > &u0, vector<vector<double> > &v0);
void press_map(vector<vector<double> > &press);
void temp_map(vector<vector<double> > &temp);
void update(vector<vector<double> > &mat1, vector<vector<double> > &mat2);
void R_vel_u(vector<double> &x_N, vector<double> &xVC, vector<double> &y_N, vector
    <double> &yVC, vector<vector<double> > &R_u, vector<vector<double> > &vel_X,
    vector<vector<double> > &vel_Y);
void R_vel_v(vector<vector<double> > &temp_PRE, vector<double> &x_N, vector<double
    > &xVC, vector<double> &y_N, vector<double> &yVC, vector<vector<double> > &R_v,
     vector<vector<double> > &vel_X, vector<vector<double> > &vel_Y);
void R_initial(vector<vector<double> > &R_u, vector<vector<double> > &R_v);
void interm_vel(double delta_t, vector<vector<double> > &vel_p, vector<vector<
    double> > &vel, vector<vector<double> > &R, vector<vector<double> > &R_PRE);
void resolution(double delta_t, vector<double> &x_N, vector<double> &xVC, vector<
    double> &y_N, vector<double> &yVC, vector<vector<double> > &press, double delta
    , vector<vector<double> > &vP_u, vector<vector<double> > &vP_v);
void resolution_temp(double delta_t, vector<double> &x_N, vector<double> &xVC,
    vector<double> &y_N, vector<double> &yVC, vector<vector<double> > &temp_PRE,
    vector<vector<double> > &temp, double delta, vector<vector<double> > &vnext_u,
    vector<vector<double> > &vnext_v);
void maximum(vector<vector<double> > &press, vector<vector<double> > &press_est,
    double& error);
void next_vel_u(vector<double> &x_N, double delta_t, vector<vector<double> > &
    vnext_u, vector<vector<double> > &press, vector<vector<double> > &vP_u);
void next_vel_v(vector<double> &y_N, double delta_t, vector<vector<double> > &
    vnext_v, vector<vector<double> > &press, vector<vector<double> > &vP_v);
void min_time(double delta_X, double delta_Y, double& delta_t, vector<vector<
    double> > &vnext_u, vector<vector<double> > &vnext_v);

ofstream outg ("g_DHC_C_10_3.txt");
ofstream outu ("u_DHC_C_10_3.txt");
ofstream outv ("v_DHC_C_10_3.txt");
ofstream outp ("p_DHC_C_10_3.txt");
ofstream outt ("t_DHC_C_10_3.txt");

//-----------------------------------------------------------//
// ------------------------- MAIN ------------------------ //
int main (){

        //-----------------------------------------------------//
        // I - DATA INPUT:
        // -------------------- NUMERICAL ---------------------- //
        double delta_X, delta_Y;
        double t, error_s;
        int i,j;

        //-----------------------------------------------------//
        // II - PREVIOUS CALCULATIONS AND VECTOR DEFINITIONS:
        // ---------------------- VECTORS ---------------------- //
        // Geometry
        vector<double> xVC(M+2), yVC(M+2), x_N(N+2), y_N(N+2);
        // Initial velocity map
```

```
    vector<vector<double> > u0(N+2,vector<double>(M+2)), v0(N+2,vector<double
        >(M+2));
    // Velocity for t=n
    vector<vector<double> > vel_X(N+2,vector<double>(M+2)), vel_Y(N+2,vector<
        double>(M+2));
    // Velocity for t=n-1
    vector<vector<double> > vel_X_PRE(N+2,vector<double>(M+2)), vel_Y_PRE(N+2,
        vector<double>(M+2));
    // Velocity for t=n+1
    vector<vector<double> > vnext_u(N+2,vector<double>(M+2)), vnext_v(N+2,
        vector<double>(M+2));
    // Intermediate velocity - Predictor velocity
    vector<vector<double> > vP_u(N+2,vector<double>(M+2)), vP_v(N+2,vector<
        double>(M+2));
    // Pressure map for t=n+1
    vector<vector<double> > press(N+2,vector<double>(M+2));
    // Pressure map for t=n
    vector<vector<double> > press_PRE(N+2,vector<double>(M+2));
    // Temperature map for t=n+1
    vector<vector<double> > temp(N+2,vector<double>(M+2));
    // Temperature map for t=n
    vector<vector<double> > temp_PRE(N+2,vector<double>(M+2));
    // R for u and v, and t=n and t=n-1
    vector<vector<double> > R_u(N+2,vector<double>(M+2)), R_u_PRE(N+2,vector<
        double>(M+2)), R_v(N+2,vector<double>(M+2)), R_v_PRE(N+2,vector<double
        >(M+2));

    // ----------------- PREVIOUS CALCULATIONS ------------------ //
    // Mesh generation
    mesh_generation(x_N, xVC, y_N, yVC, delta_X, delta_Y);
    // Initial velocity and pressure maps
    initial_vel_map(u0, v0);
    press_map(press);
    // Initial temperature map
    temp_map(temp);
    // Initially, R(u for t=n-1) = 0 and R(v for t=n-1) = 0.
    R_initial(R_u_PRE, R_v_PRE);

    //----------------------------------------------------------//
    // Data update for t=n and t=n-1
    update(u0, vel_X);
    update(u0, vel_X_PRE);
    update(v0, vel_Y);
    update(v0, vel_Y_PRE);
    update(press, press_PRE);
    update(temp, temp_PRE);

    //----------------------------------------------------------//
    //             TIME-CONVERGENCE STARTING POINT
    //----------------------------------------------------------//
    t=0;
    error_s=99999;
    //
    while (error_s>0.00001){
            // CFL condition for time step
            min_time(delta_X, delta_Y, delta_t, vnext_u, vnext_v);
            t=t+delta_t;
            //delta_t = 0.0001;

            // R(u for t=n) and R(v for t=n) evaluation
            R_vel_u(x_N, xVC, y_N, yVC, R_u, vel_X, vel_Y);
            R_vel_v(temp_PRE, x_N, xVC, y_N, yVC, R_v, vel_X, vel_Y);

```

```
                // Intermediate velocity evaluation with real velocity for t=n and
                    R for t=n and t=n-1
                interm_vel(delta_t, vP_u, vel_X, R_u, R_u_PRE);
                interm_vel(delta_t, vP_v, vel_Y, R_v, R_v_PRE);

                // Pressure solver: Gauss-Seidel node-by-node
                resolution(delta_t, x_N, xVC, y_N, yVC, press, delta, vP_u, vP_v);

                // Next velocity evaluation
                next_vel_u(x_N, delta_t, vnext_u, press, vP_u);
                next_vel_v(y_N, delta_t, vnext_v, press, vP_v);

                // Next temperature evaluation
                resolution_temp(delta_t, x_N, xVC, y_N, yVC, temp_PRE, temp, delta
                    , vnext_u, vnext_u);

                // Time convergence evaluation
                maximum(press, press_PRE, error_s);

                // Data update for the next time step
                update(vnext_u, vel_X);
                update(vnext_v, vel_Y);
                update(vel_X, vel_X_PRE);
                update(vel_Y, vel_Y_PRE);
                update(R_u, R_u_PRE);
                update(R_v, R_v_PRE);
                update(press, press_PRE);
                update(temp, temp_PRE);

                cout<<t<<"  "<<error_s;
                cout<<endl;
        }

        //----------------------------------------------//
        // --------------- OUTPUT FILES --------------- //
        for (i=0; i<N+2; i++){
                outg<<x_N[i]<<"\t";
        }
        outg<<endl;
        for (j=0; j<M+2; j++){
                outg<<y_N[j]<<"\t";
        }
        outg<<endl;
        for(i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
                        outu<<vnext_u[i][j]<<"\t";
                }
                outu<<endl;
        }
        for(i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
                        outv<<vnext_v[i][j]<<"\t";
                }
                outv<<endl;
        }
        for(i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
                        outp<<press[i][j]<<"\t";
                }
                outp<<endl;
        }
        for(i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
```

