Study for the numerical resolution of conservation equations of mass, momentum and energy to be applied to solar thermal collectors

Bachelor's thesis

Annex D: Code of the program

Author

Daniel Yago Llamas

Director

Carlos David Pérez Segarra

Co-director

Francesc Xavier Trias Miquel

Bachelor's Degree in Aerospace Technology Engineering

Universitat Politècnica de Catalunya

Escola Tècnica Superior d'Enginyeries Industrial i Aeronàutica de Terrassa

June, 2015

Contents

1 Code of the program

1.1 Main program

```
#include <iostream>
#include <sstream>
#include <fstream>
#include <vector>
#include <cmath>
#include <cassert>
#include <cstdlib>
#include <string>

#define PI 3.14159265

using namespace std;

enum R_u_BccValues {
     Wall,
     Airflow
};

enum T_BccValues {
     Neumann,
     Dirichlet
};

enum R_u_BccValues str2R_u_BccValues(string str);
enum T_BccValues str2T_BccValues(string str);
double delta_inc(const vector<vector<double> > &A , const vector<vector<double> > &B );
double A_scheme (double P, string scheme="CDS");
double det_coord_Mesh ( double A , unsigned int N , double g , unsigned int i );
vector<vector<double> > conjugate_gradient ( const vector<vector<double> > &a_p , const
     vector<vector<double> > &a_e , const vector<vector<double> > &a_w , const vector<
     vector<double> > &a_n , const vector<vector<double> > &a_s
                     ,const vector<vector<double> > &phi_0 , const vector<vector<double>
                          > &b , int max_iter , double epsilon , int N_x , int N_y);
vector<vector<double> > biconjugate_gradient_stb ( const vector<vector<double> > &a_p ,
     const vector<vector<double> > &a_e , const vector<vector<double> > &a_w , const
     vector<vector<double> > &a_n , const vector<vector<double> > &a_s ,
                    const vector<vector<double> > &phi_0 , const vector<vector<double> >
                         &b , int max_iter , double epsilon , int N_x , int N_y);


class MatrixCoef
{
  public:
    vector <vector<double> > a_p , a_n , a_s , a_e , a_w , b_p;
    MatrixCoef(unsigned int N_i , unsigned int N_j ):
      a_p(N_i , vector<double>(N_j, 1.0) ), a_n(N_i , vector<double>(N_j, 0.0) ),
      a_s(N_i , vector<double>(N_j, 0.0) ), a_e(N_i , vector<double>(N_j, 0.0) ),
      a_w(N_i , vector<double>(N_j, 0.0) ), b_p(N_i , vector<double>(N_j, 0.0) ) {};
};

class velocity_x
{
  public:
    vector <vector<double> > u_n , u_s , u_e , u_w ;
    velocity_x (unsigned int N_i , unsigned int N_j) :
      u_n (N_i , vector<double>(N_j, 0.0) ) ,
```

```
      u_s (N_i , vector<double>(N_j, 0.0) ) ,
      u_e (N_i , vector<double>(N_j, 0.0) ) ,
      u_w (N_i , vector<double>(N_j, 0.0) ) {};
};

class velocity_y
{
  public:
    vector <vector<double> > v_n , v_s , v_e , v_w ;
    velocity_y (unsigned int N_i , unsigned int N_j) :
      v_n (N_i , vector<double>(N_j, 0.0) ) ,
      v_s (N_i , vector<double>(N_j, 0.0) ) ,
      v_e (N_i , vector<double>(N_j, 0.0) ) ,
      v_w (N_i , vector<double>(N_j, 0.0) ) {};
};

class Massflow
{
  public:
    vector <vector<double> > F_n , F_s , F_e , F_w;
    Massflow (unsigned int N_i , unsigned int N_j):
      F_n (N_i , vector<double>(N_j, 0.0) ) ,
      F_s (N_i , vector<double>(N_j, 0.0) ) ,
      F_e (N_i , vector<double>(N_j, 0.0) ) ,
      F_w (N_i , vector<double>(N_j, 0.0) ) {};
};

class gradiente
{
  public:
    vector <vector<double> > grad_n , grad_s , grad_e , grad_w;
    gradiente (unsigned int N_i , unsigned int N_j):
      grad_n (N_i , vector<double>(N_j, 0.0) ) ,
      grad_s (N_i , vector<double>(N_j, 0.0) ) ,
      grad_e (N_i , vector<double>(N_j, 0.0) ) ,
      grad_w (N_i , vector<double>(N_j, 0.0) ) {};
};

class Diffusion
{
  public:
    vector <vector<double> > D_n , D_s , D_e , D_w;
    Diffusion (unsigned int N_i , unsigned int N_j):
      D_n (N_i , vector<double>(N_j, 0.0) ) ,
      D_s (N_i , vector<double>(N_j, 0.0) ) ,
      D_e (N_i , vector<double>(N_j, 0.0) ) ,
      D_w (N_i , vector<double>(N_j, 0.0) ) {};
};

class Peclet
{
  public:
    vector <vector<double> > Pe_n , Pe_s , Pe_e , Pe_w;
    Peclet (unsigned int N_i , unsigned int N_j):
      Pe_n (N_i , vector<double>(N_j, 0.0) ) ,
      Pe_s (N_i , vector<double>(N_j, 0.0) ) ,
      Pe_e (N_i , vector<double>(N_j, 0.0) ) ,
      Pe_w (N_i , vector<double>(N_j, 0.0) ) {};
};

class caseConv2D
{
  private:
    double L_x , L_y , Ax , Ay , e , Pr , Ra , gx , gy , angle ;
    unsigned int N_i , N_j , max_iter ;
    vector <vector<double> > v_0 , v_1 , v_2 ;
    vector <vector<double> > u_0 , u_1 , u_2 ;
    vector <vector<double> > u_p2 , v_p2 ;
    vector <vector<double> > S_w , S_e , S_n , S_s , Vol ;
    vector <vector<double> > x , y , x_w , y_w , x_e , y_e , x_n , y_n , x_s , y_s ;
    vector <vector<double> > R_u0 , R_v0 , R_u1 , R_v1 ;
    vector <vector<double> > mu , rho ;
    velocity_x velx;
```

```
    velocity_y vely;
    Massflow mass_x;
    Massflow mass_y;
    gradiente gradient_u;
    gradiente gradient_v;
    vector <vector<double> > v_out , u_out ;
    Diffusion diff_T ;
    Massflow mass_T ;
    Peclet pe_T ;



  public:
    double At , t_2 , epsilon_vT , epsilon_pT ;
    double t_end ;
    MatrixCoef coef;
    vector <vector<double> > p_0 , p_1 , p_1sup ;
    vector <vector<double> > T_0 , T_1 , T_1sup;
    MatrixCoef coef_T ;
    caseConv2D(double Lx , double Ly , double A_x , double A_y , double E , double
        Epsilon_vT , double Epsilon_pT , double Pr_vol , double Ra_vol , double Mu ,
        double Rho , unsigned int Nx , unsigned int Ny , unsigned int Max_iter ,
        double t_End , double Gx , double Gy , double Angle);
    void SetUniformMesh();
    void SetNonUniformMesh();
    void Initialvalues();
    void Eval_boundary_velocity(enum R_u_BccValues bcc[4] , const vector<double> &value);
    void interfacevelocity_u();
    void interfacevelocity_v();
    void massflow_x();
    void massflow_y();
    void grad_u();
    void grad_v();
    void EvalR_u();
    void EvalR_v();
    void EvalVelocity_p();
    void EvalCoef();
    void EvalCoefbc();
    void Eval_coef_constantes_P();
    void Eval_coef_variables_P();
    void solveGS();
    void solveTDMA();
    void EvalVelocityfield();
    void Eval_boundary_temperature(enum T_BccValues bcc[4] ,const vector<double> &value);
    void Evaldiffusion_T();
    void Evalmassflow_T();
    void Evalpeclet_T();
    void EvalCoef_T();
    void Eval_boundary_CoefbcT_temperature(enum T_BccValues bcc[4] , const vector<double>
        &value);
    void solveTDMA_Temperature();
    void solveGS_Temp();
    double GetConvergence();
    void NewStep();
    void saveValues(double conv , unsigned int iter);
    void EvalTime();
    double CDSscheme_u1(int i , int j , int i1 , int j1);
    double CDSscheme_v1(int i , int j , int i1 , int j1);
    void Massconservation();
    double Nu_value();
};

int main()
{

  ifstream infile ("input.txt");
  if (infile.fail()){
    cout << "Error opening file " << "input.txt" << endl;
    exit(0);
  }

  string s;
  double Lx , Ly , A_x , A_y , E = 1.0 , Epsilon_vT , Epsilon_pT , Pr_vol , Ra_vol , Mu ,
```

```
    Rho ,t_End , convergence;
double Gx , Gy , Angle ;
unsigned int Nx , Ny , Max_iter , comp = 0 ;
vector<string> s_R_u_bcc(4) , s_T_bcc(4) ;
vector<double> v_R_u_bcc(8,0.0)  , v_T_bcc(4,0.0) ;
unsigned int count = 1;
unsigned int i , j , k , l;

infile >> s >> Lx;
infile >> s >> Ly;
infile >> s >> Nx;
infile >> s >> Ny;

infile >> s >> s ;
if (s.compare("NonUniform")==0)
{
  infile >> Gx >> Gy ;
}

infile >> s >> Max_iter;
infile >> s >> Epsilon_vT;
infile >> s >> Epsilon_pT;
infile >> s >> t_End;
infile >> s >> Pr_vol;
infile >> s >> Ra_vol;
infile >> s >> Angle;

Mu = 1.0 ;
Rho = Mu / sqrt(9.81 * Lx * Lx * Lx * Pr_vol / Ra_vol ) ;

infile >> s >> s_R_u_bcc[0] ;
    if(str2R_u_BccValues(s_R_u_bcc[0])==Airflow) infile >> v_R_u_bcc[0] >> v_R_u_bcc[1] ;
infile >> s >> s_R_u_bcc[1] ;
    if(str2R_u_BccValues(s_R_u_bcc[1])==Airflow) infile >> v_R_u_bcc[2] >> v_R_u_bcc[3] ;
infile >> s >> s_R_u_bcc[2] ;
    if(str2R_u_BccValues(s_R_u_bcc[2])==Airflow) infile >> v_R_u_bcc[4] >> v_R_u_bcc[5] ;
infile >> s >> s_R_u_bcc[3] ;
    if(str2R_u_BccValues(s_R_u_bcc[3])==Airflow) infile >> v_R_u_bcc[6] >> v_R_u_bcc[7] ;

infile >> s >> s_T_bcc[0] ;
    if(str2T_BccValues(s_T_bcc[0])==Dirichlet) infile >> v_T_bcc[0] ;
infile >> s >> s_T_bcc[1] ;
    if(str2T_BccValues(s_T_bcc[1])==Dirichlet) infile >> v_T_bcc[1] ;
infile >> s >> s_T_bcc[2] ;
    if(str2T_BccValues(s_T_bcc[2])==Dirichlet) infile >> v_T_bcc[2] ;
infile >> s >> s_T_bcc[3] ;
    if(str2T_BccValues(s_T_bcc[3])==Dirichlet) infile >> v_T_bcc[3] ;

A_y = Ly / Ny;
A_x = Lx / Nx;

enum R_u_BccValues R_u_bcc[] = {str2R_u_BccValues(s_R_u_bcc[0]), str2R_u_BccValues(
    s_R_u_bcc[1]), str2R_u_BccValues(s_R_u_bcc[2]) ,str2R_u_BccValues(s_R_u_bcc[3]) };

enum T_BccValues T_bcc[] = {str2T_BccValues(s_T_bcc[0]), str2T_BccValues(s_T_bcc[1]),
    str2T_BccValues(s_T_bcc[2]) ,str2T_BccValues(s_T_bcc[3]) };

cout.setf(ios::floatfield,ios::scientific);
cout.precision(3);

  caseConv2D cas( Lx , Ly , A_x , A_y , E , Epsilon_vT , Epsilon_pT , Pr_vol , Ra_vol ,
      Mu , Rho , Nx , Ny , Max_iter , t_End , Gx , Gy , Angle);
  cas.SetNonUniformMesh();
  cas.Initialvalues();

//first time step
//find predictor velocity field
cas.Eval_boundary_velocity(R_u_bcc , v_R_u_bcc);
cas.interfacevelocity_u();
cas.interfacevelocity_v();
cas.massflow_x();
cas.massflow_y();
cas.grad_u();
```

```
cas.grad_v();
cas.EvalR_u();
cas.EvalR_v();
cas.EvalVelocity_p();

//find pressure field
cas.Eval_coef_constantes_P();
cas.Eval_coef_variables_P();
cas.p_1 = conjugate_gradient ( cas.coef.a_p , cas.coef.a_e , cas.coef.a_w , cas.coef.a_n , cas.coef.a_s , cas.p_0 , cas.coef.b_p , Max_iter , Epsilon_pT, Nx , Ny);
cas.p_1sup = cas.p_1 ;

//find velocity field and check mass conservation
cas.EvalVelocityfield();
cas.Massconservation();

//Find temperature field
cas.Evaldiffusion_T();
cas.Evalmassflow_T();
cas.Evalpeclet_T();
cas.EvalCoef_T();
cas.Eval_boundary_CoefbcT_temperature( T_bcc , v_T_bcc );
cas.T_1 = biconjugate_gradient_stb ( cas.coef_T.a_p , cas.coef_T.a_e , cas.coef_T.a_w , cas.coef_T.a_n , cas.coef_T.a_s , cas.T_1sup , cas.coef_T.b_p , Max_iter , Epsilon_pT, Nx , Ny);
cas.T_1sup = cas.T_1 ;

//new time step parameters
cas.EvalTime();
cas.NewStep();
cout << cas.t_2 <<endl;

  ofstream outfile("conv.m");
