

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



Escola Tècnica Superior d'Enginyeries
Industrial i Aeronàutica de Terrassa

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Universitat Politècnica de Catalunya

Escola Tècnica Superior d'Enginyeries Industrial i Aeronàutica de Terrassa

June, 2015

Contents

1	Code of the program	1
1.1	Main program	1
1.2	Numerical scheme selection program	31
1.3	Conjugate gradient solver	31
1.4	Biconjugate gradient stabilized solver	33
1.5	Mesh coordinates program	35
1.6	Input file	35

1 Code of the program

1.1 Main program

```

1  #include <iostream>
2  #include <sstream>
3  #include <fstream>
4  #include <vector>
5  #include <cmath>
6  #include <cassert>
7  #include <cstdlib>
8  #include <string>
9
10 #define PI 3.14159265
11
12 using namespace std;
13
14 enum R_u_BccValues {
15     Wall,
16     Airflow
17 };
18
19 enum T_BccValues {
20     Neumann,
21     Dirichlet
22 };
23
24 enum R_u_BccValues str2R_u_BccValues(string str);
25 enum T_BccValues str2T_BccValues(string str);
26 double delta_inc(const vector<vector<double>> &A , const vector<vector<double>> &B );
27 double A_scheme (double P, string scheme="CDS");
28 double det_coord_Mesh ( double A , unsigned int N , double g , unsigned int i );
29 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
30     , const vector<vector<double>> &phi_0 , const vector<vector<double>
    > &b , int max_iter , double epsilon , int N_x , int N_y);
31 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 ,
32     const vector<vector<double>> &phi_0 , const vector<vector<double>>
    &b , int max_iter , double epsilon , int N_x , int N_y);
33
34
35 class MatrixCoef
36 {
37     public:
38     vector <vector<double>> a_p , a_n , a_s , a_e , a_w , b_p;
39     MatrixCoef(unsigned int N_i , unsigned int N_j ):
40         a_p(N_i , vector<double>(N_j, 1.0) ) , a_n(N_i , vector<double>(N_j, 0.0) ) ,
41         a_s(N_i , vector<double>(N_j, 0.0) ) , a_e(N_i , vector<double>(N_j, 0.0) ) ,
42         a_w(N_i , vector<double>(N_j, 0.0) ) , b_p(N_i , vector<double>(N_j, 0.0) ) {};
43 };
44
45 class velocity_x
46 {
47     public:
48     vector <vector<double>> u_n , u_s , u_e , u_w ;
49     velocity_x (unsigned int N_i , unsigned int N_j) :
50         u_n (N_i , vector<double>(N_j, 0.0) ) ,