```
                        outt<<temp[i][j]<<"\t";
                }
                outt<<endl;
        }
        //----------------------------------------------//
        //----------------------------------------------//
}

//--------------------------------------------------------------------//
// -------------------------- MAIN FUNCTIONS ------------------------- //
void mesh_generation(vector<double> &x_N, vector<double> &xVC, vector<double> &y_N
    , vector<double> &yVC, double& delta_X, double& delta_Y)
{
    int i, j;
        delta_X=L/N;                    delta_Y=H/M;

        // -------- x-mesh -------- //
        x_N[0]=0;                               xVC[0]=0;
        for (i=1; i<N+1; i++){
                x_N[i]= (i-0.5)*delta_X;
                xVC[i]= i*delta_X;
        }
    x_N[N+1]=L;                         xVC[N+1]=0;

    // -------- y-mesh -------- //
    y_N[0]=0;                       yVC[0]=0;
    for (j=1; j<M+1; j++){
                y_N[j]= (j-0.5)*delta_Y;
                yVC[j]= j*delta_Y;
        }
    y_N[M+1]=H;                         yVC[M+1]=0;
}

void initial_vel_map(vector<vector<double> > &u0, vector<vector<double> > &v0)
{
        int i, j;
        double u_inicial=0;
        double v_inicial=0;

        for (i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
                        if(j==M and i<N+1){
                                u0[i][M] = u_inicial;
                        }
                        else {
                                u0[i][j] = u_inicial;
                        }
        }
        }

        for (i=0; i<N+2; i++){
                for(j=0; j<M+1; j++){
                        v0[i][j] = v_inicial;
                }
                v0[i][M+1] = 0;
        }
}

void press_map(vector<vector<double> > &press)
{
        int i,j;

        // Initial arbitrary pressure
```

```
	for (i=0; i<N+2; i++){
		for(j=0; j<M+2; j++){
			press[i][j] = 0;
		}
	}
}

void temp_map(vector<vector<double> > &temp)
{
	int i,j;

	// Initial arbitrary temperature
	for (i=0; i<N+2; i++){
		for(j=0; j<M+2; j++){
			temp[i][j] = T0;
		}
	}

}

void update(vector<vector<double> > &mat1, vector<vector<double> > &mat2)
{
	int i,j;

	for (i=0; i<N+2; i++){
		for (j=0; j<M+2; j++){
			mat2[i][j] = mat1[i][j];
		}
	}
}

void R_initial(vector<vector<double> > &R_u, vector<vector<double> > &R_v)
{
	int i,j;

	for(i=0; i<N+2; i++ ){
		for(j=0; j<M+2; j++){
			R_u[i][j]=0;
			R_v[i][j]=0;
		}
	}
}

void R_vel_u(vector<double> &x_N, vector<double> &xVC, vector<double> &y_N, vector
    <double> &yVC, vector<vector<double> > &R_u, vector<vector<double> > &vel_X,
    vector<vector<double> > &vel_Y)
{
	int i, j;
	double me, mw, mn, ms;				// mass fluxes
	double ue, uw, un, us;				// velocities with CDS -
	    UDS - QUICK
	double Ae, Aw, An, As;				// Areas
	double dEP, dPW, dNP, dPS, vVC;			// Distances
	double fe, fw, fn, fs;				// fe = 0 if me>0

	double xU, xC, xD, xf;				// Positions
	double fiU, fiC, fiD;				// Phi values
	double fiC_ad, xC_ad, xf_ad;		// Adimensional values

	for (i=0; i<N+2; i++){
		for(j=0; j<M+2; j++){
			if(i>0 && i<N && j>1 && j<M){
				Ae = yVC[j+1]-yVC[j];				Aw = Ae;
```

```
                                An = x_N[i+1]-x_N[i];                   As = An;

                                me = rho*(vel_X[i+1][j]+vel_X[i][j])*Ae/2;
                                mw = rho*(vel_X[i][j]+vel_X[i-1][j])*Aw/2;
                                mn = rho*(vel_Y[i][j]*(xVC[i]-x_N[i]) + vel_Y[i
                                    +1][j]*(x_N[i+1]-xVC[i]) );
                                ms = rho*(vel_Y[i][j-1]*(xVC[i]-x_N[i]) + vel_Y[i
                                    +1][j-1]*(x_N[i+1]-xVC[i]) );

                                // CDS
                                ue = ((x_N[i+1]-xVC[i])/(xVC[i+1]-xVC[i]))*(vel_X[
                                    i+1][j]-vel_X[i][j])+vel_X[i][j];
                                uw = ((xVC[i]-x_N[i])/(xVC[i]-xVC[i-1]))*(vel_X[i
                                    -1][j]-vel_X[i][j])+vel_X[i][j];
                                un = ((yVC[j]-y_N[j])/(y_N[j+1]-y_N[j]))*(vel_X[i
                                    ][j+1]-vel_X[i][j])+vel_X[i][j];
                                us = ((y_N[j]-yVC[j-1])/(y_N[j]-y_N[j-1]))*(vel_X[
                                    i][j-1]-vel_X[i][j])+vel_X[i][j];

                                dEP = xVC[i+1]-xVC[i];                  dPW = xVC[
                                    i]-xVC[i-1];
                                dNP = y_N[j+1]-y_N[j];                  dPS = y_N[
                                    j]-y_N[j-1];
                                vVC = (yVC[j]-yVC[j-1])*(x_N[i+1]-x_N[i]);

                                R_u[i][j]=(1/vVC)*(-(me*ue-mw*uw+mn*un-ms*us)+mu
                                    *((vel_X[i+1][j]-vel_X[i][j])*Ae/dEP-(vel_X[i][
                                    j]-vel_X[i-1][j])*Aw/dPW +(vel_X[i][j+1]-vel_X[
                                    i][j])*An/dNP-(vel_X[i][j]-vel_X[i][j-1])*As/
                                    dPS));
                        }
                        else{
                                R_u[i][j]=0;
                        }
                }
        }
}

void R_vel_v(vector<vector<double> > &temp_PRE, vector<double> &x_N, vector<double
    > &xVC, vector<double> &y_N, vector<double> &yVC, vector<vector<double> > &R_v,
     vector<vector<double> > &vel_X, vector<vector<double> > &vel_Y)
{
        int i, j;
        double me, mw, mn, ms;                          // mass fluxes
        double ve, vw, vn, vs;                          // velocities with CDS -
            UDS - QUICK
        double Ae, Aw, An, As;                          // Areas
        double dEP, dPW, dNP, dPS, vVC;         // Distances
        double fe, fw, fn, fs;

        double xU, xC, xD, xf;                  // Positions
        double fiU, fiC, fiD;                   // Phi values
        double fiC_ad, xC_ad, xf_ad;    // Adimensional values

        for (i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
                        if(i>1 && i<N && j>0 && j<M){
                                Ae = y_N[j+1]-y_N[j];                   Aw = Ae;
                                An = xVC[i]-xVC[i-1];                   As = An;

                                me = rho*(vel_X[i][j]*(yVC[j]-y_N[j])+vel_X[i][j
                                    +1]*(y_N[j+1]-yVC[j]));
```