if(outfile.fail()){
  cerr << "Error creating conv.m file. " << endl;
  exit(-1);
}

outfile.setf(ios::floatfield,ios::scientific);
outfile.precision(10);
outfile << "convs=[" ;
outfile << cas.t_2 << "  " << cas.GetConvergence() << " " << cas.Nu_value() << endl;

  while (cas.t_2 < cas.t_end && comp==0)
{
  //Find predictor-velocity field
  cas.Eval_boundary_velocity(R_u_bcc , v_R_u_bcc);
  cas.interfacevelocity_u();
  cas.interfacevelocity_v();
  cas.massflow_x();
  cas.massflow_y();
  cas.grad_u();
  cas.grad_v();
  cas.EvalR_u();
  cas.EvalR_v();
  cas.EvalVelocity_p();

  //Find pressure field
  cas.Eval_coef_variables_P();
    cas.p_1 = conjugate_gradient ( cas.coef.a_p , cas.coef.a_e , cas.coef.a_w , cas.coef.a_n , cas.coef.a_s , cas.p_0 , cas.coef.b_p , Max_iter , Epsilon_pT, Nx , Ny);
    cas.p_1sup = cas.p_1 ;

    //Find velocity field
    cas.EvalVelocityfield();
    cas.Massconservation();

    //Find temperature field
    cas.Evalmassflow_T();
    cas.Evalpeclet_T();
    cas.EvalCoef_T();
    cas.Eval_boundary_CoefbcT_temperature( T_bcc , v_T_bcc );
    cas.T_1 = biconjugate_gradient_stb ( cas.coef_T.a_p , cas.coef_T.a_e , cas.coef_T.a_w
```

```
        , cas.coef_T.a_n , cas.coef_T.a_s , cas.T_1sup , cas.coef_T.b_p , Max_iter , Epsilon_pT, Nx , Ny);
    cas.T_1sup = cas.T_1 ;

    convergence = cas.GetConvergence();
    outfile << cas.t_2 << "␣␣" << convergence << "␣" << cas.Nu_value() << endl;
    if( convergence < cas.epsilon_vT && cas.t_2 > 1.0 )
    {
      comp=1;
    }
    else
    {
      if ( count % 1000 == 0 )
      {
        cas.saveValues(convergence , count);
      }
      count++;

        cas.EvalTime();
        if (cas.t_2 < cas.t_end)
        {
          cas.NewStep();
      }
      cout << "time=" << cas.t_2 << "␣iter=" << count << "␣error=" << convergence << "␣Nu=" << cas.Nu_value() << endl;
      }
    }

    cas.saveValues(convergence , count);
    outfile << "];" ;
    outfile.close();

  return 0;
}

caseConv2D::caseConv2D(double Lx , double Ly , double A_x , double A_y , double E , double Epsilon_vT , double Epsilon_pT , double Pr_vol ,
        double Ra_vol , double Mu , double Rho , unsigned int Nx , unsigned int Ny , unsigned int Max_iter , double t_End , double Gx , double Gy , double Angle):
      L_x(Lx) , L_y(Ly) , Ax(A_x) , Ay(A_y) , e(E) , epsilon_vT(Epsilon_vT) , epsilon_pT(Epsilon_pT) , Pr(Pr_vol) , Ra(Ra_vol) , At(1E-15) , t_2(1E-15) , N_i(Ny) , N_j(Nx) , max_iter(Max_iter) , t_end(t_End) , gx(Gx) , gy(Gy) , angle(Angle) , v_0(N_i+1, vector<double>(N_j+2,0.0)) , v_1(N_i+1, vector<double>(N_j+2,0.0)) , v_2(N_i+1, vector<double>(N_j+2,0.0)) , u_0(N_i+2, vector<double>(N_j+1,0.0)) , u_1(N_i+2, vector<double>(N_j+1,0.0)) , u_2(N_i+2, vector<double>(N_j+1,0.0)) , u_p2(N_i+2, vector<double>(N_j+1,0.0)) , v_p2(N_i+1, vector<double>(N_j+2,0.0)) , S_w(N_i+1, vector<double>(N_j+1,0.0)) , S_e(N_i+1, vector<double>(N_j+1,0.0)) , S_n(N_i+1, vector<double>(N_j+1,0.0)) , S_s(N_i+1, vector<double>(N_j+1,0.0)) , Vol(N_i+1, vector<double>(N_j+1,0.0)) , x(N_i+2, vector<double>(N_j+2,0.0)) , y(N_i+2, vector<double>(N_j+2,0.0)) , x_w(N_i+2, vector<double>(N_j+2,0.0)) , y_w(N_i+2, vector<double>(N_j+2,0.0)) , x_e(N_i+2, vector<double>(N_j+2,0.0)) , y_e(N_i+2, vector<double>(N_j+2,0.0)) ,  x_n(N_i+2, vector<double>(N_j+2,0.0)) , y_n(N_i+2, vector<double>(N_j+2,0.0)) , x_s(N_i+2, vector<double>(N_j+2,0.0)) , y_s(N_i+2, vector<double>(N_j+2,0.0)) , R_u0(N_i+2, vector<double>(N_j+1,0.0)) , R_v0(N_i+1, vector<double>(N_j+2,0.0)) , R_u1(N_i+2, vector<double>(N_j+1,0.0)) , R_v1(N_i+1, vector<double>(N_j+2,0.0)) , p_0(N_i+2, vector<double>(N_j+2,1.0)) ,  p_1(N_i+2, vector<double>(N_j+2,1.0)) , p_1sup(N_i+2, vector<double>(N_j+2,1.0)) , mu(N_i+1, vector<double>(N_j+1,Mu)) , rho(N_i+1, vector<double>(N_j+1,Rho)) , T_0(N_i+2, vector<double>(N_j+2,0.5)) , T_1(N_i+2, vector<double>(N_j+2,0.5)) , T_1sup(N_i+2, vector<double>(N_j+2,0.5)) , coef(N_i+1, N_j+1) , velx(N_i+1, N_j) , vely(N_i, N_j+1) , mass_x(N_i+1, N_j+1) , mass_y(N_i+1, N_j+1) , gradient_u(N_i+1, N_j) , gradient_v(N_i, N_j+1) , v_out(N_i+1, vector<double>(N_j+1,0.0)) , u_out(N_i+1, vector<double>(N_j+1,0.0)) , diff_T(N_i+1, N_j+1) , mass_T(N_i+1, N_j+1) , pe_T(N_i+1, N_j+1) , coef_T(N_i+1, N_j+1) {};

void caseConv2D::SetUniformMesh()
{
  unsigned int i , j;

  for (i=1;i<=N_i;i++)
  {
```

```
    for(j=1;j<=N_j;j++)
    {
      x[i][j] = (j-0.5) * Ax ;
      y[i][j] = (i-0.5) * Ay ;

      x_w[i][j] = x[i][j] - 0.5 * Ax ;
      x_e[i][j] = x[i][j] + 0.5 * Ax ;
      x_n[i][j] = x[i][j] ;
      x_s[i][j] = x[i][j] ;
      y_w[i][j] = y[i][j] ;
      y_e[i][j] = y[i][j] ;
      y_n[i][j] = y[i][j] + 0.5 * Ay ;
      y_s[i][j] = y[i][j] - 0.5 * Ay ;

      S_w[i][j] = (y_n[i][j] - y_s[i][j]) * e;
      S_e[i][j] = (y_n[i][j] - y_s[i][j]) * e;
      S_n[i][j] = (x_e[i][j] - x_w[i][j]) * e;
      S_s[i][j] = (x_e[i][j] - x_w[i][j]) * e;
      Vol[i][j] = (y_n[i][j] - y_s[i][j]) * (x_e[i][j] - x_w[i][j]) * e;
    }
  }

  for (j=1;j<=N_j;j++)
  {
    x[0][j] = (j-0.5) * Ax ;
    x[N_i+1][j] = (j-0.5) * Ax ;
    y[0][j] = 0.0 ;
    y[N_i+1][j] = L_y ;

    x_w[0][j] = x[0][j] - 0.5 * Ax ;
    y_w[0][j] = y[0][j] ;
    x_e[0][j] = x[0][j] + 0.5 * Ax ;
    y_e[0][j] = y[0][j] ;
    x_n[0][j] = x[0][j] ;
    y_n[0][j] = y[0][j] ;
    x_s[0][j] = x[0][j] ;
    y_s[0][j] = y[0][j] ;

    x_w[N_i+1][j] = x[N_i+1][j] - 0.5 * Ax ;
    y_w[N_i+1][j] = y[N_i+1][j] ;
    x_e[N_i+1][j] = x[N_i+1][j] + 0.5 * Ax ;
    y_e[N_i+1][j] = y[N_i+1][j] ;
    x_n[N_i+1][j] = x[N_i+1][j];
    y_n[N_i+1][j] = y[N_i+1][j] ;
    x_s[N_i+1][j] = x[N_i+1][j];
    y_s[N_i+1][j] = y[N_i+1][j] ;
  }

  for (i=1;i<=N_i;i++)
  {
    x[i][0] = 0.0 ;
    x[i][N_j+1] = L_x ;
    y[i][0] = (i-0.5) * Ay ;
    y[i][N_j+1] = (i-0.5) * Ay ;

    x_n[i][0] = x[i][0] ;
    y_n[i][0] = y[i][0] + 0.5 * Ay ;
    x_s[i][0] = x[i][0] ;
    y_s[i][0] = y[i][0] - 0.5 * Ay ;
    x_e[i][0] = x[i][0] ;
    y_e[i][0] = y[i][0] ;
    x_w[i][0] = x[i][0] ;
    y_w[i][0] = y[i][0] ;


    x_n[i][N_j+1] = x[i][N_j+1] ;
    y_n[i][N_j+1] = y[i][N_j+1] + 0.5 * Ay ;
    x_s[i][N_j+1] = x[i][N_j+1] ;
    y_s[i][N_j+1] = y[i][N_j+1] - 0.5 * Ay ;
    x_e[i][N_j+1] = x[i][N_j+1] ;
    y_e[i][N_j+1] = y[i][N_j+1] ;
    x_w[i][N_j+1] = x[i][N_j+1] ;
    y_w[i][N_j+1] = y[i][N_j+1] ;
```

```
  }
    y[N_i+1][0] = L_y ;
    y_n[N_i+1][0] = L_y ;
    y_s[N_i+1][0] = L_y ;
    y_e[N_i+1][0] = L_y ;
    y_w[N_i+1][0] = L_y ;

    x[0][N_j+1] = L_x ;
    x_n[0][N_j+1] = L_x;
    x_s[0][N_j+1] = L_x;
    x_e[0][N_j+1] = L_x;
    x_w[0][N_j+1] = L_x;

    x[N_i+1][N_j+1] = L_x ;
    y[N_i+1][N_j+1] = L_y ;
    x_n[N_i+1][N_j+1] = L_x ;
    y_n[N_i+1][N_j+1] = L_y ;
    x_s[N_i+1][N_j+1] = L_x ;
    y_s[N_i+1][N_j+1] = L_y ;
    x_e[N_i+1][N_j+1] = L_x ;
    y_e[N_i+1][N_j+1] = L_y ;
    x_w[N_i+1][N_j+1] = L_x ;
    y_w[N_i+1][N_j+1] = L_y ;
}

void caseConv2D::SetNonUniformMesh()
{

  unsigned int i , j;
  unsigned int N_x = N_j , N_y = N_i ;

  for (i=1;i<=N_i;i++)
  {
    for(j=1;j<=N_j;j++)
    {

      x[i][j] = (det_coord_Mesh ( L_x , N_x , gx , j ) + det_coord_Mesh ( L_x , N_x , gx ,
           j+1 ))/2.0 ;
      y[i][j] = (det_coord_Mesh ( L_y , N_y , gy , i ) + det_coord_Mesh ( L_y , N_y , gy ,
           i+1 ))/2.0  ;

      x_w[i][j] = det_coord_Mesh ( L_x , N_x , gx , j ) ;
      x_e[i][j] = det_coord_Mesh ( L_x , N_x , gx , j+1 ) ;
      x_n[i][j] = x[i][j] ;
      x_s[i][j] = x[i][j] ;
      y_w[i][j] = y[i][j] ;
      y_e[i][j] = y[i][j] ;
      y_n[i][j] = det_coord_Mesh ( L_y , N_y , gy , i+1 ) ;
      y_s[i][j] = det_coord_Mesh ( L_y , N_y , gy , i ) ;

      S_w[i][j] = (y_n[i][j] - y_s[i][j]) * e;
      S_e[i][j] = (y_n[i][j] - y_s[i][j]) * e;
      S_n[i][j] = (x_e[i][j] - x_w[i][j]) * e;
      S_s[i][j] = (x_e[i][j] - x_w[i][j]) * e;
      Vol[i][j] = (y_n[i][j] - y_s[i][j]) * (x_e[i][j] - x_w[i][j]) * e;

    }
  }

  for (j=1;j<=N_j;j++)
  {
    x[0][j] = (det_coord_Mesh ( L_x , N_x , gx , j ) + det_coord_Mesh ( L_x , N_x , gx , j
        +1 ))/2.0 ;
    x[N_i+1][j] = (det_coord_Mesh ( L_x , N_x , gx , j ) + det_coord_Mesh ( L_x , N_x , gx
        , j+1 ))/2.0 ;
    y[0][j] = 0.0 ;
    y[N_i+1][j] = L_y ;

    x_w[0][j] = det_coord_Mesh ( L_x , N_x , gx , j ) ;
    y_w[0][j] = y[0][j] ;
    x_e[0][j] = det_coord_Mesh ( L_x , N_x , gx , j+1 ) ;
    y_e[0][j] = y[0][j] ;
    x_n[0][j] = x[0][j] ;
```

```
    y_n[0][j] = y[0][j] ;
    x_s[0][j] = x[0][j] ;
    y_s[0][j] = y[0][j] ;

    x_w[N_i+1][j] = det_coord_Mesh ( L_x , N_x , gx , j ) ;
    y_w[N_i+1][j] = y[N_i+1][j] ;
    x_e[N_i+1][j] = det_coord_Mesh ( L_x , N_x , gx , j+1 ) ;
    y_e[N_i+1][j] = y[N_i+1][j] ;
    x_n[N_i+1][j] = x[N_i+1][j];
    y_n[N_i+1][j] = y[N_i+1][j] ;
    x_s[N_i+1][j] = x[N_i+1][j];
    y_s[N_i+1][j] = y[N_i+1][j] ;
  }

  for (i=1;i<=N_i;i++)
  {
    x[i][0] = 0.0 ;
    x[i][N_j+1] = L_x ;
    y[i][0] = (det_coord_Mesh ( L_y , N_y , gy , i ) + det_coord_Mesh ( L_y , N_y , gy , i
         +1 ))/2.0  ;
    y[i][N_j+1] = (det_coord_Mesh ( L_y , N_y , gy , i ) + det_coord_Mesh ( L_y , N_y , gy
         , i+1 ))/2.0  ;

    x_n[i][0] = x[i][0] ;
    y_n[i][0] = det_coord_Mesh ( L_y , N_y , gy , i+1 ) ;
    x_s[i][0] = x[i][0] ;
    y_s[i][0] = det_coord_Mesh ( L_y , N_y , gy , i ) ;
    x_e[i][0] = x[i][0] ;
    y_e[i][0] = y[i][0] ;
    x_w[i][0] = x[i][0] ;
    y_w[i][0] = y[i][0] ;


    x_n[i][N_j+1] = x[i][N_j+1] ;
    y_n[i][N_j+1] = det_coord_Mesh ( L_y , N_y , gy , i+1 ) ;
    x_s[i][N_j+1] = x[i][N_j+1] ;
    y_s[i][N_j+1] = det_coord_Mesh ( L_y , N_y , gy , i ) ;
    x_e[i][N_j+1] = x[i][N_j+1] ;
    y_e[i][N_j+1] = y[i][N_j+1] ;
    x_w[i][N_j+1] = x[i][N_j+1] ;
    y_w[i][N_j+1] = y[i][N_j+1] ;
  }
    y[N_i+1][0] = L_y ;
    y_n[N_i+1][0] = L_y ;
    y_s[N_i+1][0] = L_y ;
    y_e[N_i+1][0] = L_y ;
    y_w[N_i+1][0] = L_y ;

    x[0][N_j+1] = L_x ;
    x_n[0][N_j+1] = L_x;
    x_s[0][N_j+1] = L_x;
    x_e[0][N_j+1] = L_x;
    x_w[0][N_j+1] = L_x;

    x[N_i+1][N_j+1] = L_x ;
    y[N_i+1][N_j+1] = L_y ;
    x_n[N_i+1][N_j+1] = L_x ;
    y_n[N_i+1][N_j+1] = L_y ;
    x_s[N_i+1][N_j+1] = L_x ;
    y_s[N_i+1][N_j+1] = L_y ;
    x_e[N_i+1][N_j+1] = L_x ;
    y_e[N_i+1][N_j+1] = L_y ;
    x_w[N_i+1][N_j+1] = L_x ;
    y_w[N_i+1][N_j+1] = L_y ;

  ofstream outfile("mesh.m");

    if(outfile.fail())
    {
      cerr << "Error creating mesh.m file. " << endl;
      exit(-1);
    }