```

```

51     u_s (N_i , vector<double>(N_j, 0.0) ) ,
52     u_e (N_i , vector<double>(N_j, 0.0) ) ,
53     u_w (N_i , vector<double>(N_j, 0.0) ) {}};
54 };
55
56 class velocity_y
57 {
58     public:
59     vector <vector<double>> v_n , v_s , v_e , v_w ;
60     velocity_y (unsigned int N_i , unsigned int N_j) :
61         v_n (N_i , vector<double>(N_j, 0.0) ) ,
62         v_s (N_i , vector<double>(N_j, 0.0) ) ,
63         v_e (N_i , vector<double>(N_j, 0.0) ) ,
64         v_w (N_i , vector<double>(N_j, 0.0) ) {}};
65 };
66
67 class Massflow
68 {
69     public:
70     vector <vector<double>> F_n , F_s , F_e , F_w;
71     Massflow (unsigned int N_i , unsigned int N_j):
72         F_n (N_i , vector<double>(N_j, 0.0) ) ,
73         F_s (N_i , vector<double>(N_j, 0.0) ) ,
74         F_e (N_i , vector<double>(N_j, 0.0) ) ,
75         F_w (N_i , vector<double>(N_j, 0.0) ) {}};
76 };
77
78 class gradiente
79 {
80     public:
81     vector <vector<double>> grad_n , grad_s , grad_e , grad_w;
82     gradiente (unsigned int N_i , unsigned int N_j):
83         grad_n (N_i , vector<double>(N_j, 0.0) ) ,
84         grad_s (N_i , vector<double>(N_j, 0.0) ) ,
85         grad_e (N_i , vector<double>(N_j, 0.0) ) ,
86         grad_w (N_i , vector<double>(N_j, 0.0) ) {}};
87 };
88
89 class Diffusion
90 {
91     public:
92     vector <vector<double>> D_n , D_s , D_e , D_w;
93     Diffusion (unsigned int N_i , unsigned int N_j):
94         D_n (N_i , vector<double>(N_j, 0.0) ) ,
95         D_s (N_i , vector<double>(N_j, 0.0) ) ,
96         D_e (N_i , vector<double>(N_j, 0.0) ) ,
97         D_w (N_i , vector<double>(N_j, 0.0) ) {}};
98 };
99
100 class Peclet
101 {
102     public:
103     vector <vector<double>> Pe_n , Pe_s , Pe_e , Pe_w;
104     Peclet (unsigned int N_i , unsigned int N_j):
105         Pe_n (N_i , vector<double>(N_j, 0.0) ) ,
106         Pe_s (N_i , vector<double>(N_j, 0.0) ) ,
107         Pe_e (N_i , vector<double>(N_j, 0.0) ) ,
108         Pe_w (N_i , vector<double>(N_j, 0.0) ) {}};
109 };
110
111 class caseConv2D
112 {
113     private:
114     double L_x , L_y , Ax , Ay , e , Pr , Ra , gx , gy , angle ;
115     unsigned int N_i , N_j , max_iter ;
116     vector <vector<double>> v_0 , v_1 , v_2 ;
117     vector <vector<double>> u_0 , u_1 , u_2 ;
118     vector <vector<double>> u_p2 , v_p2 ;
119     vector <vector<double>> S_w , S_e , S_n , S_s , Vol ;
120     vector <vector<double>> x , y , x_w , y_w , x_e , y_e , x_n , y_n , x_s , y_s ;
121     vector <vector<double>> R_u0 , R_v0 , R_ul , R_vl ;
122     vector <vector<double>> mu , rho ;
123     velocity_x velx;

```

```

124     velocity_y vely;
125     Massflow mass_x;
126     Massflow mass_y;
127     gradiente gradient_u;
128     gradiente gradient_v;
129     vector <vector<double>> v_out , u_out ;
130     Diffusion diff_T ;
131     Massflow mass_T ;
132     Peclet pe_T ;
133
134
135
136     public:
137         double At , t_2 , epsilon_vT , epsilon_pT ;
138         double t_end ;
139         MatrixCoef coef;
140         vector <vector<double>> p_0 , p_1 , p_1sup ;
141         vector <vector<double>> T_0 , T_1 , T_1sup;
142         MatrixCoef coef_T ;
143         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);
144         void SetUniformMesh();
145         void SetNonUniformMesh();
146         void Initialvalues();
147         void Eval_boundary_velocity(enum R_u_BccValues bcc[4] , const vector<double> &value);
148         void interfacevelocity_u();
149         void interfacevelocity_v();
150         void massflow_x();
151         void massflow_y();
152         void grad_u();
153         void grad_v();
154         void EvalR_u();
155         void EvalR_v();
156         void EvalVelocity_p();
157         void EvalCoef();
158         void EvalCoefbc();
159         void Eval_coef_constantes_P();
160         void Eval_coef_variables_P();
161         void solveGS();
162         void solveTDMA();
163         void EvalVelocityfield();
164         void Eval_boundary_temperature(enum T_BccValues bcc[4] ,const vector<double> &value);
165         void Evaldiffusion_T();
166         void Evalmassflow_T();
167         void Evalpeclet_T();
168         void EvalCoef_T();
169         void Eval_boundary_CoeffbcT_temperature(enum T_BccValues bcc[4] , const vector<double>
            &value);
170         void solveTDMA_Temperature();
171         void solveGS_Temp();
172         double GetConvergence();
173         void NewStep();
174         void saveValues(double conv , unsigned int iter);
175         void EvalTime();
176         double CDSscheme_ul(int i , int j , int il , int jl);
177         double CDSscheme_vl(int i , int j , int il , int jl);
178         void Massconservation();
179         double Nu_value();
180     };
181
182     int main()
183     {
184
185         ifstream infile ("input.txt");
186         if (infile.fail()){
187             cout << "Error opening file" << "input.txt" << endl;
188             exit(0);
189         }
190
191         string s;
192         double Lx , Ly , A_x , A_y , E = 1.0 , Epsilon_vT , Epsilon_pT , Pr_vol , Ra_vol , Mu ,