```
                                mw = rho*(vel_X[i-1][j]*(yVC[j]-y_N[j])+vel_X[i-1][j+1]*(y_N[j+1]-yVC[j]));
                                mn = rho*(vel_Y[i][j]+vel_Y[i][j+1])*An/2;
                                ms = rho*(vel_Y[i][j-1]+vel_Y[i][j])*As/2;

                                // CDS
                                ve = ((xVC[i]-x_N[i])/(x_N[i+1]-x_N[i]))*(vel_Y[i+1][j]-vel_Y[i][j])+vel_Y[i][j];
                                vw = ((x_N[i]-xVC[i-1])/(x_N[i]-x_N[i-1]))*(vel_Y[i-1][j]-vel_Y[i][j])+vel_Y[i][j];
                                vn = ((y_N[j+1]-yVC[j])/(yVC[j+1]-yVC[j]))*(vel_Y[i][j+1]-vel_Y[i][j])+vel_Y[i][j];
                                vs = ((yVC[j]-y_N[j])/(yVC[j]-yVC[j-1]))*(vel_Y[i][j-1]-vel_Y[i][j])+vel_Y[i][j];

                                dEP = x_N[i+1]-x_N[i];                  dPW = x_N[i]-x_N[i-1];
                                dNP = yVC[j+1]-yVC[j];                  dPS = yVC[j]-yVC[j-1];
                                vVC = (xVC[i]-xVC[i-1])*(y_N[j+1]-y_N[j]);

                                R_v[i][j]= -g*(1-beta*(temp_PRE[i][j] - T0)) + (1/vVC)*( -(me*ve-mw*vw+mn*vn-ms*vs)+mu*((vel_Y[i+1][j]-vel_Y[i][j])*Ae/dEP-(vel_Y[i][j]-vel_Y[i-1][j])*Aw/dPW +(vel_Y[i][j+1]-vel_Y[i][j])*An/dNP-(vel_Y[i][j]-vel_Y[i][j-1])*As/dPS));
                        }
                        else {
                                R_v[i][j]=0;
                        }
                }
        }
}

void interm_vel(double delta_t, vector<vector<double> > &vel_p, vector<vector<double> > &vel, vector<vector<double> > &R, vector<vector<double> > &R_PRE)
{
        int i, j;
        for (i=0; i<N+2; i++){
                for (j=0; j<M+2; j++){
                        //vel_p[i][j] = vel[i][j]+(delta_t/rho)*(1.5*R[i][j]-0.5*R_PRE[i][j]);
                        vel_p[i][j] = vel[i][j]+(delta_t/rho)*R[i][j];
                }
        }
}

void resolution(double delta_t, vector<double> &x_N, vector<double> &xVC, vector<double> &y_N, vector<double> &yVC, vector<vector<double> > &press, double delta, vector<vector<double> > &vP_u, vector<vector<double> > &vP_v)
{
        vector<vector<double> > press_est(N+2,vector<double>(M+2));
        double error;
        int i, j;


        double dpe, dpw, dpn, dps;
        double Ae, Aw, An, As;
        double aE, aW, aN, aS, aP, bP;

        // ------------ GAUSS-SEIDEL solver ------------ //
        error=1000;
        while (error>delta){
```

```

                update(press, press_est);
                // Internal nodes
                for(i=1; i<N+1; i++){
                        for (j=1; j<M+1; j++){

                                dpe = x_N[i+1]-x_N[i];          dpw = x_N[i]-x_N[i-1];
                                dpn = y_N[j+1]-y_N[j];          dps = y_N[j]-y_N[j-1];
                                Ae = yVC[j]-yVC[j-1];           Aw = Ae;
                                An = xVC[i]-xVC[i-1];           As = An;

                                aE = Ae/dpe;                            aW = Aw/dpw;
                                aN = An/dpn;                            aS = As/dps;
                                aP = aE + aN + aS + aW;
                                bP = (-rho/delta_t)*(vP_u[i][j]*Ae - vP_u[i-1][j]*Aw + vP_v[i][j]*An - vP_v[i][j-1]*As);

                                press[i][j] = (press_est[i+1][j]*aE+press_est[i-1][j]*aW+press_est[i][j+1]*aN+press_est[i][j-1]*aS+bP)/aP;
                        }
                }
                // Upper and Lower nodes
                for (i=1; i<N+1; i++){
                        press[i][0] = press[i][1];
                        press[i][M+1] = press[i][M];
                }
                // Right and left nodes
                for (j=1; j<M+1; j++){
                        press[0][j] = press[1][j];
                        press[N+1][j] = press[N][j];
                }
                // Corner nodes
                press[0][0] = press[1][1];                      press[0][M+1] = press[1][M];
                press[N+1][M+1] = press[N][M];                  press[N+1][0] = press[N][1];

                // Convergence analysis
                maximum(press, press_est, error);

        }
}

void resolution_temp(double delta_t, vector<double> &x_N, vector<double> &xVC, vector<double> &y_N, vector<double> &yVC, vector<vector<double> > &temp_PRE, vector<vector<double> > &temp, double delta, vector<vector<double> > &vnext_u, vector<vector<double> > &vnext_v)
{
        vector<vector<double> > temp_est(N+2,vector<double>(M+2));
        double error;
        int i, j;


        double dpe, dpw, dpn, dps;
        double me, mw, mn, ms;
        double fe, fw, fn, fs;
        double Ae, Aw, An, As;
        double aE, aW, aN, aS, aP, bP;
```

```

        // ------------ GAUSS-SEIDEL solver ------------ //
        error=1000;
        while (error>delta){

                update(temp, temp_est);
                // Internal nodes
                for(i=1; i<N+1; i++){
                        for (j=1; j<M+1; j++){

                                dpe = x_N[i+1]-x_N[i];          dpw = x_N[i]-x_N[i-1];
                                dpn = y_N[j+1]-y_N[j];          dps = y_N[j]-y_N[j-1];
                                Ae = yVC[j]-yVC[j-1];           Aw = Ae;
                                An = xVC[i]-xVC[i-1];           As = An;

                                me = vnext_u[i+1][j]*Ae*rho;    mw = -vnext_u[i][j]*Aw*rho;
                                mn = vnext_v[i][j+1]*An*rho;    ms = -vnext_v[i][j]*As*rho;

                                fe=0.5; fw=0.5; fn=0.5; fs=0.5;

                                aE = Ae/dpe - me*fe;            aW = Aw/dpw - mw*fw;
                                aN = An/dpn - mn*fn;            aS = As/dps - ms*fs;
                                aP = aE + aN + aS + aW + rho*dpe*dpn/delta_t;

                                bP = (rho*dpe*dpn/delta_t)*temp_PRE[i][j];

                                temp[i][j] = (temp_est[i+1][j]*aE+temp_est[i-1][j]*aW+temp_est[i][j+1]*aN+temp_est[i][j-1]*aS+bP)/aP;
                        }
                }
                // Upper and Lower nodes
                for (i=1; i<N+1; i++){
                        temp[i][0] = temp[i][1];
                        temp[i][M+1] = temp[i][M];
                }
                // Right and left nodes
                for (j=1; j<M+1; j++){
                        temp[0][j] = T0;
                        temp[N+1][j] = T0-T_var;
                }
                // Corner nodes
                temp[0][0] = temp[1][1];                temp[0][M+1] = temp[1][M];
                temp[N+1][M+1] = temp[N][M];            temp[N+1][0] = temp[N][1];

                // Convergence analysis
                maximum(temp, temp_est, error);