```

```
      outfile << "x=[" << endl;
      for (i=1;i<N_i+1;i++)
      {
        for (j=1;j<N_j+1;j++)
        {
          outfile << x[i][j] << "␣" ;
        }
        outfile << endl ;
      }
      outfile << "];" << endl << endl;

      outfile << "y=[" << endl;
      for (i=1;i<N_i+1;i++)
      {
        for (j=1;j<N_j+1;j++)
        {
          outfile << y[i][j] << "␣" ;
        }
        outfile << endl ;
      }
      outfile << "];" << endl << endl;

    outfile.close();

}

void caseConv2D::Initialvalues()
{
  u_1 = u_0;
  v_1 = v_0;
  p_1sup = p_0;
  T_1sup = T_0 ;

  unsigned int i , j ;
  double temp = 0.0 ;
  for (i=1;i<N_i;i++)
  {
    for (j=1;j<N_j+1;j++)
    {
      temp = (y_n[i][j]-y[i][j]) / (y[i+1][j]-y[i][j]) ;
      R_v0[i][j] = Pr * ( T_0[i][j] * (1.0-temp) + temp * T_0[i+1][j] ) ;
    }
  }
}

enum R_u_BccValues str2R_u_BccValues(string str)
{
    if(str=="Wall")
    {
      return (Wall);
    }
    else if (str=="Airflow")
    {
      return (Airflow);
    }
    else
    {
      cerr << "Error␣in␣" << __FUNCTION__ << ":␣" << str << "␣condition␣doesn't␣exist." << endl;
      exit(0);
    }
}

enum T_BccValues str2T_BccValues(string str)
{
    if(str=="Neumann")
    {
      return (Neumann);
    }
    else if (str=="Dirichlet")
    {
      return (Dirichlet);
    }
```

```
    else
    {
      cerr << "Error␣in␣" << __FUNCTION__ << ":␣" << str << "␣condition␣doesn't␣exist." <<
          endl;
      exit(0);
    }
}

void caseConv2D::Eval_boundary_velocity(enum R_u_BccValues bcc[4] , const vector<double>
    &value)
{
  unsigned int i , j ;

  switch (bcc[0])
  {
    case Wall:
      j=0;
      for(i=0;i<N_i+2;i++)
      {
        u_1[i][j] = 0.0;
        u_p2[i][j] = 0.0;
      }
      for(i=0;i<N_i+1;i++)
      {
        v_1[i][j] = 0.0;
        v_p2[i][j] = 0.0;
      }
      break;
    case Airflow:
      j=0;
      for(i=0;i<N_i+2;i++)
      {
        u_1[i][j] = value[0];
        u_p2[i][j] = value[0];
      }
      for(i=0;i<N_i+1;i++)
      {
        v_1[i][j] = value[1];
        v_p2[i][j] = value[1];
      }
      break;
    default:
      assert(false);
      break;
  }
  switch (bcc[1]){
    case Wall:
      i=0;
      for(j=0;j<N_j+1;j++)
      {
        u_1[i][j] = 0.0;
        u_p2[i][j] = 0.0;
      }
      for(j=0;j<N_j+2;j++)
      {
        v_1[i][j] = 0.0;
        v_p2[i][j] = 0.0;
      }
      break;
    case Airflow:
      i=0;
      for(j=0;j<N_j+1;j++)
      {
        u_1[i][j] = value[2];
        u_p2[i][j] = value[2];
      }
      for(j=0;j<N_j+2;j++)
      {
        v_1[i][j] = value[3];
        v_p2[i][j] = value[3];
      }
      break;
    default:
```

```
        assert(false);
        break;
  }
  switch (bcc[2]){
    case Wall:
      j=N_j;
      for(i=0;i<N_i+2;i++)
      {
        u_1[i][j] = 0.0;
        u_p2[i][j] = 0.0;
      }
      j=N_j+1;
      for(i=0;i<N_i+1;i++)
      {
        v_1[i][j] = 0.0;
        v_p2[i][j] = 0.0;
      }
      break;
    case Airflow:
      j=N_j;
      for(i=0;i<N_i+2;i++)
      {
        u_1[i][j] = value[4];
        u_p2[i][j] = value[4];
      }
      j=N_j+1;
      for(i=0;i<N_i+1;i++)
      {
        v_1[i][j] = value[5];
        v_p2[i][j] = value[5];
      }
      break;
    default:
      assert(false);
      break;
  }
  switch (bcc[3]){
    case Wall:
      i=N_i+1;
      for(j=0;j<N_j+1;j++)
      {
        u_1[i][j] = 0.0;
        u_p2[i][j] = 0.0;
      }
      i=N_i;
      for(j=0;j<N_j+2;j++)
      {
        v_1[i][j] = 0.0;
        v_p2[i][j] = 0.0;
      }
      break;
    case Airflow:
      i=N_i+1;
      for(j=0;j<N_j+1;j++)
      {
        u_1[i][j] = value[6];
        u_p2[i][j] = value[6];
      }
      i=N_i;
      for(j=0;j<N_j+2;j++)
      {
        v_1[i][j] = value[7];
        v_p2[i][j] = value[7];
      }
      break;
    default:
      assert(false);
      break;
  }
}

double caseConv2D::CDSscheme_u1(int i , int j , int i1 , int j1)
{
```

```
  double temp=0.0;

  if (i==0 && i!=i1)
  {
    temp = 0.0 ;
  }
  else if(i1==N_i+1 && i!=i1)
  {
    temp = 1.0 ;
  }
  else if(i==i1 && j!=j1)
  {
    temp = (x[i1][j1]-x_e[i][j]) / (x_e[i1][j1]-x_e[i][j]) ;
  }
  else
  {
    temp = (y_n[i][j]-y[i][j]) / (y[i1][j1]-y[i][j]) ;
  }

  return u_1[i][j] * (1.0-temp) + temp * u_1[i1][j1];
}

double caseConv2D::CDSscheme_v1(int i , int j , int i1 , int j1)
{
  double temp=0.0;

  if (j==0 && j!=j1)
  {
    temp = 0.0 ;
  }
  else if(j1==N_j+1 && j!=j1)
  {
    temp = 1.0 ;
  }
  else if(j==j1 && i!=i1)
  {
    temp = (y[i1][j1]-y_n[i][j]) / (y_n[i1][j1]-y_n[i][j]) ;
  }
  else
  {
    temp = (x_e[i][j]-x[i][j]) / (x[i1][j1]-x[i][j]) ;
  }

  return v_1[i][j] * (1.0-temp) + temp * v_1[i1][j1];
}

void caseConv2D::interfacevelocity_u()
{
  unsigned int i , j ;

  for (i=1;i<N_i+1;i++)
  {
    for (j=1;j<N_j;j++)
    {
      velx.u_e[i][j] = CDSscheme_u1(i,j,i,j+1) ;
      velx.u_w[i][j] = CDSscheme_u1(i,j-1,i,j) ;
      velx.u_n[i][j] = CDSscheme_u1(i,j,i+1,j) ;
      velx.u_s[i][j] = CDSscheme_u1(i-1,j,i,j) ;
    }
  }
}

void caseConv2D::interfacevelocity_v()
{
  unsigned int i , j ;

  for (i=1;i<N_i;i++)
  {
    for (j=1;j<N_j+1;j++)
    {
      vely.v_e[i][j] = CDSscheme_v1(i,j,i,j+1) ;
      vely.v_w[i][j] = CDSscheme_v1(i,j-1,i,j) ;
      vely.v_n[i][j] = CDSscheme_v1(i,j,i+1,j) ;
```

```
        vely.v_s[i][j] = CDSscheme_v1(i-1,j,i,j) ;
      }
    }
}

void caseConv2D::massflow_x()
{
  unsigned int i , j;

  for (i=1;i<N_i+1;i++)
  {
    for (j=1;j<N_j;j++)
    {
      mass_x.F_e[i][j] = ( u_1[i][j] + u_1[i][j+1] ) / 2.0 * (y_n[i][j] - y_s[i][j]) ;
      mass_x.F_w[i][j] = ( u_1[i][j] + u_1[i][j-1] ) / 2.0 * (y_n[i][j] - y_s[i][j])  ;
      mass_x.F_n[i][j] = ( v_1[i][j] * ( x_e[i][j] - x[i][j] ) + v_1[i][j+1] * ( x[i][j+1] - x_e[i][j] ) ) ;
      mass_x.F_s[i][j] = ( v_1[i-1][j] * ( x_e[i][j] - x[i][j] ) + v_1[i-1][j+1] * ( x[i][j+1] - x_e[i][j] ) ) ;
    }
  }
}

void caseConv2D::massflow_y()
{
  unsigned int i , j;

  for (i=1;i<N_i;i++)
  {
    for (j=1;j<N_j+1;j++)
    {
      mass_y.F_e[i][j] = ( u_1[i][j] * ( y_n[i][j] - y[i][j] ) + u_1[i+1][j] * ( y[i+1][j] - y_n[i][j] ) ) ;
      mass_y.F_w[i][j] = ( u_1[i][j-1] * ( y_n[i][j] - y[i][j] ) + u_1[i+1][j-1] * ( y[i+1][j] - y_n[i][j] ) ) ;
      mass_y.F_n[i][j] = ( v_1[i+1][j] + v_1[i][j] ) / 2.0 * ( x_e[i][j] - x_w[i][j] ) ;
      mass_y.F_s[i][j] = ( v_1[i-1][j] + v_1[i][j] ) / 2.0 * ( x_e[i][j] - x_w[i][j] ) ;
    }
  }
}

void caseConv2D::grad_u()
{
  unsigned int i , j ;

  for (i=1;i<N_i+1;i++)
  {
    for (j=1;j<N_j;j++)
    {
      gradient_u.grad_e[i][j] = (u_1[i][j+1] - u_1[i][j]) / (x_e[i][j+1] - x_e[i][j]) * (y_n[i][j] - y_s[i][j]) ;
      gradient_u.grad_w[i][j] = (u_1[i][j] - u_1[i][j-1]) / (x_e[i][j] - x_e[i][j-1]) * (y_n[i][j] - y_s[i][j]) ;
      gradient_u.grad_n[i][j] = (u_1[i+1][j] - u_1[i][j]) / (y[i+1][j] - y[i][j]) * (x[i][j+1] - x[i][j]);
      gradient_u.grad_s[i][j] = (u_1[i][j] - u_1[i-1][j]) / (y[i][j] - y[i-1][j]) * (x[i][j+1] - x[i][j]);
    }
  }
}

void caseConv2D::grad_v()
{
  unsigned int i , j ;

  for (i=1;i<N_i;i++)
  {
    for (j=1;j<N_j+1;j++)
    {
      gradient_v.grad_e[i][j] = (v_1[i][j+1] - v_1[i][j]) / (x[i][j+1] - x[i][j]) * (y[i+1][j] - y[i][j]);
      gradient_v.grad_w[i][j] = (v_1[i][j] - v_1[i][j-1]) / (x[i][j] - x[i][j-1]) * (y[i+1][j] - y[i][j]);
```

```
      gradient_v.grad_n[i][j] = (v_1[i+1][j] - v_1[i][j]) / (y_n[i+1][j] - y_n[i][j]) * (x_e[i][j] - x_w[i][j]) ;
      gradient_v.grad_s[i][j] = (v_1[i][j] - v_1[i-1][j]) / (y_n[i][j] - y_n[i-1][j]) * (x_e[i][j] - x_w[i][j]) ;
    }
  }
}

void caseConv2D::EvalR_u()
{
  unsigned int i , j ;
  double temp = 0.0 , temp1 = 0.0 , temp2 = 0.0 , temp3 = 0.0 ;

  for (i=1;i<N_i+1;i++)
  {
    for (j=1;j<N_j;j++)
    {
      temp1 = gradient_u.grad_e[i][j] + gradient_u.grad_n[i][j] - gradient_u.grad_w[i][j] - gradient_u.grad_s[i][j] ;
      temp2 = mass_x.F_e[i][j] * velx.u_e[i][j] + mass_x.F_n[i][j] * velx.u_n[i][j] - mass_x.F_w[i][j] * velx.u_w[i][j] - mass_x.F_s[i][j] * velx.u_s[i][j] ;

      temp = (x_e[i][j]-x[i][j]) / (x[i][j+1]-x[i][j]) ;
      temp3 = Pr * ( T_0[i][j] * (1.0-temp) + temp * T_0[i][j+1] ) * cos (angle*PI/180) ;

      R_u1[i][j] = ( Pr*temp1/sqrt(Ra) - temp2 ) / ( (x[i][j+1]-x[i][j])*(y_n[i][j]-y_s[i][j]) ) + temp3 ;
    }
  }
}

void caseConv2D::EvalR_v()
{
  unsigned int i , j ;
  double temp = 0.0 , temp1 = 0.0 , temp2 = 0.0 , temp3 = 0.0 ;

  for (i=1;i<N_i;i++)
  {
    for (j=1;j<N_j+1;j++)
    {
      temp1 = gradient_v.grad_e[i][j] + gradient_v.grad_n[i][j] - gradient_v.grad_w[i][j] - gradient_v.grad_s[i][j] ;
      temp2 = mass_y.F_e[i][j] * vely.v_e[i][j] + mass_y.F_n[i][j] * vely.v_n[i][j] - mass_y.F_w[i][j] * vely.v_w[i][j] - mass_y.F_s[i][j] * vely.v_s[i][j] ;

      temp = (y_n[i][j]-y[i][j]) / (y[i+1][j]-y[i][j]) ;
      temp3 = Pr * ( T_0[i][j] * (1.0-temp) + temp * T_0[i+1][j] ) * sin (angle*PI/180) ;

      R_v1[i][j] = ( Pr*temp1/sqrt(Ra) - temp2 ) / ( (x_e[i][j] - x_w[i][j])*(y[i+1][j]-y[i][j]) ) + temp3 ;
    }
  }
}

void caseConv2D::EvalVelocity_p()
{
  unsigned int i , j;

  for (i=1;i<N_i+1;i++)
  {
    for(j=1;j<N_j;j++)
    {
      u_p2[i][j] = u_1[i][j] + At * ( 3.0/2.0 * R_u1[i][j] - 1.0/2.0 * R_u0[i][j] ) ;
    }
  }
  for (i=1;i<N_i;i++)
  {
    for(j=1;j<N_j+1;j++)
    {
      v_p2[i][j] = v_1[i][j] + At * ( 3.0/2.0 * R_v1[i][j] - 1.0/2.0 * R_v0[i][j] ) ;
    }
  }
}
```

```
void caseConv2D::EvalCoef()
{
  unsigned int i , j;

  for(i=2;i<N_i;i++)
  {
    for(j=2;j<N_j;j++)
    {
      coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
      coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
      coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
      coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
      coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
      coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j]    ) ;
    }
  }
}

void caseConv2D::EvalCoefbc()
{
  unsigned int i , j;

  {
    i=1; j=1;
      coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
      coef.a_s[i][j] = 0.0 ;
      coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
      coef.a_w[i][j] = 0.0 ;
      coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
      coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j]    ) ;
  }

  {
    i=1; j=N_j;
      coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
      coef.a_s[i][j] = 0.0 ;
      coef.a_e[i][j] = 0.0 ;
      coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
      coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
      coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j]    ) ;
  }

  {
    i=N_i; j=1;
      coef.a_n[i][j] = 0.0 ;
      coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
      coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
      coef.a_w[i][j] = 0.0 ;
      coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
      coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j]    ) ;
  }

  {
    i=N_i; j=N_j;
      coef.a_n[i][j] = 0.0 ;
      coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
      coef.a_e[i][j] = 0.0 ;
      coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
      coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
      coef.b_p[i][j] = -( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                 +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j]   ) ;
  }

  i=1;
  for (j=2;j<N_j;j++)
  {
      coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
      coef.a_s[i][j] = 0.0 ;
```

```
        coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
        coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
        coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
        coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                 +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j]   ) ;
    }

    i=N_i;
    for (j=2;j<N_j;j++)
    {
        coef.a_n[i][j] = 0.0 ;
        coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
        coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
        coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
        coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
        coef.b_p[i][j] = -( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                  +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
    }

    j=1;
    for (i=2;i<N_i;i++)
    {
        coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
        coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
        coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
        coef.a_w[i][j] = 0.0 ;
        coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
        coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                 +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j]   ) ;
    }

    j=N_j;
    for (i=2;i<N_i;i++)
    {
        coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
        coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
        coef.a_e[i][j] = 0.0 ;
        coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
        coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
        coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                 +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j]   ) ;