```

```

    Rho ,t_End , convergence;
193 double Gx , Gy , Angle ;
194 unsigned int Nx , Ny , Max_iter , comp = 0 ;
195 vector<string> s_R_u_bcc(4) , s_T_bcc(4) ;
196 vector<double> v_R_u_bcc(8,0.0) , v_T_bcc(4,0.0) ;
197 unsigned int count = 1;
198 unsigned int i , j , k , l;
199
200 infile >> s >> Lx;
201 infile >> s >> Ly;
202 infile >> s >> Nx;
203 infile >> s >> Ny;
204
205 infile >> s >> s ;
206 if ( s.compare("NonUniform")==0)
207 {
208     infile >> Gx >> Gy ;
209 }
210
211 infile >> s >> Max_iter;
212 infile >> s >> Epsilon_vT;
213 infile >> s >> Epsilon_pT;
214 infile >> s >> t_End;
215 infile >> s >> Pr_vol;
216 infile >> s >> Ra_vol;
217 infile >> s >> Angle;
218
219 Mu = 1.0 ;
220 Rho = Mu / sqrt(9.81 * Lx * Lx * Lx * Pr_vol / Ra_vol ) ;
221
222 infile >> s >> s_R_u_bcc[0] ;
223     if(str2R_u_BccValues(s_R_u_bcc[0])==Airflow) infile >> v_R_u_bcc[0] >> v_R_u_bcc[1] ;
224 infile >> s >> s_R_u_bcc[1] ;
225     if(str2R_u_BccValues(s_R_u_bcc[1])==Airflow) infile >> v_R_u_bcc[2] >> v_R_u_bcc[3] ;
226 infile >> s >> s_R_u_bcc[2] ;
227     if(str2R_u_BccValues(s_R_u_bcc[2])==Airflow) infile >> v_R_u_bcc[4] >> v_R_u_bcc[5] ;
228 infile >> s >> s_R_u_bcc[3] ;
229     if(str2R_u_BccValues(s_R_u_bcc[3])==Airflow) infile >> v_R_u_bcc[6] >> v_R_u_bcc[7] ;
230
231 infile >> s >> s_T_bcc[0] ;
232     if(str2T_BccValues(s_T_bcc[0])==Dirichlet) infile >> v_T_bcc[0] ;
233 infile >> s >> s_T_bcc[1] ;
234     if(str2T_BccValues(s_T_bcc[1])==Dirichlet) infile >> v_T_bcc[1] ;
235 infile >> s >> s_T_bcc[2] ;
236     if(str2T_BccValues(s_T_bcc[2])==Dirichlet) infile >> v_T_bcc[2] ;
237 infile >> s >> s_T_bcc[3] ;
238     if(str2T_BccValues(s_T_bcc[3])==Dirichlet) infile >> v_T_bcc[3] ;
239
240 A_y = Ly / Ny;
241 A_x = Lx / Nx;
242
243 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]) };
244
245 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]) };
246
247 cout.setf( ios::floatfield , ios::scientific );
248 cout.precision(3);
249
250     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);
251     cas.SetNonUniformMesh();
252     cas.Initialvalues();
253
254 //first time step
255 //find predictor velocity field
256 cas.Eval_boundary_velocity(R_u_bcc , v_R_u_bcc);
257 cas.interfacevelocity_u();
258 cas.interfacevelocity_v();
259 cas.massflow_x();
260 cas.massflow_y();
261 cas.grad_u();