        }
}

void maximum(vector<vector<double> > &press, vector<vector<double> > &press_est, double& error)
{
        int i,j;
        double dif;
```

```

        error=0;
        for(i=0; i<N+2; i++){
                for (j=0; j<M+2; j++){
                        dif = abs(press[i][j]-press_est[i][j]);
                        if(dif>error){
                                error=dif;
                        }
                        else{
                                error=error;
                        }
                }
        }
}

void next_vel_u(vector<double> &x_N, double delta_t, vector<vector<double> > &
    vnext_u, vector<vector<double> > &press, vector<vector<double> > &vP_u)
{
        int i, j;
        double dpe;

        for (i=0; i<N+2; i++){
                for (j=0; j<M+2; j++){
                        if(i>0 && i<N && j>1 && j<M){
                                dpe = x_N[i+1]-x_N[i];
                                vnext_u[i][j] = vP_u[i][j]-(delta_t/rho)*((press[i
                                    +1][j]-press[i][j])/dpe);
                        }
                        else {
                                vnext_u[i][j] = vP_u[i][j];
                        }
                }
        }
}

void next_vel_v(vector<double> &y_N, double delta_t, vector<vector<double> > &
    vnext_v, vector<vector<double> > &press, vector<vector<double> > &vP_v)
{
        int i,j;
        double dpn;

        for (i=0; i<N+2; i++){
                for(j=0; j<M+2; j++){
                        if(i>1 && i<N && j>0 && j<M){
                                dpn = y_N[j+1]-y_N[j];
                                vnext_v[i][j] = vP_v[i][j]-(delta_t/rho)*((press[i
                                    ][j+1]-press[i][j])/dpn);
                        }
                        else{
                                vnext_v[i][j] = vP_v[i][j];
                        }
                }
        }
}

void min_time(double delta_X, double delta_Y, double& delta_t, vector<vector<
    double> > &vnext_u, vector<vector<double> > &vnext_v)
{
        double t_cX, t_cY, t_dX, t_dY, t_TX, t_TY, v_modul;
        int i,j;

        delta_t=999999;

```

```
        for(i=1; i<=N-1; i++){
                for(j=1; j<=M-1; j++){
                        v_modul=sqrt(pow(vnext_u[i][j],2)+pow(vnext_v[i][j],2));
                        t_cX=0.35*delta_X/v_modul;
                        t_cY=0.35*delta_Y/v_modul;
                        t_dX=0.2*pow(delta_X, 2)*rho/mu;
                        t_dY=0.2*pow(delta_Y, 2)*rho/mu;
                        t_TX=0.2*pow(delta_X,2)/(lambda/(rho*cp));
                        t_TY=0.2*pow(delta_Y,2)/(lambda/(rho*cp));
                        if (t_cX<delta_t){
                                delta_t = t_cX;
                        }
                        else{
                                delta_t = delta_t;
                        }
                        if (t_cY<delta_t){
                                delta_t = t_cY;
                        }
                        else {
                                delta_t = delta_t;
                        }
                        if (t_dX<delta_t){
                                delta_t = t_dX;
                        }
                        else {
                                delta_t = delta_t;
                        }
                        if (t_dY<delta_t){
                                delta_t = t_dY;
                        }
                        else {
                                delta_t = delta_t;
                        }
                        if (t_TX<delta_t){
                                delta_t = t_TX;
                        }
                        else {
                                delta_t = delta_t;
                        }
                        if (t_TY<delta_t){
                                delta_t = t_TY;
                        }
                        else {
                                delta_t = delta_t;
                        }
                }
        }
}
```

Annex F

Square Cylinder code

NOTE: This code is the same used to solve the Flow between Parallel Plates problem and its verification cases. By changing the solid dimensions in the ***xmesh*** *and* ***ymesh*** *functions, these other cases can be solved.*

```
//-----------------------------------------------------------//
//                                                           //
//          Study for the computational resolution of        //
//     conservation equations of mass, momentum and energy   //
//          Numerical analysis and turbulence models         //
//                                                           //
//-----------------------------------------------------------//
// Author: Gerard Toset Alonso                               //
// Date: June, 2023                                          //
// Code: Chapter 5 - FSM - Square cylinder                   //
//-----------------------------------------------------------//
//                    GENERAL DESCRIPTION                    //
// - SOLVER : Gauss-Seidel                                   //
// - Transient regime                                        //
// - 2D spatial discretization - Cell-centered nodes         //
// - CDS                                                     //
//-----------------------------------------------------------//

#include <iostream>
#include <math.h>
#include <cmath>
#include <fstream>
#include <vector>
using namespace std;

//-----------------------------------------------------------//
// ------------------ Constant definitions ----------------- //
const double L = 50;
const double H = 8;
const double l = 12.5;
const double E = 1;
const double D = 1;
const double rho = 1;
const double mu = 1;
const double u_ref = 1;

const int N=100;             // Number of CVs in X direction
const int M=50;              // Number of CVs in Y direction
double delta=10e-6;          // Convergence parameter
```

```

//------------------------------------------------------------//
// ----------------- Functions definition ------------------- //
typedef vector <double> vect;
typedef vector <vector<double> > mat;

void xmesh(vect &xN, vect &xVC, double &delta_X, int &imin, int &imax);
void ymesh(vect &yN, vect &yVC, double &delta_Y, int &jmin, int &jmax);
void initial_vel_map_u(mat &u, mat &up, mat &Rx, vect &yN);
void initial_vel_map_v(mat &v, mat &vp, mat &Ry);
void press_map(mat &P);
void velP_u(int &imin, int &imax, int &jmin, int &jmax, mat &up, mat &Rx, mat &u, mat &v, vect &xN, vect &yN, vect &xVC, vect &yVC, double delta_t);
void velP_v(int &imin, int &imax, int &jmin, int &jmax, mat &vp, mat &Ry, mat &u, mat &v, vect &xN, vect &yN, vect &xVC, vect &yVC, double delta_t);
void coeff_a(mat &ae, mat &aw, mat &an, mat &as, vect &xN, vect &yN, vect &xVC, vect &yVC);
void coeff_b(mat &bp, mat &up, mat &vp, vect &xN, vect &yN, vect &xVC, vect &yVC, double delta_t);
void resolution(int &imin, int &imax, int &jmin, int &jmax, mat &P, mat &ae, mat &aw, mat &an, mat &as, mat &bp);
void next_vel_u(int &imin, int &imax, int &jmin, int &jmax, mat &u, mat &up, mat &P, vect &xN, double delta_t);
void next_vel_v(int &imin, int &imax, int &jmin, int &jmax, mat &v, mat &vp, mat &P, vect &yN, double delta_t);
void convergence(mat &P, mat &P_PRE, bool &conv);
void update(mat &ANT, mat &NEW);
void abs_vel(mat &v_abs, mat &u, mat &v);
void min_time(mat &v_abs, double &delta_t);

ofstream outxn("xN_R1.txt");
ofstream outyn("yN_R1.txt");
ofstream outxvc("xVC_R1.txt");
ofstream outyvc("yVC_R1.txt");
ofstream outu("u_R1.txt");
ofstream outv("v_R1.txt");
ofstream outp("P_R1.txt");
ofstream outvabs("v_abs_R1.txt");

//------------------------------------------------------------//
//--------------------------- MAIN --------------------------//
int main(){

        //-------------------------------------------------------//
        // I - DATA INPUT:
        // -------------------- NUMERICAL ---------------------- //
        int i, j, imin, imax, jmin, jmax;
        double delta_X, delta_Y, delta_t,t;
        bool conv;