    }
}

void caseConv2D::Eval_coef_constantes_P()
{
    unsigned int i , j;

    for(i=2;i<N_i;i++)
    {
      for(j=2;j<N_j;j++)
      {
        coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
        coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
        coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
        coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
        coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
      }
    }

    {
      i=1; j=1;
        coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
        coef.a_s[i][j] = 0.0 ;
        coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
        coef.a_w[i][j] = 0.0 ;
        coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
    }

    {
      i=1; j=N_j;
        coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
        coef.a_s[i][j] = 0.0 ;
```

```
      coef.a_e[i][j] = 0.0 ;
      coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
      coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
  }

  {
    i=N_i; j=1;
      coef.a_n[i][j] = 0.0 ;
      coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
      coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
      coef.a_w[i][j] = 0.0 ;
      coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
  }

  {
    i=N_i; j=N_j;
      coef.a_n[i][j] = 0.0 ;
      coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
      coef.a_e[i][j] = 0.0 ;
      coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
      coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
  }

  i=1;
  for (j=2;j<N_j;j++)
  {
      coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
      coef.a_s[i][j] = 0.0 ;
      coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
      coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
      coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
  }

  i=N_i;
  for (j=2;j<N_j;j++)
  {
      coef.a_n[i][j] = 0.0 ;
      coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
      coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
      coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
      coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
  }

  j=1;
  for (i=2;i<N_i;i++)
  {
      coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
      coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
      coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
      coef.a_w[i][j] = 0.0 ;
      coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
  }

  j=N_j;
  for (i=2;i<N_i;i++)
  {
      coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
      coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
      coef.a_e[i][j] = 0.0 ;
      coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
      coef.a_p[i][j] = +coef.a_n[i][j] + coef.a_s[i][j] + coef.a_e[i][j] + coef.a_w[i][j] ;
  }
}

void caseConv2D::Eval_coef_variables_P()
{
  unsigned int i , j;

  for(i=2;i<N_i;i++)
  {
    for(j=2;j<N_j;j++)
    {
      coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
```

```
                 +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
    }
  }

  {
    i=1; j=1;
      coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                 +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
  }

  {
    i=1; j=N_j;
      coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                 +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
  }

  {
    i=N_i; j=1;
      coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                 +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
  }

  {
    i=N_i; j=N_j;
      coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                 +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
  }

  i=1;
  for (j=2;j<N_j;j++)
  {
      coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                   +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
  }

  i=N_i;
  for (j=2;j<N_j;j++)
  {
      coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                 +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
  }

  j=1;
  for (i=2;i<N_i;i++)
  {
      coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                 +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
  }

  j=N_j;
  for (i=2;i<N_i;i++)
  {
      coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
                 +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
  }
}

void caseConv2D::solveGS()
{
  unsigned int i , j;
  unsigned int iter;
  double deltap_ = epsilon_pT + 1.0;

  for(iter=1; deltap_> epsilon_pT && iter<max_iter ; iter++)
  {
    for(i=1;i<N_i+1;i++)
    {
      for(j=1;j<N_j+1;j++)
      {
        p_1[i][j] = 1.0/coef.a_p[i][j] * ( coef.a_n[i][j] * p_1sup[i+1][j] + coef.a_s[i][j]
              * p_1[i-1][j] + coef.a_e[i][j] * p_1sup[i][j+1] + coef.a_w[i][j] * p_1[i][j-1]
            + coef.b_p[i][j]);
      }
```

```
    }
    deltap_ = delta_inc( p_1 , p_1sup );
    p_1sup = p_1 ;
  }
  if(iter>=max_iter) cerr << "Warning: max number of iterations achieved: " << deltap_ << endl;
  assert (deltap_ < epsilon_pT);
}

void caseConv2D::solveTDMA()
{
  unsigned int i , j;
  unsigned int iter;
  double deltap_ = epsilon_pT + 1.0;
  vector<double> P_x(N_j+1,0.0) , R_x(N_j+1,0.0) , b_x(N_j+1,0.0);
  vector<double> P_y(N_i+1,0.0) , R_y(N_i+1,0.0) , b_y(N_i+1,0.0);

  for(iter=1; deltap_>epsilon_pT && iter<max_iter ; iter++)
  {

    for(i=1;i<N_i+1;i++)
    {
      for(j=1;j<N_j+1;j++)
      {
        b_x[j] = coef.b_p[i][j] +  coef.a_n[i][j] * p_1sup[i+1][j] + coef.a_s[i][j] * p_1[i-1][j] ;
      }

      j=1;
      P_x[j] = coef.a_e[i][j] / coef.a_p[i][j];
      R_x[j] = b_x[j] / coef.a_p[i][j];

      for (j=2;j<N_j+1;j++)
      {
        P_x[j] = coef.a_e[i][j] / (coef.a_p[i][j] - coef.a_w[i][j]*P_x[j-1]);
        R_x[j] = (b_x[j] + coef.a_w[i][j]*R_x[j-1]) / (coef.a_p[i][j] - coef.a_w[i][j]*P_x[j-1]);
      }

      for (j=N_j;j>=1;j--)
      {
        p_1[i][j]=P_x[j]*p_1[i][j+1]+R_x[j];
      }
    }

    p_1sup = p_1 ;

    for(j=1;j<N_j+1;j++)
    {
      for(i=1;i<N_i+1;i++)
      {
        b_y[i] = coef.b_p[i][j] +  coef.a_e[i][j] * p_1sup[i][j+1] + coef.a_w[i][j] * p_1[i][j-1] ;
      }

      i=1;
      P_y[i] = coef.a_n[i][j] / coef.a_p[i][j];
      R_y[i] = b_y[i] / coef.a_p[i][j];

      for (i=2;i<N_i+1;i++)
      {
        P_y[i] = coef.a_n[i][j] / (coef.a_p[i][j] - coef.a_s[i][j]*P_y[i-1]);
        R_y[i] = (b_y[i] + coef.a_s[i][j]*R_y[i-1]) / (coef.a_p[i][j] - coef.a_s[i][j]*P_y[i-1]);
      }

      for (i=N_i;i>=1;i--)
      {
        p_1[i][j]=P_y[i]*p_1[i+1][j]+R_y[i];
      }
    }

    deltap_ = delta_inc( p_1 , p_1sup );
```

```
    p_1sup = p_1 ;
  }
  if(iter>=max_iter) cerr << "Warning: max number of iterations achieved: " << deltap_ << endl;
  assert (deltap_ < epsilon_pT);

}

void caseConv2D::Evaldiffusion_T()
{
  unsigned int i , j ;

  for (i=1;i<N_i+1;i++)
  {
    for (j=1;j<N_j+1;j++)
    {
      diff_T.D_e[i][j] = S_e[i][j] / ( ( x[i][j+1]-x[i][j] ) * ( sqrt(Ra) ) ) ;
      diff_T.D_w[i][j] = S_w[i][j] / ( ( x[i][j]-x[i][j-1] ) * ( sqrt(Ra) ) ) ;
      diff_T.D_n[i][j] = S_n[i][j] / ( ( y[i+1][j]-y[i][j] ) * ( sqrt(Ra) ) ) ;
      diff_T.D_s[i][j] = S_s[i][j] / ( ( y[i][j]-y[i-1][j] ) * ( sqrt(Ra) ) ) ;
    }
  }
}

void caseConv2D::Evalmassflow_T()
{
  unsigned int i , j ;

  for (i=1;i<N_i+1;i++)
  {
    for (j=1;j<N_j+1;j++)
    {
      mass_T.F_e[i][j] = u_2[i][j] * S_e[i][j] ;
      mass_T.F_w[i][j] = u_2[i][j-1] * S_w[i][j] ;
      mass_T.F_n[i][j] = v_2[i][j] * S_n[i][j] ;
      mass_T.F_s[i][j] = v_2[i-1][j] * S_s[i][j] ;
    }
  }
}

void caseConv2D::Evalpeclet_T()
{
  unsigned int i , j ;

  for (i=1;i<N_i+1;i++)
  {
    for (j=1;j<N_j+1;j++)
    {
      pe_T.Pe_e[i][j] = mass_T.F_e[i][j] / diff_T.D_e[i][j] ;
      pe_T.Pe_w[i][j] = mass_T.F_w[i][j] / diff_T.D_w[i][j] ;
      pe_T.Pe_n[i][j] = mass_T.F_n[i][j] / diff_T.D_n[i][j] ;
      pe_T.Pe_s[i][j] = mass_T.F_s[i][j] / diff_T.D_s[i][j] ;
    }
  }
}

void caseConv2D::EvalCoef_T()
{
  unsigned int i , j;

  for(i=2;i<N_i;i++)
  {
    for(j=2;j<N_j;j++)
    {
      coef_T.a_n[i][j] = diff_T.D_n[i][j] * A_scheme(pe_T.Pe_n[i][j]) + ((0.0>(-mass_T.F_n[i][j]))?0.0:(-mass_T.F_n[i][j])) ;
      coef_T.a_s[i][j] = diff_T.D_s[i][j] * A_scheme(pe_T.Pe_s[i][j]) + ((0.0>(mass_T.F_s[i][j]))?0.0:(mass_T.F_s[i][j])) ;
      coef_T.a_e[i][j] = diff_T.D_e[i][j] * A_scheme(pe_T.Pe_e[i][j]) + ((0.0>(-mass_T.F_e[i][j]))?0.0:(-mass_T.F_e[i][j])) ;
      coef_T.a_w[i][j] = diff_T.D_w[i][j] * A_scheme(pe_T.Pe_w[i][j]) + ((0.0>(mass_T.F_w[i][j]))?0.0:(mass_T.F_w[i][j])) ;
      coef_T.a_p[i][j] = coef_T.a_n[i][j] + coef_T.a_s[i][j] + coef_T.a_e[i][j] + coef_T.
```

```
            a_w[i][j] + Vol[i][j] / At ;
        coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
      }
    }
}

void caseConv2D::Eval_boundary_CoefbcT_temperature(enum T_BccValues bcc[4] , const vector
    <double> &value)
{
  unsigned int i , j ;

  switch (bcc[0]) //left
  {
    case Neumann:
      j=0;
      for(i=0;i<N_i+2;i++)
      {
        T_1sup[i][j] = 0.0;
        T_1[i][j] = 0.0;
      }
      {
        i=1; j=1; coef_T.a_w[i][j] = 0.0 ;
      }
      j=1;
      for (i=2;i<N_i;i++)
      {
        coef_T.a_w[i][j] = 0.0 ;
      }
      {
        i=N_i; j=1; coef_T.a_w[i][j] = 0.0 ;
      }

      break;
    case Dirichlet:
      j=0;
      for(i=0;i<N_i+2;i++)
      {
        T_1sup[i][j] = value[0];
        T_1[i][j] = value[0];
      }
      {
        i=1; j=1; coef_T.a_w[i][j] = diff_T.D_w[i][j] * A_scheme(pe_T.Pe_w[i][j]) + ((0.0>(
            mass_T.F_w[i][j]))?0.0:(mass_T.F_w[i][j])) ;
      }
      j=1;
      for (i=2;i<N_i;i++)
      {
        coef_T.a_w[i][j] = diff_T.D_w[i][j] * A_scheme(pe_T.Pe_w[i][j]) + ((0.0>(mass_T.F_w
            [i][j]))?0.0:(mass_T.F_w[i][j])) ;
      }
      {
        i=N_i; j=1; coef_T.a_w[i][j] = diff_T.D_w[i][j] * A_scheme(pe_T.Pe_w[i][j]) +
            ((0.0>(mass_T.F_w[i][j]))?0.0:(mass_T.F_w[i][j])) ;
      }

      break;
    default:
      assert(false);
      break;
  }
  switch (bcc[1])//bottom
  {
    case Neumann:
      i=0;
      for(j=0;j<N_j+2;j++)
      {
        T_1sup[i][j] = 0.0;
        T_1[i][j] = 0.0;
      }
      {
        i=1; j=1; coef_T.a_s[i][j] = 0.0 ;
      }
      {
```

```
            i=1; j=N_j; coef_T.a_s[i][j] = 0.0 ;
        }
        i=1;
        for (j=2;j<N_j;j++)
        {
            coef_T.a_s[i][j] = 0.0 ;
        }

        break;
    case Dirichlet:
        i=0;
        for(j=0;j<N_j+2;j++)
        {
            T_1sup[i][j] = value[1];
            T_1[i][j] = value[1];
        }
        {
            i=1; j=1; coef_T.a_s[i][j] = diff_T.D_s[i][j] * A_scheme(pe_T.Pe_s[i][j]) + ((0.0>(
                mass_T.F_s[i][j]))?0.0:(mass_T.F_s[i][j])) ;
        }
        {
            i=1; j=N_j; coef_T.a_s[i][j] = diff_T.D_s[i][j] * A_scheme(pe_T.Pe_s[i][j]) +
                ((0.0>(mass_T.F_s[i][j]))?0.0:(mass_T.F_s[i][j])) ;
        }
        i=1;
        for (j=2;j<N_j;j++)
        {
            coef_T.a_s[i][j] = diff_T.D_s[i][j] * A_scheme(pe_T.Pe_s[i][j]) + ((0.0>(mass_T.F_s
                [i][j]))?0.0:(mass_T.F_s[i][j])) ;
        }

        break;
    default:
        assert(false);
        break;
}
switch (bcc[2])//right
{
    case Neumann:
        j=N_j+1;
        for(i=0;i<N_i+2;i++)
        {
            T_1sup[i][j] = 0.0;
            T_1[i][j] = 0.0;
        }
        {
            i=N_i; j=N_j; coef_T.a_e[i][j] = 0.0 ;
        }
        j=N_j;
        for (i=2;i<N_i;i++)
        {
            coef_T.a_e[i][j] = 0.0 ;
        }
        {
            i=1; j=N_j; coef_T.a_e[i][j] = 0.0 ;
        }

        break;
    case Dirichlet:
        j=N_j+1;
        for(i=0;i<N_i+2;i++)
        {
            T_1sup[i][j] = value[2];
            T_1[i][j] = value[2];
        }
        {
            i=N_i; j=N_j; coef_T.a_e[i][j] = diff_T.D_e[i][j] * A_scheme(pe_T.Pe_e[i][j]) +
                ((0.0>(-mass_T.F_e[i][j]))?0.0:(-mass_T.F_e[i][j])) ;
        }
        j=N_j;
        for (i=2;i<N_i;i++)
        {
            coef_T.a_e[i][j] = diff_T.D_e[i][j] * A_scheme(pe_T.Pe_e[i][j]) + ((0.0>(-mass_T.
```

```
F_e[i][j]))?0.0:(-mass_T.F_e[i][j])) ;
    }
    {
      i=1; j=N_j; coef_T.a_e[i][j] = diff_T.D_e[i][j] * A_scheme(pe_T.Pe_e[i][j]) +
           ((0.0>(-mass_T.F_e[i][j]))?0.0:(-mass_T.F_e[i][j])) ;
    }

    break;
  default:
    assert(false);
    break;
}
switch (bcc[3])//top
{
  case Neumann:
    i=N_i+1;
    for(j=0;j<N_j+2;j++)
    {
      T_1sup[i][j] = 0.0;
      T_1[i][j] = 0.0;
    }
    {
      i=N_i; j=1; coef_T.a_n[i][j] = 0.0 ;
    }
    i=N_i;
    for (j=2;j<N_j;j++)
    {
      coef_T.a_n[i][j] = 0.0 ;
    }
    {
      i=N_i; j=N_j; coef_T.a_n[i][j] = 0.0 ;
    }

    break;
  case Dirichlet:
    i=N_i+1;
    for(j=0;j<N_j+2;j++)
    {
      T_1sup[i][j] = value[3];
      T_1[i][j] = value[3];
    }
    {
      i=N_i; j=1; coef_T.a_n[i][j] = diff_T.D_n[i][j] * A_scheme(pe_T.Pe_n[i][j]) +
           ((0.0>(-mass_T.F_n[i][j]))?0.0:(-mass_T.F_n[i][j])) ;
    }
    i=N_i;
    for (j=2;j<N_j;j++)
    {
      coef_T.a_n[i][j] = diff_T.D_n[i][j] * A_scheme(pe_T.Pe_n[i][j]) + ((0.0>(-mass_T.
           F_n[i][j]))?0.0:(-mass_T.F_n[i][j])) ;