```

```

262 cas.grad_v();
263 cas.EvalR_u();
264 cas.EvalR_v();
265 cas.EvalVelocity_p();
266
267 //find pressure field
268 cas.Eval_coef_constants_P();
269 cas.Eval_coef_variables_P();
270 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);
271 cas.p_lsupsup = cas.p_1 ;
272
273 //find velocity field and check mass conservation
274 cas.EvalVelocityfield();
275 cas.Massconservation();
276
277 //Find temperature field
278 cas.Evaldiffusion_T();
279 cas.Evalmassflow_T();
280 cas.Evalpeplet_T();
281 cas.EvalCoef_T();
282 cas.Eval_boundary_CoeffbcT_temperature( T_bcc , v_T_bcc );
283 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_lsupsup , cas.coef_T.b_p , Max_iter ,
    Epsilon_pT, Nx , Ny);
284 cas.T_lsupsup = cas.T_1 ;
285
286 //new time step parameters
287 cas.EvalTime();
288 cas.NewStep();
289 cout << cas.t_2 << endl;
290
291 ofstream outfile ("conv.m");
292 if (outfile.fail()){
293     cerr << "Error creating conv.m file ." << endl;
294     exit(-1);
295 }
296
297 outfile.setf(ios::floatfield , ios::scientific);
298 outfile.precision(10);
299 outfile << "convs=[" ;
300 outfile << cas.t_2 << " " << cas.GetConvergence() << " " << cas.Nu_value() << endl;
301
302 while (cas.t_2 < cas.t_end && comp==0)
303 {
304     //Find predictor-velocity field
305     cas.Eval_boundary_velocity(R_u_bcc , v_R_u_bcc);
306     cas.interfacevelocity_u();
307     cas.interfacevelocity_v();
308     cas.massflow_x();
309     cas.massflow_y();
310     cas.grad_u();
311     cas.grad_v();
312     cas.EvalR_u();
313     cas.EvalR_v();
314     cas.EvalVelocity_p();
315
316     //Find pressure field
317     cas.Eval_coef_variables_P();
318     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);
319     cas.p_lsupsup = cas.p_1 ;
320
321     //Find velocity field
322     cas.EvalVelocityfield();
323     cas.Massconservation();
324
325     //Find temperature field
326     cas.Evalmassflow_T();
327     cas.Evalpeplet_T();
328     cas.EvalCoef_T();
329     cas.Eval_boundary_CoeffbcT_temperature( T_bcc , v_T_bcc );
330     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_lsup , cas.coef_T.b_p , Max_iter ,
        Epsilon_pT, Nx , Ny);
331 cas.T_lsup = cas.T_1 ;
332
333 convergence = cas.GetConvergence();
334 outfile << cas.t_2 << " " << convergence << " " << cas.Nu_value() << endl;
335 if( convergence < cas.epsilon_vT && cas.t_2 > 1.0 )
336 {
337     comp=1;
338 }
339 else
340 {
341     if ( count % 1000 == 0 )
342     {
343         cas.saveValues(convergence , count);
344     }
345     count++;
346
347     cas.EvalTime();
348     if (cas.t_2 < cas.t_end)
349     {
350         cas.NewStep();
351     }
352     cout << "time=" << cas.t_2 << " iter=" << count << " error=" << convergence << " Nu="
        << cas.Nu_value() << endl;
353 }
354 }
355
356 cas.saveValues(convergence , count);
357 outfile << "];" ;
358 outfile.close();
359
360 return 0;
361 }
362
363 caseConv2D::caseConv2D(double Lx , double Ly , double A_x , double A_y , double E ,
        double Epsilon_vT , double Epsilon_pT , double Pr_vol ,
364 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):
365 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_ul(N_i+2, vector<double>(N_j+1,0.0)) , R_vl(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_lsup(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_lsup(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) {});
366
367 void caseConv2D::SetUniformMesh()
368 {
369     unsigned int i , j;
370
371     for (i=1;i<=N_i;i++)
372     {