        //-------------------------------------------------------//
        // II - PREVIOUS CALCULATIONS AND VECTOR DEFINITIONS:
        // ---------------------- VECTORS ---------------------- //
        // Geometry
        vect xN = vect(N);                                vect xVC = vect(N+1);
        vect yN = vect(M);                                vect yVC = vect(M+1);
        // Pressure, velocities, predictor velocities and R(v) matrices
        mat P = mat(N, vect(M));                  mat P_PRE = mat(N, vect(M));

        mat u = mat(N+1, vect(M));      mat v = mat(N, vect(M+1));
        mat up = mat(N+1, vect(M));     mat vp = mat(N, vect(M+1));
        mat Rx = mat(N+1, vect(M));     mat Ry = mat(N, vect(M+1));
        // Discretization coefficients
```

```
        mat ae = mat(N, vect(M));                mat aw = mat(N, vect(M));
        mat as = mat(N, vect(M));                mat an = mat(N, vect(M));
        mat bp = mat(N, vect(M));
        // CV areas
        mat aE = mat(N, vect(M));                mat aW = mat(N, vect(M));
        mat aS = mat(N, vect(M));                mat aN = mat(N, vect(M));
        // Absolute velocity matrix
        mat v_abs = mat(N, vect(M));

        // ----------------- PREVIOUS CALCULATIONS ------------------ //
        // Mesh generation
        xmesh(xN, xVC, delta_X, imin, imax);
        ymesh(yN, yVC, delta_Y, jmin, jmax);

        cout<<imin<<" "<<imax<<" "<<jmin<<" "<<jmax<<endl;
        cin>>i;

        // Initial velocity and pressure maps
        initial_vel_map_u(u, up, Rx, yN);
        initial_vel_map_v(v, vp, Ry);
        press_map(P);

        // Constant coefficient evaluation
        coeff_a(ae, aw, an, as, xN, yN, xVC, yVC);

        //-----------------------------------------------------------//
        //              TIME-CONVERGENCE STARTING POINT
        //-----------------------------------------------------------//
        t=0;
        conv=false;

        while (conv == false){

                // CFL condition for time step
                min_time(v_abs, delta_t);
                t=t+delta_t;

                //-----------------------------------------------------------//
                // Data update
                update(P_PRE, P);

                // Intermediate velocity evaluation with real velocity for t=n and R for t=n and t=n-1
                velP_u(imin, imax, jmin, jmax, up, Rx, u, v, xN,  yN, xVC, yVC, delta_t);
                velP_v(imin, imax, jmin, jmax, vp, Ry, u, v, xN,  yN, xVC, yVC, delta_t);

                // bP coefficient evaluation with the new time step
                coeff_b(bp, up, vp,  xN, yN, xVC, yVC, delta_t );

                // Pressure solver: Gauss-Seidel node-by-node
                resolution(imin, imax, jmin, jmax, P, ae, aw, an, as, bp);

                // Next velocity evaluation
                next_vel_u(imin, imax, jmin, jmax, u, up, P, xN, delta_t);
                next_vel_v(imin, imax, jmin, jmax, v, vp, P, yN, delta_t);

                // Time convergence evaluation
                convergence(P, P_PRE, conv);

                // Absolute velocity computation for the new CFL delta_T value
                abs_vel(v_abs, u, v);
```

```
            cout<<t<<endl;
    }


    //----------------------------------------------//
    // --------------- OUTPUT FILES --------------- //
    for(i=0; i<N; i++){
            outxn<<xN[i]<<" ";
    }
    outxn<<endl;
    for(i=0; i<N; i++){
            outxn<<xN[i]<<" ";
    }
    outxn<<endl;

    for  (j=M-1; j>=0; j--){
            outyn<<yN[j]<<" ";
    }
    outyn<<endl;
    for  (j=M-1; j>=0; j--){
            outyn<<yN[j]<<" ";
    }
    outyn<<endl;

    for(i=0; i<=N; i++){
            outxvc<<xVC[i]<<" ";
    }
    outxvc<<endl;
    for(i=0; i<=N; i++){
            outxvc<<xVC[i]<<" ";
    }
    outxvc<<endl;

    for  (j=M; j>=0; j--){
            outyvc<<yVC[j]<<" ";
    }
    outyvc<<endl;
    for  (j=M; j>=0; j--){
            outyvc<<yVC[j]<<" ";
    }
    outyvc<<endl;

    for (j=M-1; j>=0; j--){
            for(i=0; i<=N; i++){
                    outu<<u[i][j]<<"\t";
            }
            outu<<endl;
    }

    for (j=M; j>=0; j--){
            for(i=0; i<N; i++){
                    outv<<v[i][j]<<"\t";
            }
            outv<<endl;
    }

    for (j=M-1; j>=0; j--){
            for(i=0; i<N; i++){
                    outp<<P[i][j]<<"\t";
            }
            outp<<endl;
    }

```

```
        for (j=M-1; j>=0; j--){
                for(i=0; i<N; i++){
                        outvabs<<v_abs[i][j]<<"\t";
                }
                outvabs<<endl;
        }
        //----------------------------------------------//
        //----------------------------------------------//

}

//----------------------------------------------------------------------//
// ------------------------- MAIN FUNCTIONS ------------------------- //
void xmesh(vect &xN, vect &xVC, double &delta_X, int &imin, int &imax)
{
        int i;
        delta_X=L/N;

        for (i=0; i<N; i++){
                xN[i]= (i+0.5)*delta_X-l;
                xVC[i]= (i)*delta_X-l;
        }
        xVC[N]=L-l;

        i=0;
    while (xN[i] <= -D/2){
        imin=i;
        i++;
    }
        i=0;
        while (xN[i] <= D/2){
        imax=i;
                i++;
    }

}

void ymesh( vect &yN, vect &yVC, double &delta_Y, int &jmin, int &jmax)
{
        int j;
        delta_Y=H/M;
    for (j=0; j<M; j++){
                yN[j]= (j+0.5)*delta_Y-0.5*H;
                yVC[j]=(j)*delta_Y-0.5*H;
        }
    yVC[M]=0.5*H;

    j=0;
    while (yN[j] < -D/2){
        jmin = j;
        j++;
    }
        j=0;
        while (yN[j] < D/2){
        jmax = j;
        j++;
        }
}

void initial_vel_map_u(mat &u, mat &up, mat &Rx, vect &yN)
{
        int i, j;
        double y, h;
```

```

        for (i=0; i<=N; i++){
                for (j=0; j<M; j++){

                        u[i][j]=0;
                        up[i][j]=0;
                        Rx[i][j]=0;


                        if(i==0){
                                y=yN[j]-yN[0];
                                h=yN[M-1]-yN[0];
                                u[i][j]=(-4*u_ref/(h*h))*(y*y)+(4*u_ref/h)*y;
                        }

                }
        }

}

void initial_vel_map_v(mat &v, mat &vp, mat &Ry)
{
        int i,j;

        for (i=0; i<N; i++){
                for (j=0; j<=M; j++){
                        v[i][j]=0;
                        vp[i][j]=0;
                    Ry[i][j]=0;
                }
        }

}

void press_map(mat &P)
{
        int i, j;

        for (i=0; i<N; i++){
                for (j=0; j<M; j++){
                        P[i][j]=0;
                }
        }