    }
    {
      i=N_i; j=N_j; coef_T.a_n[i][j] = diff_T.D_n[i][j] * A_scheme(pe_T.Pe_n[i][j]) +
           ((0.0>(-mass_T.F_n[i][j]))?0.0:(-mass_T.F_n[i][j])) ;
    }

    break;
  default:
    assert(false);
    break;
}

{
  i=1; j=1;
  coef_T.a_n[i][j] = diff_T.D_n[i][j] * A_scheme(pe_T.Pe_n[i][j]) + ((0.0>(-mass_T.F_n[i
       ][j]))?0.0:(-mass_T.F_n[i][j])) ;
  coef_T.a_e[i][j] = diff_T.D_e[i][j] * A_scheme(pe_T.Pe_e[i][j]) + ((0.0>(-mass_T.F_e[i
       ][j]))?0.0:(-mass_T.F_e[i][j])) ;
  coef_T.a_p[i][j] = coef_T.a_n[i][j] + coef_T.a_s[i][j] + coef_T.a_e[i][j] + coef_T.a_w[
       i][j] +  Vol[i][j] / At ;
  coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
}
```

```
    j=1;
    for (i=2;i<N_i;i++)
    {
      coef_T.a_n[i][j] = diff_T.D_n[i][j] * A_scheme(pe_T.Pe_n[i][j]) + ((0.0>(-mass_T.F_n[i][j]))?0.0:(-mass_T.F_n[i][j])) ;
      coef_T.a_s[i][j] = diff_T.D_s[i][j] * A_scheme(pe_T.Pe_s[i][j]) + ((0.0>(mass_T.F_s[i][j]))?0.0:(mass_T.F_s[i][j])) ;
      coef_T.a_e[i][j] = diff_T.D_e[i][j] * A_scheme(pe_T.Pe_e[i][j]) + ((0.0>(-mass_T.F_e[i][j]))?0.0:(-mass_T.F_e[i][j])) ;
      coef_T.a_p[i][j] = coef_T.a_n[i][j] + coef_T.a_s[i][j] + coef_T.a_e[i][j] + coef_T.a_w[i][j] +  Vol[i][j] / At ;
      coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
    }

    {
      i=N_i; j=1;
      coef_T.a_s[i][j] = diff_T.D_s[i][j] * A_scheme(pe_T.Pe_s[i][j]) + ((0.0>(mass_T.F_s[i][j]))?0.0:(mass_T.F_s[i][j])) ;
      coef_T.a_e[i][j] = diff_T.D_e[i][j] * A_scheme(pe_T.Pe_e[i][j]) + ((0.0>(-mass_T.F_e[i][j]))?0.0:(-mass_T.F_e[i][j])) ;
      coef_T.a_p[i][j] = coef_T.a_n[i][j] + coef_T.a_s[i][j] + coef_T.a_e[i][j] + coef_T.a_w[i][j] + Vol[i][j] / At ;
      coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
    }

    i=N_i;
    for (j=2;j<N_j;j++)
    {
      coef_T.a_s[i][j] = diff_T.D_s[i][j] * A_scheme(pe_T.Pe_s[i][j]) + ((0.0>(mass_T.F_s[i][j]))?0.0:(mass_T.F_s[i][j])) ;
      coef_T.a_e[i][j] = diff_T.D_e[i][j] * A_scheme(pe_T.Pe_e[i][j]) + ((0.0>(-mass_T.F_e[i][j]))?0.0:(-mass_T.F_e[i][j])) ;
      coef_T.a_w[i][j] = diff_T.D_w[i][j] * A_scheme(pe_T.Pe_w[i][j]) + ((0.0>(mass_T.F_w[i][j]))?0.0:(mass_T.F_w[i][j])) ;
      coef_T.a_p[i][j] = coef_T.a_n[i][j] + coef_T.a_s[i][j] + coef_T.a_e[i][j] + coef_T.a_w[i][j] + Vol[i][j] / At ;
      coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
    }

    {
      i=N_i; j=N_j;
      coef_T.a_s[i][j] = diff_T.D_s[i][j] * A_scheme(pe_T.Pe_s[i][j]) + ((0.0>(mass_T.F_s[i][j]))?0.0:(mass_T.F_s[i][j])) ;
      coef_T.a_w[i][j] = diff_T.D_w[i][j] * A_scheme(pe_T.Pe_w[i][j]) + ((0.0>(mass_T.F_w[i][j]))?0.0:(mass_T.F_w[i][j])) ;
      coef_T.a_p[i][j] = coef_T.a_n[i][j] + coef_T.a_s[i][j] + coef_T.a_e[i][j] + coef_T.a_w[i][j] + Vol[i][j] / At ;
      coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
    }

    j=N_j;
    for (i=2;i<N_i;i++)
    {
      coef_T.a_n[i][j] = diff_T.D_n[i][j] * A_scheme(pe_T.Pe_n[i][j]) + ((0.0>(-mass_T.F_n[i][j]))?0.0:(-mass_T.F_n[i][j])) ;
      coef_T.a_s[i][j] = diff_T.D_s[i][j] * A_scheme(pe_T.Pe_s[i][j]) + ((0.0>(mass_T.F_s[i][j]))?0.0:(mass_T.F_s[i][j])) ;
      coef_T.a_w[i][j] = diff_T.D_w[i][j] * A_scheme(pe_T.Pe_w[i][j]) + ((0.0>(mass_T.F_w[i][j]))?0.0:(mass_T.F_w[i][j])) ;
      coef_T.a_p[i][j] = coef_T.a_n[i][j] + coef_T.a_s[i][j] + coef_T.a_e[i][j] + coef_T.a_w[i][j] + Vol[i][j] / At ;
      coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
    }

    {
      i=1; j=N_j;
      coef_T.a_n[i][j] = diff_T.D_n[i][j] * A_scheme(pe_T.Pe_n[i][j]) + ((0.0>(-mass_T.F_n[i][j]))?0.0:(-mass_T.F_n[i][j])) ;
      coef_T.a_w[i][j] = diff_T.D_w[i][j] * A_scheme(pe_T.Pe_w[i][j]) + ((0.0>(mass_T.F_w[i][j]))?0.0:(mass_T.F_w[i][j])) ;
      coef_T.a_p[i][j] = coef_T.a_n[i][j] + coef_T.a_s[i][j] + coef_T.a_e[i][j] + coef_T.a_w[i][j] + Vol[i][j] / At ;
      coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
```

```
  }

  i=1;
  for (j=2;j<N_j;j++)
  {
    coef_T.a_n[i][j] = diff_T.D_n[i][j] * A_scheme(pe_T.Pe_n[i][j]) + ((0.0>(-mass_T.F_n[i
        ][j]))?0.0:(-mass_T.F_n[i][j])) ;
    coef_T.a_e[i][j] = diff_T.D_e[i][j] * A_scheme(pe_T.Pe_e[i][j]) + ((0.0>(-mass_T.F_e[i
        ][j]))?0.0:(-mass_T.F_e[i][j])) ;
    coef_T.a_w[i][j] = diff_T.D_w[i][j] * A_scheme(pe_T.Pe_w[i][j]) + ((0.0>(mass_T.F_w[i][
        j]))?0.0:(mass_T.F_w[i][j])) ;
    coef_T.a_p[i][j] = coef_T.a_n[i][j] + coef_T.a_s[i][j] + coef_T.a_e[i][j] + coef_T.a_w[
        i][j] + Vol[i][j] / At ;
    coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
  }
}

void caseConv2D::solveTDMA_Temperature()
{
  unsigned int i , j;
  unsigned int iter;
  double deltaT = epsilon_pT + 1.0;
  vector<double> P_x(N_j+1,0.0) , R_x(N_j+1,0.0) , b_x(N_j+1,0.0);
  vector<double> P_y(N_i+1,0.0) , R_y(N_i+1,0.0) , b_y(N_i+1,0.0);

  for(iter=1; deltaT>epsilon_pT && iter<max_iter ; iter++)
  {

    for(i=1;i<N_i+1;i++)
    {
      for(j=1;j<N_j+1;j++)
      {
        b_x[j] = coef_T.b_p[i][j] +  coef_T.a_n[i][j] * T_1sup[i+1][j] + coef_T.a_s[i][j] *
            T_1sup[i-1][j] ;
      }

      j=1;
      P_x[j] = coef_T.a_e[i][j] / coef_T.a_p[i][j];
      R_x[j] = b_x[j] / coef_T.a_p[i][j];

      for (j=2;j<N_j+1;j++)
      {
        P_x[j] = coef_T.a_e[i][j] / (coef_T.a_p[i][j] - coef_T.a_w[i][j]*P_x[j-1]);
        R_x[j] = (b_x[j] + coef_T.a_w[i][j]*R_x[j-1]) / (coef_T.a_p[i][j] - coef_T.a_w[i][j
            ]*P_x[j-1]);
      }

      for (j=N_j;j>=1;j--)
      {
        T_1[i][j] = P_x[j] * T_1[i][j+1] + R_x[j];
      }
    }

    T_1sup = T_1 ;

    for(j=1;j<N_j+1;j++)
    {
      for(i=1;i<N_i+1;i++)
      {
        b_y[i] = coef_T.b_p[i][j] +  coef_T.a_e[i][j] * T_1sup[i][j+1] + coef_T.a_w[i][j] *
            T_1sup[i][j-1] ;
      }

      i=1;
      P_y[i] = coef_T.a_n[i][j] / coef_T.a_p[i][j];
      R_y[i] = b_y[i] / coef_T.a_p[i][j];

      for (i=2;i<N_i+1;i++)
      {
        P_y[i] = coef_T.a_n[i][j] / (coef_T.a_p[i][j] - coef_T.a_s[i][j]*P_y[i-1]);
        R_y[i] = (b_y[i] + coef_T.a_s[i][j]*R_y[i-1]) / (coef_T.a_p[i][j] - coef_T.a_s[i][j
            ]*P_y[i-1]);
      }
```

```

      for (i=N_i; i>=1;i--)
      {
        T_1[i][j] = P_y[i] * T_1[i+1][j] + R_y[i];
      }
    }

    deltaT = delta_inc( T_1 , T_1sup );
    T_1sup = T_1 ;
  }
  if(iter>=max_iter) cerr << "Warning: max number of iterations achieved: " << deltaT << endl;
  assert (deltaT < epsilon_pT);
}

void caseConv2D::solveGS_Temp()
{
  unsigned int i , j;
  unsigned int iter;
  double deltaT = epsilon_pT + 1.0;

  for(iter=1; deltaT > epsilon_pT && iter<max_iter ; iter++)
  {
    for(i=1;i<N_i+1;i++)
    {
      for(j=1;j<N_j+1;j++)
      {
        T_1[i][j] = 1.0/coef_T.a_p[i][j] * ( coef_T.a_n[i][j] * T_1sup[i+1][j] + coef_T.a_s[i][j] * T_1[i-1][j]
                + coef_T.a_e[i][j] * T_1sup[i][j+1] + coef_T.a_w[i][j] * T_1[i][j-1] + coef_T.b_p[i][j]);
      }
    }
    deltaT = delta_inc( T_1 , T_1sup );
    T_1sup = T_1 ;
  }
  if(iter>=max_iter) cerr << "Warning: max number of iterations achieved: " << deltaT << endl;
  assert (deltaT < epsilon_pT);
}

void caseConv2D::EvalTime()
{
  double v_condx_max=0.0 , v_condy_max=0.0 , At_conv=0.0;
  double mu_condx_max=0.0 , mu_condy_max=0.0 , At_visc=0.0;

  unsigned int i,j;

  for (i=1;i<N_i+1;i++)
  {
    for(j=1;j<N_j;j++)
    {
      v_condx_max = (v_condx_max > fabs(u_1[i][j]/(x[i][j+1] - x[i][j])) )
      ? v_condx_max : fabs(u_1[i][j]/(x[i][j+1] - x[i][j])) ;
    }
  }

  for (i=1;i<N_i;i++)
  {
    for(j=1;j<N_j+1;j++)
    {
      v_condy_max = (v_condy_max > fabs(v_1[i][j]/(y[i+1][j] - y[i][j])) )
      ? v_condy_max : fabs(v_1[i][j]/(y[i+1][j] - y[i][j])) ;
    }
  }

  for (i=1;i<N_i+1;i++)
  {
    for(j=1;j<N_j+1;j++)
    {
      mu_condx_max = (mu_condx_max > fabs(mu[i][j]/rho[i][j]/(x_e[i][j] - x_w[i][j])/(x_e[i][j] - x_w[i][j])) )
      ? mu_condx_max : fabs(mu[i][j]/rho[i][j]/(x_e[i][j] - x_w[i][j])/(x_e[i][j] - x_w[i][
```

```
            j]));

        mu_condy_max = (mu_condy_max > fabs(mu[i][j]/rho[i][j]/(y_n[i][j] - y_s[i][j])/(y_n[i][j] - y_s[i][j])) )
        ? mu_condy_max : fabs(mu[i][j]/rho[i][j]/(y_n[i][j] - y_s[i][j])/(y_n[i][j] - y_s[i][j]));
      }
    }

    At_conv = (v_condx_max>v_condy_max) ? 0.35/v_condx_max : 0.35/v_condy_max;
    At_visc = (mu_condx_max>mu_condy_max) ? 0.2/mu_condx_max : 0.2/mu_condy_max;

    At = (At_conv<At_visc) ? At_conv : At_visc ;

    t_2 += At;
}

void caseConv2D::EvalVelocityfield()
{
    unsigned int i , j;

    for(i=1;i<N_i+1;i++)
    {
      for(j=1;j<N_j;j++)
      {

        u_2[i][j] = u_p2[i][j] - ( p_1[i][j+1] - p_1[i][j] )/(x[i][j+1] - x[i][j]);
      }
    }

    for(i=1;i<N_i;i++)
    {
      for(j=1;j<N_j+1;j++)
      {
        v_2[i][j] = v_p2[i][j] - ( p_1[i+1][j] - p_1[i][j] )/(y[i+1][j] - y[i][j]);
      }
    }
}

void caseConv2D::Massconservation()
{
    unsigned int i , j;

    for(i=1;i<N_i+1;i++)
    {
      for(j=1;j<N_j+1;j++)
      {
        assert ( fabs(u_2[i][j] * S_e[i][j] + v_2[i][j] * S_n[i][j] - u_2[i][j-1] * S_w[i][j]
              - v_2[i-1][j] * S_s[i][j])/Vol[i][j] < 1E-1);
      }
    }
}

double caseConv2D::GetConvergence()
{
    unsigned int i,j;
    double maximum=0.0;

    maximum = delta_inc ( u_1 , u_2 );
    maximum = (maximum > delta_inc ( v_1 , v_2 ) ) ? maximum : delta_inc ( v_1 , v_2 ) ;
    maximum = (maximum > delta_inc ( T_0 , T_1 ) ) ? maximum : delta_inc ( T_0 , T_1 ) ;

    return maximum/At;
}

void caseConv2D::NewStep()
{

    u_0 = u_1 ;
    v_0 = v_1 ;

    u_1 = u_2 ;
    v_1 = v_2 ;
```

```
  p_0 = p_1 ;

  R_u0 = R_u1 ;
  R_v0 = R_v1 ;

  T_0 = T_1 ;
}

void caseConv2D::saveValues(double conv , unsigned int iter)
{
  unsigned int i,j;
  string outputname;
  string temp;

  stringstream ss(stringstream::in | stringstream::out);
  ss.setf(ios::fixed);
  ss.precision(0);
  ss << iter ;
  ss >> temp ;

  outputname = "output_" + temp + ".m";
  ofstream outfile(outputname.c_str());

    if(outfile.fail())
    {
      cerr << "Error creating output.m file. " << endl;
      exit(-1);
    }

    outfile.setf(ios::floatfield,ios::scientific);
      outfile.precision(10);

      for (i=1;i<N_i+1;i++)
    {
      for (j=1;j<N_j+1;j++)
      {
        u_out[i][j] =  (u_2[i][j]+u_2[i][j-1])/2 * sqrt(Ra);
      }
    }

    for (i=1;i<N_i+1;i++)
    {
      for (j=1;j<N_j+1;j++)
      {
        v_out[i][j] = (v_2[i][j]+v_2[i-1][j])/2 * sqrt(Ra);
      }
    }

    double Nu_average = 0.0;
    j = 0;
    for (i=1;i<N_i+1;i++)
    {
      Nu_average += - (T_1[i][j+1]-T_1[i][j])/(x[i][j+1]-x[i][j]) * S_e[i][j+1] / L_y ;
    }

    //Nusselt number expected
    double Nu_expected = 0.0 , Nu_60 = 0.0 , Nu_90 = 0.0 ;

    if (angle<=60.0)
    {
      Nu_expected = 1.0 + 1.44 * max(0.0, 1.0 - 1708.0/(Ra * cos (angle * PI /180))) * (1.0 - 1708.0 * pow(sin(1.8*angle*PI/180.0),1.6)/(Ra * cos (angle * PI /180.0)))
        + max(0.0,pow((Ra * cos(angle*PI/180.0) / 5830.0 ),1.0/3.0)-1.0);
    }
    else
    {
      Nu_60 = 1.0 + 1.44 * max(0.0, 1.0 - 1708.0/(Ra * cos (60.0 * PI /180))) * (1.0 - 1708.0 * pow(sin(1.8*60.*PI/180.0),1.6)/(Ra * cos (60.0 * PI /180.0)))
        + max(0.0,pow((Ra * cos(60.0*PI/180.0) / 5830.0 ),1.0/3.0)-1.0);

      Nu_90 = sqrt( 1.0 + pow( (0.066*pow(Ra,1.0/3.0))/(1.0+pow(9000.0/Ra,1.0/4.0) ), 2.0 ) );
```

```
        Nu_expected = (90.0-angle)/30.0 * Nu_60 + (angle-60.0)/30.0 * Nu_90;
      }

      outfile << "%Time: " << t_2 << " s" << endl;
      outfile << "%Derivative value: " << conv << endl << endl;
      outfile << "%Nusselt number at x = 0.0 : " << Nu_average << endl;
      outfile << "%Nusselt number expected " <<  Nu_expected << endl;

      outfile << "u=[" << endl;
      for (i=1;i<N_i+1;i++)
      {
        for (j=1;j<N_j+1;j++)
        {
          outfile << u_out[i][j] << " " ;
        }
        outfile << endl ;
      }
      outfile << "];" << endl;

      outfile << endl << endl;
      outfile << "v=[" << endl;
      for (i=1;i<N_i+1;i++)
      {
        for (j=1;j<N_j+1;j++)
        {
          outfile << v_out[i][j] << " " ;
        }
        outfile << endl ;
      }
      outfile << "];" << endl;

      outfile << endl << endl;
      outfile << "p=[" << endl;
      for (i=1;i<N_i+1;i++)
      {
        for (j=1;j<N_j+1;j++)
        {
          outfile << p_1[i][j] / At << " " ;
        }
        outfile << endl ;
      }
      outfile << "];" << endl;

      outfile << endl << endl;
      outfile << "T=[" << endl;
      for (i=1;i<N_i+1;i++)
      {
        for (j=1;j<N_j+1;j++)
        {
          outfile << T_1[i][j] << " " ;
        }
        outfile << endl ;
      }
      outfile << "];" << endl;
    outfile.close();
}

double caseConv2D::Nu_value()
{
    unsigned int i , j;
    double Nu_average = 0.0;
    j = 0;
    for (i=1;i<N_i+1;i++)
    {
      Nu_average += - (T_1[i][j+1]-T_1[i][j])/(x[i][j+1]-x[i][j]) * S_e[i][j+1] / L_y ;
    }

    return Nu_average;
}

double delta_inc (const vector<vector<double> > &A , const vector<vector<double> > &B )
{
```

```
  unsigned int i , j;
  double temporal = 0.0 , delta = 0.0;

  assert(A.size()==B.size() && A[1].size()==B[1].size());

  for(i=1;i<A.size()-1;i++)
  {
    for(j=1;j<A[1].size()-1;j++)
    {
        temporal = fabs(A[i][j] - B[i][j]) ;
        delta = ( (delta > temporal ) ? delta : temporal );
    }
  }
  return delta;
}
```