```



```

373     for (j=1;j<=N_j;j++)
374     {
375         x[i][j] = (j-0.5) * Ax ;
376         y[i][j] = (i-0.5) * Ay ;
377
378         x_w[i][j] = x[i][j] - 0.5 * Ax ;
379         x_e[i][j] = x[i][j] + 0.5 * Ax ;
380         x_n[i][j] = x[i][j] ;
381         x_s[i][j] = x[i][j] ;
382         y_w[i][j] = y[i][j] ;
383         y_e[i][j] = y[i][j] ;
384         y_n[i][j] = y[i][j] + 0.5 * Ay ;
385         y_s[i][j] = y[i][j] - 0.5 * Ay ;
386
387         S_w[i][j] = (y_n[i][j] - y_s[i][j]) * e ;
388         S_e[i][j] = (y_n[i][j] - y_s[i][j]) * e ;
389         S_n[i][j] = (x_e[i][j] - x_w[i][j]) * e ;
390         S_s[i][j] = (x_e[i][j] - x_w[i][j]) * e ;
391         Vol[i][j] = (y_n[i][j] - y_s[i][j]) * (x_e[i][j] - x_w[i][j]) * e ;
392     }
393 }
394
395 for (j=1;j<=N_j;j++)
396 {
397     x[0][j] = (j-0.5) * Ax ;
398     x[N_i+1][j] = (j-0.5) * Ax ;
399     y[0][j] = 0.0 ;
400     y[N_i+1][j] = L_y ;
401
402     x_w[0][j] = x[0][j] - 0.5 * Ax ;
403     y_w[0][j] = y[0][j] ;
404     x_e[0][j] = x[0][j] + 0.5 * Ax ;
405     y_e[0][j] = y[0][j] ;
406     x_n[0][j] = x[0][j] ;
407     y_n[0][j] = y[0][j] ;
408     x_s[0][j] = x[0][j] ;
409     y_s[0][j] = y[0][j] ;
410
411     x_w[N_i+1][j] = x[N_i+1][j] - 0.5 * Ax ;
412     y_w[N_i+1][j] = y[N_i+1][j] ;
413     x_e[N_i+1][j] = x[N_i+1][j] + 0.5 * Ax ;
414     y_e[N_i+1][j] = y[N_i+1][j] ;
415     x_n[N_i+1][j] = x[N_i+1][j] ;
416     y_n[N_i+1][j] = y[N_i+1][j] ;
417     x_s[N_i+1][j] = x[N_i+1][j] ;
418     y_s[N_i+1][j] = y[N_i+1][j] ;
419 }
420
421 for (i=1;i<=N_i;i++)
422 {
423     x[i][0] = 0.0 ;
424     x[i][N_j+1] = L_x ;
425     y[i][0] = (i-0.5) * Ay ;
426     y[i][N_j+1] = (i-0.5) * Ay ;
427
428     x_n[i][0] = x[i][0] ;
429     y_n[i][0] = y[i][0] + 0.5 * Ay ;
430     x_s[i][0] = x[i][0] ;
431     y_s[i][0] = y[i][0] - 0.5 * Ay ;
432     x_e[i][0] = x[i][0] ;
433     y_e[i][0] = y[i][0] ;
434     x_w[i][0] = x[i][0] ;
435     y_w[i][0] = y[i][0] ;
436
437
438     x_n[i][N_j+1] = x[i][N_j+1] ;
439     y_n[i][N_j+1] = y[i][N_j+1] + 0.5 * Ay ;
440     x_s[i][N_j+1] = x[i][N_j+1] ;
441     y_s[i][N_j+1] = y[i][N_j+1] - 0.5 * Ay ;
442     x_e[i][N_j+1] = x[i][N_j+1] ;
443     y_e[i][N_j+1] = y[i][N_j+1] ;
444     x_w[i][N_j+1] = x[i][N_j+1] ;
445     y_w[i][N_j+1] = y[i][N_j+1] ;