}

void velP_u(int &imin, int &imax, int &jmin, int &jmax, mat &up, mat &Rx, mat &u,
    mat &v, vect &xN, vect &yN, vect &xVC, vect &yVC, double delta_t)
{
        int i,j;
        double me, mw, mn, ms;
        double ue, uw, un, us;
        double Ae, Aw, An, As;
        double dEP, dWP, dNP, dSP, vVC;
        double uE, uW, uN, uS, uP;

        for (i=0; i<=N; i++){
                for (j=0; j<M; j++){
                        if(i>0 && i<N && j>0 && j<M-1){
                                if(i>imin && i<=imax+1 && j>=jmin && j<=jmax+1){
    // SOLID NODES
                                        up[i][j]=0;
                                        Rx[i][j]=0;
```

```
                                }
                                        else{ // FLUID NODES

                                                Ae=yVC[j+1]-yVC[j];   Aw=Ae;
                                                An=xN[i]-xN[i-1];   As=An;

                                                me=rho*(u[i+1][j]+u[i][j])*Ae/2;
                                                mw=rho*(u[i][j]+u[i-1][j])*Aw/2;
                                                mn=rho*(v[i][j+1]*(xN[i]-xVC[i])+v[i-1][j
                                                    +1]*(xVC[i]-xN[i-1]));
                                                ms=rho*(v[i][j]*(xN[i]-xVC[i])+v[i-1][j]*(
                                                    xVC[i]-xN[i-1]));

                                                ue=((xN[i]-xVC[i])/(xVC[i+1]-xVC[i]))*(u[i
                                                    +1][j]-u[i][j])+u[i][j];
                                                uw=((xVC[i]-xN[i-1])/(xVC[i]-xVC[i-1]))*(u
                                                    [i-1][j]-u[i][j])+u[i][j];
                                                un=((yVC[j+1]-yN[j])/(yN[j+1]-yN[j]))*(u[i
                                                    ][j+1]-u[i][j])+u[i][j];
                                                us=((yN[j]-yVC[j])/(yN[j]-yN[j-1]))*(u[i][
                                                    j-1]-u[i][j])+u[i][j];

                                                dEP=xVC[i+1]-xVC[i];            dWP=xVC[i
                                                    ]-xVC[i-1];
                                                dNP=yN[j+1]-yN[j];
                                                    dSP=yN[j]-yN[j-1];

                                                vVC=Ae*An;

                                                uE=u[i+1][j];  uW=u[i-1][j];
                                                uN=u[i][j+1];  uS=u[i][j-1];
                                                uP=u[i][j];

                                                Rx[i][j]=(1/vVC)*(-(me*ue-mw*uw+mn*un-ms*
                                                    us)+mu*((uE-uP)*Ae/dEP-(uP-uW)*Aw/dWP+(
                                                    uN-uP)*An/dNP-(uP-uS)*As/dSP));
                                                //up[i][j]=u[i][j]+(delta_t/rho)*(1.5*R
                                                    -0.5*Rx[i][j]);
                                                up[i][j]=u[i][j]+(delta_t/rho)*(Rx[i][j]);
                                        }

                                }
                                else if(i==0){
                                        up[i][j] = u[i][j];
                                }
                                else if(i==N){
                        up[i][j] = up[i-1][j];
                }
                                else {
                                        up[i][j] = 0;
                                }

                        }
                }

}

void velP_v(int &imin, int &imax, int &jmin, int &jmax, mat &vp, mat &Ry, mat &u,
    mat &v, vect &xN, vect &yN, vect &xVC, vect &yVC, double delta_t)
{
        int i,j;
        double me, mw, mn, ms;
        double ue, uw, un, us;
```

```
        double Ae, Aw, An, As;
        double dEP, dWP, dNP, dSP, vVC;
        double uE, uW, uN, uS, uP;

        for (i=0; i<N; i++){
                for (j=0; j<=M; j++){
                        if(i>0 && i<N-1 && j>0 && j<M){
                                if(i>=imin && i<=imax+1 && j>jmin && j<=jmax+1){
                                    // SOLID NODES
                                        vp[i][j]=0;
                                        Ry[i][j]=0;
                            }
                            else{  // FLUID NODES

                                        Ae=yN[j]-yN[j-1];  Aw=Ae;
                                        An=xVC[i+1]-xVC[i];  As=An;

                                        me=rho*(u[i+1][j]*(yN[j]-yVC[j])+u[i+1][j-1]*(yVC[j]-yN[j-1]));
                                        mw=rho*(u[i][j]*(yN[j]-yVC[j])+u[i][j-1]*(yVC[j]-yN[j-1]));
                                        mn=rho*An*(v[i][j]+v[i][j+1])/2;
                                        ms=rho*As*(v[i][j]+v[i][j-1])/2;

                                        ue=((xVC[i+1]-xN[i])/(xN[i+1]-xN[i]))*(v[i+1][j]-v[i][j])+v[i][j];
                                        uw=((xN[i]-xVC[i])/(xN[i]-xN[i-1]))*(v[i-1][j]-v[i][j])+v[i][j];
                                        un=((yN[j]-yVC[j])/(yVC[j+1]-yVC[j]))*(v[i][j+1]-v[i][j])+v[i][j];
                                        us=((yVC[j]-yN[j-1])/(yVC[j]-yVC[j-1]))*(v[i][j-1]-v[i][j])+v[i][j];

                                        dEP=xN[i+1]-xN[i];    dWP=xN[i]-xN[i-1];
                                        dNP=yVC[j+1]-yVC[j];            dSP=yVC[j]-yVC[j-1];

                                    vVC=Ae*An;

                                        uE=v[i+1][j];  uW=v[i-1][j];
                                        uN=v[i][j+1];  uS=v[i][j-1];
                                        uP=v[i][j];

                                        Ry[i][j]=(1/vVC)*(-(me*ue-mw*uw+mn*un-ms*us)+mu*((uE-uP)*Ae/dEP-(uP-uW)*Aw/dWP+(uN-uP)*An/dNP-(uP-uS)*As/dSP));
                                        //vp[i][j]=v[i][j]+(delta_t/rho)*(1.5*R-0.5*Ry[i][j]);
                                        vp[i][j]=v[i][j]+(delta_t/rho)*(Ry[i][j]);
                                }
                        }
                        else if(i==N-1){
                        vp[i][j] = vp[i-1][j];
                        }
                        else{
                                        vp[i][j] = 0;
                                }
                }
        }
}

void coeff_a(mat &ae, mat &aw, mat &an, mat &as, vect &xN, vect &yN, vect &xVC,
```

```
    vect &yVC)
{
        int i,j;

        for (i=0; i<N; i++){
                for (j=0; j<M; j++){
                        if(i<N-1){
                                ae[i][j]=(yVC[j+1]-yVC[j])/(xN[i+1]-xN[i]);
                        }
                        if(i>0){
                                aw[i][j]=(yVC[j+1]-yVC[j])/(xN[i]-xN[i-1]);
                        }
                        if(j<M-1){
                                an[i][j]=(xVC[i+1]-xVC[i])/(yN[j+1]-yN[j]);
                        }
                        if(j>0){
                                as[i][j]=(xVC[i+1]-xVC[i])/(yN[j]-yN[j-1]);
                        }
                }
        }
}

void coeff_b(mat &bp, mat &up, mat &vp, vect &xN, vect &yN, vect &xVC, vect &yVC,
    double delta_t)
{
        int i, j;
        double Ae, Aw, An, As;
        double ue, uw, us, un;

        for (i=0; i<N; i++){
                for (j=0; j<M; j++){

                        Ae=yVC[j+1]-yVC[j];  Aw=Ae;
                        An=xVC[i+1]-xVC[i];  As=An;

                        ue=up[i+1][j];  uw=up[i][j];
                        un=vp[i][j+1];  us=vp[i][j];