1.2 Numerical scheme selection program

```
#include <iostream>
#include <cmath>

using namespace std;

enum schValues {
    CDS,
    UDS,
    HDS,
    PLDS,
    EDS
};

enum schValues str2schValues(string str){
  if(str=="CDS") return(CDS);
  else if(str=="UDS") return(UDS);
  else if(str=="HDS") return(HDS);
  else if(str=="PLDS") return(PLDS);
  else if(str=="EDS") return(EDS);
  else {
    cerr<<"Error␣in␣"<<__FUNCTION__<<":␣"<<str<<"␣condition␣doesn't␣exist."<<endl;
  }
}

double A_scheme (double P, string scheme="CDS"){
  schValues sch;

  sch=str2schValues(scheme);

  switch (sch){
    case CDS: //Central diference
      return 1-0.5*fabs(P);
    case UDS: //Upwind
      return 1;
    case HDS: //Hybrid
      return (0 > (1 - 0.5*fabs(P)) )? 0 : (1 - 0.5*fabs(P));
    case PLDS: //power law
      return ( 0 > pow((1 - 0.1*fabs(P)) , 5) )? 0 : pow((1 - 0.1*fabs(P)) , 5);
    case EDS: //exponential
      return fabs(P) / ( exp(fabs(P)) - 1 );
    default:
      cerr << "Wrong␣scheme" << endl;
      break;
  }
}
```