```

```

446     }
447     y[N_i+1][0] = L_y ;
448     y_n[N_i+1][0] = L_y ;
449     y_s[N_i+1][0] = L_y ;
450     y_e[N_i+1][0] = L_y ;
451     y_w[N_i+1][0] = L_y ;
452
453     x[0][N_j+1] = L_x ;
454     x_n[0][N_j+1] = L_x ;
455     x_s[0][N_j+1] = L_x ;
456     x_e[0][N_j+1] = L_x ;
457     x_w[0][N_j+1] = L_x ;
458
459     x[N_i+1][N_j+1] = L_x ;
460     y[N_i+1][N_j+1] = L_y ;
461     x_n[N_i+1][N_j+1] = L_x ;
462     y_n[N_i+1][N_j+1] = L_y ;
463     x_s[N_i+1][N_j+1] = L_x ;
464     y_s[N_i+1][N_j+1] = L_y ;
465     x_e[N_i+1][N_j+1] = L_x ;
466     y_e[N_i+1][N_j+1] = L_y ;
467     x_w[N_i+1][N_j+1] = L_x ;
468     y_w[N_i+1][N_j+1] = L_y ;
469 }
470
471 void caseConv2D::SetNonUniformMesh()
472 {
473
474     unsigned int i , j ;
475     unsigned int N_x = N_j , N_y = N_i ;
476
477     for (i=1;i<=N_i;i++)
478     {
479         for (j=1;j<=N_j;j++)
480         {
481
482             x[i][j] = (det_coord_Mesh ( L_x , N_x , gx , j ) + det_coord_Mesh ( L_x , N_x , gx ,
483                                     j+1 ))/2.0 ;
484             y[i][j] = (det_coord_Mesh ( L_y , N_y , gy , i ) + det_coord_Mesh ( L_y , N_y , gy ,
485                                     i+1 ))/2.0 ;
486
487             x_w[i][j] = det_coord_Mesh ( L_x , N_x , gx , j ) ;
488             x_e[i][j] = det_coord_Mesh ( L_x , N_x , gx , j+1 ) ;
489             x_n[i][j] = x[i][j] ;
490             x_s[i][j] = x[i][j] ;
491             y_w[i][j] = y[i][j] ;
492             y_e[i][j] = y[i][j] ;
493             y_n[i][j] = det_coord_Mesh ( L_y , N_y , gy , i+1 ) ;
494             y_s[i][j] = det_coord_Mesh ( L_y , N_y , gy , i ) ;
495
496             S_w[i][j] = (y_n[i][j] - y_s[i][j]) * e ;
497             S_e[i][j] = (y_n[i][j] - y_s[i][j]) * e ;
498             S_n[i][j] = (x_e[i][j] - x_w[i][j]) * e ;
499             S_s[i][j] = (x_e[i][j] - x_w[i][j]) * e ;
500             Vol[i][j] = (y_n[i][j] - y_s[i][j]) * (x_e[i][j] - x_w[i][j]) * e ;
501         }
502     }
503
504     for (j=1;j<=N_j;j++)
505     {
506         x[0][j] = (det_coord_Mesh ( L_x , N_x , gx , j ) + det_coord_Mesh ( L_x , N_x , gx , j
507                                     +1 ))/2.0 ;
508         x[N_i+1][j] = (det_coord_Mesh ( L_x , N_x , gx , j ) + det_coord_Mesh ( L_x , N_x , gx
509                                     , j+1 ))/2.0 ;
510         y[0][j] = 0.0 ;
511         y[N_i+1][j] = L_y ;
512
513         x_w[0][j] = det_coord_Mesh ( L_x , N_x , gx , j ) ;
514         y_w[0][j] = y[0][j] ;
515         x_e[0][j] = det_coord_Mesh ( L_x , N_x , gx , j+1 ) ;
516         y_e[0][j] = y[0][j] ;
517         x_n[0][j] = x[0][j] ;

```

```

515     y_n[0][j] = y[0][j] ;
516     x_s[0][j] = x[0][j] ;
517     y_s[0][j] = y[0][j] ;
518
519     x_w[N_i+1][j] = det_coord_Mesh ( L_x , N_x , gx , j ) ;
520     y_w[N_i+1][j] = y[N_i+1][j] ;
521     x_e[N_i+1][j] = det_coord_Mesh ( L_x , N_x , gx , j+1 ) ;
522     y_e[N_i+1][j] = y[N_i+1][j] ;
523     x_n[N_i+1][j] = x[N_i+1][j] ;
524     y_n[N_i+1][j] = y[N_i+1][j] ;
525     x_s[N_i+1][j] = x[N_i+1][j] ;
526     y_s[N_i+1][j] = y[N_i+1][j] ;
527 }
528
529 for ( i=1;i<=N_i;i++)
530 {
531     x[i][0] = 0.0 ;
532     x[i][N_j+1] = L_x ;
533     y[i][0] = (det_coord_Mesh ( L_y , N_y , gy , i ) + det_coord_Mesh ( L_y , N_y , gy , i
534         +1 ))/2.0 ;
535     y[i][N_j+1] = (det_coord_Mesh ( L_y , N_y , gy , i ) + det_coord_Mesh ( L_y , N_y , gy
536         , i+1 ))/2.0 ;
537
538     x_n[i][0] = x[i][0] ;
539     y_n[i][0] = det_coord_Mesh ( L_y , N_y , gy , i+1 ) ;
540     x_s[i][0] = x[i][0] ;
541     y_s[i][0] = det_coord_Mesh ( L_y , N_y , gy , i ) ;
542     x_e[i][0] = x[i][0] ;
543     y_e[i][0] = y[i][0] ;
544     x_w[i][0] = x[i][0] ;
545     y_w[i][0] = y[i][0] ;
546
547     x_n[i][N_j+1] = x[i][N_j+1] ;
548     y_n[i][N_j+1] = det_coord_Mesh ( L_y , N_y , gy , i+1 ) ;
549     x_s[i][N_j+1] = x[i][N_j+1] ;
550     y_s[i][N_j+1] = det_coord_Mesh ( L_y , N_y , gy , i ) ;
551     x_e[i][N_j+1] = x[i][N_j+1] ;
552     y_e[i][N_j+1] = y[i][N_j+1] ;
553     x_w[i][N_j+1] = x[i][N_j+1] ;
554     y_w[i][N_j+1] = y[i][N_j+1] ;
555 }
556     y[N_i+1][0] = L_y ;
557     y_n[N_i+1][0] = L_y ;
558     y_s[N_i+1][0] = L_y ;
559     y_e[N_i+1][0] = L_y ;
560     y_w[N_i+1][0] = L_y ;
561
562     x[0][N_j+1] = L_x ;
563     x_n[0][N_j+1] = L_x ;
564     x_s[0][N_j+1] = L_x ;
565     x_e[0][N_j+1] = L_x ;
566     x_w[0][N_j+1] = L_x ;
567
568     x[N_i+1][N_j+1] = L_x ;
569     y[N_i+1][N_j+1] = L_y ;
570     x_n[N_i+1][N_j+1] = L_x ;
571     y_n[N_i+1][N_j+1] = L_y ;
572     x_s[N_i+1][N_j+1] = L_x ;
573     y_s[N_i+1][N_j+1] = L_y ;
574     x_e[N_i+1][N_j+1] = L_x ;
575     y_e[N_i+1][N_j+1] = L_y ;
576     x_w[N_i+1][N_j+1] = L_x ;
577     y_w[N_i+1][N_j+1] = L_y ;
578
579 ofstream outfile("mesh.m");
580
581 if(outfile.fail())
582 {
583     cerr << "Error creating mesh.m file ." << endl;
584     exit(-1);
585 }