                        bp[i][j]=(-rho/delta_t)*(ue*Ae-uw*Aw+un*An-us*As);
                }
        }

}

void resolution(int &imin, int &imax, int &jmin, int &jmax, mat &P, mat &ae, mat &
    aw, mat &an, mat &as, mat &bp)
{
        int i,j;
        bool conv;
        mat P_est=mat(N, vect(M));

        conv=false;

        while(conv==false){
                for (i=0; i<N; i++){
                        for (j=0; j<M; j++){
                                double aE=0;  double aW=0;
                                double aN=0;  double aS=0;
                                double aP=0;
                                double PE=0;  double PW=0;
                                double PN=0;  double PS=0;

                                P_est[i][j]=P[i][j];
```

```
                                        if(i!=0){
                                    aW = aw[i][j];
                                    PW = P[i-1][j];
                                }
                                if(i!=N-1){
                                    aE = ae[i][j];
                                    PE = P[i+1][j];
                                }
                                if(j!=0){
                                    aS = as[i][j];
                                    PS = P[i][j-1];
                                }
                                if(j!=M-1){
                                    aN = an[i][j];
                                    PN = P[i][j+1];
                                }
                                        // SOLID WALL NODES
                                    if(i==imax+1 && j>jmin && j<=jmax){
                                                aW = 0;
                                        }
                                        if(i==imin && j>jmin && j<=jmax){
                                                aE = 0;
                                        }
                                    if(j==jmax+1 && i>imin && i<=imax){
                                                aS = 0;
                                        }
                                        if(j==jmin && i>imin && i<=imax){
                                                aN = 0;
                                        }
                                        if(i==N-1){
                                        P[i][j]=0;
                                }
                                else if(i>imin && i<=imax && j>jmin && j<=jmax){ // SOLID
                                    NODES
                                                P[i][j]=0;
                                        }
                                else{ // FLUID NODES
                                            aP =aE+aW+aN+aS;
                                        P[i][j] =(aE*PE+aW*PW+aN*PN+aS*PS+bp[i][j])/aP;
                                    }
                                }
                        }
                        convergence(P,P_est,conv);
                }
}

void convergence(mat &P, mat &P_PRE, bool &conv)
{
        int i, j;
        double dif, Maxdif;
        Maxdif=0;
        for (i=0; i<N; i++){
                for(j=0; j<M; j++){
                        dif=abs(P[i][j]-P_PRE[i][j]);

                        if (dif > Maxdif){
                                Maxdif=dif;
                        }
                        else{
                                Maxdif = Maxdif;
                        }
                }
        }
```

```
        if(Maxdif < delta){
                conv=true;
        }
        else{
                conv=false;
        }
}

void next_vel_u(int &imin, int &imax, int &jmin, int &jmax, mat &u, mat &up, mat &
    P, vect &xN, double delta_t)
{
        int i,j;
        for (i=0; i<=N; i++){
                for (j=0; j<M; j++){
                        if(i>0 && i<N && j>0 && j<M-1){
                                if(i>imin && i<=imax+1 && j>=jmin && j<=jmax+1){
                                    // SOLID NODES
                                        u[i][j]=0;
                            }
                                else{ // FLUID NODES
                                        u[i][j]=up[i][j]-(delta_t/rho)*((P[i][j]-P
                                            [i-1][j])/(xN[i]-xN[i-1]));
                                }
                        }
                        else if(i==0){
                                u[i][j]=u[i][j];
                        }
                        else if(i==N){
                                u[i][j]=u[i-1][j];
                        }
                        else {
                                u[i][j]=0;
                        }
                }
        }
}

void next_vel_v(int &imin, int &imax, int &jmin, int &jmax, mat &v, mat &vp, mat &
    P, vect &yN, double delta_t)
{
        int i,j;
        for (i=0; i<N; i++){
                for (j=0; j<=M; j++){
                        if(i>0 && i<N-1 && j>0 && j<M){
                                if(i>=imin && i<=imax+1 && j>jmin && j<=jmax+1){
                                    // SOLID NODES
                                        v[i][j]=0;
                            }
                            else{ // FLUID NODES
                                        v[i][j]=vp[i][j]-(delta_t/rho)*((P[i][j]-P
                                            [i][j-1])/(yN[j]-yN[j-1]));
                            }
                        }
                        else if(i==N-1){
                                v[i][j]=v[i-1][j];
                        }

                        else {
                                v[i][j]=0;
                        }
                }
        }
}
```

```

void update(mat &ANT, mat &NEW)
{
        int i, j;
        for (i=0; i<N; i++){
            for (j=0; j<M; j++){
                        ANT[i][j]=NEW[i][j];
                }
        }
}

void abs_vel(mat &v_abs, mat &u, mat &v)
{
        int i, j;
        double u_mean, v_mean;
        for (i=0; i<N; i++){
                for (j=0; j<M; j++){
                        u_mean=(u[i][j]+u[i+1][j])/2;
                        v_mean=(v[i][j]+v[i][j+1])/2;

                        v_abs[i][j]=sqrt(pow(u_mean,2)+pow(v_mean,2));
                }
        }
}

void min_time(mat &v_abs, double &delta_t)
{
        int i, j;
        double delta_X,delta_Y, t1, t2, t3, t4, tc;
        delta_X=L/N;
        delta_Y=H/M;
        tc=100;
        for (i=0; i<N; i++){
                for (j=0; j<M; j++){
                        t1=0.35*delta_X/v_abs[i][j];
                        t2=0.35*delta_Y/v_abs[i][j];
                    t3=0.2*delta_X*delta_X*rho/mu;
                        t4=0.2*delta_Y*delta_Y*rho/mu;

                        if(t1<tc){
                                tc=t1;
                        }
                        if(t2<tc){
                                tc=t2;
                        }
                        if(t3<tc){
                                tc=t3;
                        }
                        if(t4<tc){
                                tc=t4;
                        }
                }
        }
        delta_t=tc;
}
```

Annex G

Burgers equation code

```
//-----------------------------------------------------------//
//                                                           //
//         Study for the computational resolution of         //
//     conservation equations of mass, momentum and energy   //
//         Numerical analysis and turbulence models          //
//                                                           //
//-----------------------------------------------------------//
// Author: Gerard Toset Alonso                               //
// Date: June, 2023                                          //
// Code: Chapter 6 - Burger's equation in Fourier Space      //
//-----------------------------------------------------------//

#include <iostream>
#include <fstream>
#include <complex>
#include <cmath>
#include <math.h>

using namespace std;

//-----------------------------------------------------------//
// ------------------ Constant definitions ----------------- //
const int N = 110;                                   // number of modes
const double Re = 200;                               // Reynolds
const double Ck = 0.05;                              // LES parameter :
    KOLMOGOROV constant
const double m = 2;                                  // LES parameter : SLOPE

const double delta = 1e-7;                           // Convergence parameter
const double t_final = 1e6;                          // Final time

const double C1 = 0.001;                             // Relaxation factor for
    Delta_t
double delta_t = C1*Re/ pow(N,2);                 // Delta_t by CFL condition

const int LES = 0;
// Large -Eddy - Simulation . (0) No , (1) True

//-----------------------------------------------------------//
// ---------------- Functions definition ----------------- //
void initial_map (complex <double> u_nprev[2*N+1], complex <double> u_n[2*N+1]);
void Rn_evaluation (complex <double> R_n[2*N+1], complex <double> u_n[2*N+1]);
```