1.3 Conjugate gradient solver

```
#include <iostream>
#include <vector>
```

```
#include <cmath>
#include <cassert>

using namespace std;

double max_residu (const vector<vector<double> > r);

vector<vector<double> > conjugate_gradient ( const vector<vector<double> > &a_p , const
    vector<vector<double> > &a_e , const vector<vector<double> > &a_w , const vector<
    vector<double> > &a_n , const vector<vector<double> > &a_s , const vector<vector<
    double> > &phi_0 , const vector<vector<double> > &b , int max_iter , double epsilon
     , int N_x , int N_y)
{

  unsigned int i , j;
  unsigned int iter;
  unsigned int N=N_x*N_y;
  vector<vector<double> > r_0(N_y+2,vector<double>(N_x+2,0.0)) , r_1(N_y+2,vector<double
      >(N_x+2,0.0));
  vector<vector<double> > p_0(N_y+2,vector<double>(N_x+2,0.0)) , p_1(N_y+2,vector<double
      >(N_x+2,0.0));
  vector<vector<double> > x_0 = phi_0 , x(N_y+2,vector<double>(N_x+2,0.0));

  for (i=1;i<N_y+1;i++)
  {
    for(j=1;j<N_x+1;j++)
    {
      r_0[i][j] = r_0[i][j] + a_p[i][j]*x_0[i][j] - a_e[i][j]*x_0[i][j+1] - a_w[i][j]*x_0[i
          ][j-1] - a_n[i][j]*x_0[i+1][j] - a_s[i][j]*x_0[i-1][j] ;
      r_0[i][j] = r_0[i][j] - b[i][j];
      p_0[i][j] = - r_0[i][j];
    }
  }


  for(iter=1; max_residu(r_0) > epsilon && iter<max_iter ; iter++)
  {
    double a_0=0.0 ;
    double temp1 = 0.0 , temp2 = 0.0;
    for (i=1;i<N_y+1;i++)
    {
      for(j=1;j<N_x+1;j++)
      {
        temp1 = temp1 + r_0[i][j]*r_0[i][j];
        temp2 = temp2 + p_0[i][j]*(a_p[i][j]*p_0[i][j] -a_e[i][j]*p_0[i][j+1] - a_w[i][j]*
            p_0[i][j-1] - a_n[i][j]*p_0[i+1][j] - a_s[i][j]*p_0[i-1][j]);
      }
    }
    a_0 = temp1 / temp2 ;

    for (i=1;i<N_y+1;i++)
    {
      for(j=1;j<N_x+1;j++)
      {
        x[i][j] = x_0[i][j] + a_0 * p_0[i][j];
      }
    }

    for (i=1;i<N_y+1;i++)
    {
      for(j=1;j<N_x+1;j++)
      {

        r_1[i][j] = r_0[i][j] + a_0 *(a_p[i][j]*p_0[i][j] -a_e[i][j]*p_0[i][j+1] - a_w[i][j
            ]*p_0[i][j-1] - a_n[i][j]*p_0[i+1][j] - a_s[i][j]*p_0[i-1][j]);
      }
    }

    double b_1 = 0.0 ;
    temp1 = 0.0 , temp2=0.0 ;
    for (i=1;i<N_y+1;i++)
    {
      for(j=1;j<N_x+1;j++)
```

```
      {
        temp1 = temp1 + r_1[i][j] * r_1[i][j];
        temp2 = temp2 + r_0[i][j] * r_0[i][j];
      }
    }
    b_1 = temp1 / temp2 ;

    for (i=1;i<N_y+1;i++)
    {
      for(j=1;j<N_x+1;j++)
      {
        p_1[i][j] = -r_1[i][j] + b_1 * p_0[i][j];
      }
    }

    p_0 = p_1 ;
    r_0 = r_1 ;
    x_0 = x ;

  }
  if(iter>=max_iter) cerr << "Warning: max number of iterations achieved: " << max_residu(r_0) << endl;
  assert (max_residu(r_0) < epsilon);

  return x;

}

double max_residu (const vector<vector<double> > r)
{
  double temp = 0.0;
  unsigned int i , j;


  for (i=0;i<r.size();i++)
  {
    for (j=0;j<r[1].size();j++)
    {
      temp=(temp<fabs(r[i][j]))?fabs(r[i][j]):temp;
    }
  }

  return temp;

}
```