```

```

586     outfile << "x=[" << endl;
587     for (i=1;i<N_i+1;i++)
588     {
589         for (j=1;j<N_j+1;j++)
590         {
591             outfile << x[i][j] << " " ;
592         }
593         outfile << endl ;
594     }
595     outfile << "];" << endl << endl;
596
597     outfile << "y=[" << endl;
598     for (i=1;i<N_i+1;i++)
599     {
600         for (j=1;j<N_j+1;j++)
601         {
602             outfile << y[i][j] << " " ;
603         }
604         outfile << endl ;
605     }
606     outfile << "];" << endl << endl;
607
608     outfile.close();
609
610 }
611
612 void caseConv2D::Initialvalues ()
613 {
614     u_1 = u_0;
615     v_1 = v_0;
616     p_1sup = p_0;
617     T_1sup = T_0 ;
618
619     unsigned int i , j ;
620     double temp = 0.0 ;
621     for (i=1;i<N_i;i++)
622     {
623         for (j=1;j<N_j+1;j++)
624         {
625             temp = (y_n[i][j]-y[i][j]) / (y[i+1][j]-y[i][j]) ;
626             R_v0[i][j] = Pr * ( T_0[i][j] * (1.0-temp) + temp * T_0[i+1][j] ) ;
627         }
628     }
629 }
630
631 enum R_u_BccValues str2R_u_BccValues(string str)
632 {
633     if(str=="Wall")
634     {
635         return (Wall);
636     }
637     else if (str=="Airflow")
638     {
639         return (Airflow);
640     }
641     else
642     {
643         cerr << "Error in " << __FUNCTION__ << " : " << str << " condition doesn't exist." <<
        endl;
644         exit(0);
645     }
646 }
647
648 enum T_BccValues str2T_BccValues(string str)
649 {
650     if(str=="Neumann")
651     {
652         return (Neumann);
653     }
654     else if (str=="Dirichlet")
655     {
656         return (Dirichlet);
657     }