```
void next_u (complex <double> u_nm1[2*N+1], complex <double> u_n[2*N+1], double
    delta_t, complex <double> R_nprev[2*N+1], complex <double> R_n[2*N+1]);
void convergence (complex <double> u_nm1[2*N+1], complex <double> u_n[2*N+1], bool
    & steady_conv, double delta);
void update (complex <double> u_nm1[2*N+1], complex <double> u_n[2*N+1], complex <
    double> R_nprev[2*N+1], complex <double> R_n[2*N+1]);
void Ek_evaluation (complex <double> u_nm1[2*N+1], double Ek[N]); // Kinetic
    energy calculation

ofstream outK("k_values.txt");
ofstream outE("kinetic_energy.txt");

//-----------------------------------------------------------//
//--------------------------- MAIN --------------------------//

int main(){

        //-----------------------------------------------------//
        // I - DATA INPUT:
        // -------------------- NUMERICAL ---------------------- //
        double t = 0;
        bool steady_conv = 0;                              // Steady - state
            parameter

        //-----------------------------------------------------//
        // II - COMPLEX VECTOR DEFINITIONS:
        // ---------------------- VECTORS ---------------------- //
        // Velocities for t=n-1, t=n and t=n+1
        complex <double > u_nprev [2*N+1] , u_n [2*N+1] , u_nm1 [2*N+1];
        // R for t=n and t=n-1
        complex <double> R_nprev [2*N+1] , R_n [2*N+1];
        // Kinetic energy vector
        double Ek[N];
        // ---------------------------------------------------------- //
        // INITIALIZATION for u_k and R(u_k^n-1)
        initial_map (u_nprev, u_n);
        Rn_evaluation( R_nprev, u_nprev);


        //---------------------------------------------------------//
        //            TIME-CONVERGENCE STARTING POINT
        //---------------------------------------------------------//
        t = t+delta_t;

        while (t<=t_final && steady_conv == false){

                // R(u for t=n) evaluation
                Rn_evaluation( R_n, u_n);

                // Next velocity evaluation
                next_u (u_nm1, u_n, delta_t, R_nprev, R_n);

                // Time convergence evaluation
                convergence (u_nm1, u_n, steady_conv, delta);
                if(steady_conv == false){
                        t = t+delta_t ;
                        update (u_nm1, u_n, R_nprev, R_n);
                        cout<<t<<"\t"<<steady_conv<<endl;
                }
        }

        //-----------------------------------------------------//
        // III - FINAL CALCULATIONS:
```

```
        Ek_evaluation (u_nm1, Ek);

        //-------------------------------------------------------//
        // IV - RESULTS TO .txt FILES:
        for(int k=1; k<N+1; k++){
                outK<<k<<endl;
        }
        for (int k = 1; k <= N; k++){
                outE<<Ek[k-1]<<endl;
        }
        //-------------------------------------------------------//

}

//----------------------------------------------------------------------//
// -------------------------- MAIN FUNCTIONS ------------------------- //
void initial_map ( complex < double > u_nprev [2*N+1] , complex < double > u_n [2*
    N+1])
{
        int k, k_new, q, q_new, p, p_new;

        for(k=-N; k <N+1; k++){
                k_new = k+N;

                if(k == 0){
                        u_nprev[k_new] = (0,0);
                        u_n[k_new] = (0,0);
                }
                else{
                        u_nprev[k_new] = 1/fabs(double(k));
                        u_n[k_new] = 1/fabs(double(k));
                }
        }

}

void Rn_evaluation (complex <double> R_n[2*N+1], complex <double> u_n[2*N+1])
{
        int k, k_new, q, q_new, p, p_new;
        double nu_t_ast , nu_t_inf , nu_t;
        complex <double > diff;
        complex <double > i (0,1);

        if(LES == 0){
                for (k=-N; k<N+1; k++){
                        k_new = k+N;

                        diff = -pow(double(k),2)*u_n[k_new]/Re;

                        complex <double > conv = (0,0);
                        for (q=-N; q<N+1; q++){
                                p = k-q;
                                if(p>=-N && p<=N){
                                        q_new = q+N;
                                        p_new = p+N;

                                        conv = conv + u_n[q_new]*u_n[p_new]*double
                                            (q)*i;
                                }
                        }
                        R_n[k_new] = diff - conv;
                }
        }
```

```
	else{		// LES == 1

		for (k=-N; k<N+1; k++){
			k_new = k+N;

			if(k != 0){
				nu_t_ast = 1+34.5*exp(-3.03*(N/fabs(double(k))));
			}
			else{
				nu_t_ast = 1;
			}

			nu_t_inf = 0.31*( (5-m)/(m+1) )*sqrt(3-m)*pow(Ck,-1.5);
			nu_t = nu_t_inf*sqrt((abs(u_n[2*N]*conj(u_n[2*N])))/N)*nu_t_ast;

			diff = -pow(double(k),2)*u_n[k_new]*(1/Re+nu_t);

			complex <double > conv = (0,0);
			for(q=-N; q<N+1; q++){
				p = k-q;
				if(p>=-N && p<=N){
					q_new = q+N;
					p_new = p+N;

					conv = conv + u_n[q_new]*u_n[p_new]*double(q)*i;
				}
			}

			R_n[k_new] = diff - conv ;
		}
	}
}

void next_u ( complex <double> u_nm1[2*N+1], complex <double> u_n[2*N+1], double delta_t, complex <double> R_nprev[2*N+1], complex <double> R_n[2*N+1])
{
	int k, k_new;
	for(k=-N; k<N+1; k++){
		k_new = k+N;

		if(k != -1 && k != 1){
			u_nm1[k_new] = u_n[k_new]+ delta_t*(1.5*R_n[k_new]-0.5*R_nprev[k_new]);
		}
		else{
			u_nm1[k_new] = u_n[k_new];
		}
	}
}

void convergence ( complex <double> u_nm1[2*N+1], complex <double> u_n[2*N+1], bool& steady_conv, double delta)
{
	bool real_conv = 0;
	bool imag_conv = 0;
	int k, k_new;

	double max_dif = 0;
	double max_dif_real, max_dif_imag;

```

```
        for (k=-N; k<N+1; k++){
                k_new = k+N;

                max_dif_real = fabs( real(u_nm1[k_new])-real(u_n[k_new]) );
                if(max_dif_real >= max_dif){
                        max_dif = max_dif_real;
                }
        }
        if(max_dif <= delta){
                real_conv = true;
        }

        max_dif = 0;
        for (k=-N; k<N+1; k++){
                k_new = k+N;

                max_dif_imag = fabs( imag(u_nm1[k_new])-imag(u_n[k_new]) );
                if(max_dif_imag >= max_dif){
                        max_dif = max_dif_imag;
                }
        }
        if(max_dif <= delta){
                imag_conv = true;
        }

        if(real_conv == true && imag_conv == true){
                steady_conv = true;
        }

        //cout << deltaMax1 <<" " <<deltaMax2 << endl ;
}

void update ( complex <double> u_nm1[2*N+1], complex <double> u_n[2*N+1], complex
    <double> R_nprev[2*N+1], complex <double> R_n[2*N+1])
{
        int k, k_new;
        for(k=-N; k<N+1; k++){
                k_new = k+N;

                u_n[k_new] = u_nm1[k_new];
                R_nprev[k_new] = R_n[k_new];
        }
}

void Ek_evaluation ( complex <double> u_nm1[2*N+1], double Ek[N])
{
        int k, k_new;
        for(k=1; k<N+1; k++){
                k_new = k+N;

                complex <double> A = u_nm1[k_new]*conj(u_nm1[k_new]);

                Ek[k-1] = sqrt(pow(real(A),2) + pow(imag(A),2));
        }
}
```