1.4 Biconjugate gradient stabilized solver

```
#include <iostream>
#include <vector>
#include <cmath>
#include <cassert>

using namespace std;

double max_residu (const vector<vector<double> > r);

vector<vector<double> > biconjugate_gradient_stb ( const vector<vector<double> > &a_p , const vector<vector<double> > &a_e , const vector<vector<double> > &a_w , const vector<vector<double> > &a_n , const vector<vector<double> > &a_s , const vector<vector<double> > &phi_0 , const vector<vector<double> > &b , int max_iter , double epsilon , int N_x , int N_y)
{

  unsigned int i , j;
  unsigned int iter;
  unsigned int N=N_x*N_y;
  vector<vector<double> > r_0tilde(N_y+2,vector<double>(N_x+2,0.0)), r_0(N_y+2,vector<double>(N_x+2,0.0)) , r_1(N_y+2,vector<double>(N_x+2,0.0));
  vector<vector<double> > p_0(N_y+2,vector<double>(N_x+2,0.0)) , p_1(N_y+2,vector<double>(N_x+2,0.0));
```

```
vector<vector<double> > v_0(N_y+2,vector<double>(N_x+2,0.0)) , v_1(N_y+2,vector<double
    >(N_x+2,0.0));
vector<vector<double> > t(N_y+2,vector<double>(N_x+2,0.0)) , s(N_y+2,vector<double>(N_x
    +2,0.0));
vector<vector<double> > x_0 = phi_0 , x = phi_0;
double rho_0 = 1.0 , rho_1 = 0.0 , alfa = 1.0 , beta = 0.0 , w_0 = 1.0 , w_1 = 0.0 ;

for (i=1;i<N_y+1;i++)
{
  for(j=1;j<N_x+1;j++)
  {
    x[i][j]=0.0;
  }
}

for (i=1;i<N_y+1;i++)
{
  for(j=1;j<N_x+1;j++)
  {
    r_0[i][j] = r_0[i][j] + a_p[i][j]*x_0[i][j] - a_e[i][j]*x_0[i][j+1] - a_w[i][j]*x_0[i
        ][j-1] - a_n[i][j]*x_0[i+1][j] - a_s[i][j]*x_0[i-1][j] ;
    r_0[i][j] = - r_0[i][j] + b[i][j];
  }
}
r_0tilde = r_0;

for(iter=1; max_residu(r_0) > epsilon && iter<max_iter ; iter++)
{
  rho_1 = 0.0 ;
  for (i=1;i<N_y+1;i++)
  {
    for(j=1;j<N_x+1;j++)
    {
      rho_1 = rho_1 + r_0tilde[i][j]*r_0[i][j];
    }
  }
  beta = rho_1 / rho_0 * alfa / w_0 ;

  for (i=1;i<N_y+1;i++)
  {
    for(j=1;j<N_x+1;j++)
    {
      p_1[i][j] = r_0[i][j] + beta * (p_0[i][j] - w_0*v_0[i][j]) ;
    }
  }

  double temp2 = 0.0 ;
  for (i=1;i<N_y+1;i++)
  {
    for(j=1;j<N_x+1;j++)
    {
      v_1[i][j] = (a_p[i][j]*p_1[i][j] - a_e[i][j]*p_1[i][j+1] - a_w[i][j]*p_1[i][j-1]
                  - a_n[i][j]*p_1[i+1][j] - a_s[i][j]*p_1[i-1][j]);
      temp2 = temp2 + r_0tilde[i][j] * v_1[i][j] ;
    }
  }
  alfa = rho_1 / temp2 ;

  for (i=1;i<N_y+1;i++)
  {
    for(j=1;j<N_x+1;j++)
    {
      s[i][j] = r_0[i][j] - alfa * v_1[i][j];
    }
  }

  for (i=1;i<N_y+1;i++)
  {
    for(j=1;j<N_x+1;j++)
    {
      t[i][j] = (a_p[i][j]*s[i][j] - a_e[i][j]*s[i][j+1] - a_w[i][j]*s[i][j-1]
                  - a_n[i][j]*s[i+1][j] - a_s[i][j]*s[i-1][j]);
    }
```

```
    }

    double temp3 = 0.0 , temp4 = 0.0;

    for (i=1;i<N_y+1;i++)
    {
      for(j=1;j<N_x+1;j++)
      {
        temp3 = temp3 + t[i][j] * s[i][j] ;
        temp4 = temp4 + t[i][j] * t[i][j] ;
      }
    }
    w_1 = temp3/temp4;

    for (i=1;i<N_y+1;i++)
    {
      for(j=1;j<N_x+1;j++)
      {
        x[i][j] = x_0[i][j] + alfa * p_1[i][j] + w_1 * s[i][j];
        r_1[i][j] = s[i][j] - w_1 * t[i][j] ;
      }
    }

    p_0 = p_1 ;
    v_0 = v_1 ;
    r_0 = r_1 ;
    x_0 = x ;
    rho_0 = rho_1 ;
    w_0 = w_1 ;

  }
  if(iter>=max_iter) cerr << "Warning: max number of iterations achieved: " << max_residu(
      r_0) << endl;
  assert (max_residu(r_0) < epsilon);

  return x;

}
```

1.5 Mesh coordinates program

```
#include <iostream>
#include <cmath>

using namespace std;

double det_coord_Mesh ( double A , unsigned int N , double g , unsigned int i )
{

  return A/2.0 * ( 1.0 + tanh( g*( 2.0*(i-1.0)/N - 1.0 ) ) / tanh(g) );

}
```

1.6 Input file

X_distance 1.0
Y_distance 50.0

Number_of_control_volumes_x 50
Number_of_control_volumes_y 500

```
Mesh_Type NonUniform 1.2 1.2

Maximum_iterations 100000
Epsilon_vT 1E-5
Epsilon_pT 1E-6
Time_to_end_(s) 500

Prandtl_number 0.71
Rayleigh_number 2.47e5
Angle_deg 90

left_velocity Wall
bottom_velocity Wall
right_velocity Wall
top_velocity Wall

left_temperature Dirichlet 1
bottom_temperature Neumann
right_temperature Dirichlet 0
top_temperature Neumann
```