```

```

658     else
659     {
660         cerr << "Error in " << __FUNCTION__ << ": " << str << " condition doesn't exist." <<
            endl;
661         exit(0);
662     }
663 }
664
665 void caseConv2D::Eval_boundary_velocity(enum R_u_BccValues bcc[4] , const vector<double>
    &value)
666 {
667     unsigned int i , j ;
668
669     switch (bcc[0])
670     {
671         case Wall:
672             j=0;
673             for (i=0;i<N_i+2;i++)
674             {
675                 u_1[i][j] = 0.0;
676                 u_p2[i][j] = 0.0;
677             }
678             for (i=0;i<N_i+1;i++)
679             {
680                 v_1[i][j] = 0.0;
681                 v_p2[i][j] = 0.0;
682             }
683             break;
684         case Airflow:
685             j=0;
686             for (i=0;i<N_i+2;i++)
687             {
688                 u_1[i][j] = value[0];
689                 u_p2[i][j] = value[0];
690             }
691             for (i=0;i<N_i+1;i++)
692             {
693                 v_1[i][j] = value[1];
694                 v_p2[i][j] = value[1];
695             }
696             break;
697         default:
698             assert(false);
699             break;
700     }
701     switch (bcc[1]){
702         case Wall:
703             i=0;
704             for (j=0;j<N_j+1;j++)
705             {
706                 u_1[i][j] = 0.0;
707                 u_p2[i][j] = 0.0;
708             }
709             for (j=0;j<N_j+2;j++)
710             {
711                 v_1[i][j] = 0.0;
712                 v_p2[i][j] = 0.0;
713             }
714             break;
715         case Airflow:
716             i=0;
717             for (j=0;j<N_j+1;j++)
718             {
719                 u_1[i][j] = value[2];
720                 u_p2[i][j] = value[2];
721             }
722             for (j=0;j<N_j+2;j++)
723             {
724                 v_1[i][j] = value[3];
725                 v_p2[i][j] = value[3];
726             }
727             break;
728         default:

```

```

729     assert( false );
730     break;
731 }
732 switch (bcc[2]){
733     case Wall:
734         j=N_j;
735         for (i=0;i<N_i+2;i++)
736             {
737                 u_1[i][j] = 0.0;
738                 u_p2[i][j] = 0.0;
739             }
740         j=N_j+1;
741         for (i=0;i<N_i+1;i++)
742             {
743                 v_1[i][j] = 0.0;
744                 v_p2[i][j] = 0.0;
745             }
746         break;
747     case Airflow:
748         j=N_j;
749         for (i=0;i<N_i+2;i++)
750             {
751                 u_1[i][j] = value[4];
752                 u_p2[i][j] = value[4];
753             }
754         j=N_j+1;
755         for (i=0;i<N_i+1;i++)
756             {
757                 v_1[i][j] = value[5];
758                 v_p2[i][j] = value[5];
759             }
760         break;
761     default:
762         assert( false );
763         break;
764 }
765 switch (bcc[3]){
766     case Wall:
767         i=N_i+1;
768         for (j=0;j<N_j+1;j++)
769             {
770                 u_1[i][j] = 0.0;
771                 u_p2[i][j] = 0.0;
772             }
773         i=N_i;
774         for (j=0;j<N_j+2;j++)
775             {
776                 v_1[i][j] = 0.0;
777                 v_p2[i][j] = 0.0;
778             }
779         break;
780     case Airflow:
781         i=N_i+1;
782         for (j=0;j<N_j+1;j++)
783             {
784                 u_1[i][j] = value[6];
785                 u_p2[i][j] = value[6];
786             }
787         i=N_i;
788         for (j=0;j<N_j+2;j++)
789             {
790                 v_1[i][j] = value[7];
791                 v_p2[i][j] = value[7];
792             }
793         break;
794     default:
795         assert( false );
796         break;
797 }
798 }
799
800 double caseConv2D::CDSscheme_ul(int i , int j , int il , int jl)
801 {

```

```

802     double temp=0.0;
803
804     if (i==0 && i!=i1)
805     {
806         temp = 0.0 ;
807     }
808     else if(i1==N_i+1 && i!=i1)
809     {
810         temp = 1.0 ;
811     }
812     else if(i==i1 && j!=j1)
813     {
814         temp = (x[i1][j1]-x_e[i][j]) / (x_e[i1][j1]-x_e[i][j]) ;
815     }
816     else
817     {
818         temp = (y_n[i][j]-y[i][j]) / (y[i1][j1]-y[i][j]) ;
819     }
820
821     return u_1[i][j] * (1.0-temp) + temp * u_1[i1][j1];
822 }
823
824 double caseConv2D::CDSscheme_v1(int i , int j , int i1 , int j1)
825 {
826     double temp=0.0;
827
828     if (j==0 && j!=j1)
829     {
830         temp = 0.0 ;
831     }
832     else if(j1==N_j+1 && j!=j1)
833     {
834         temp = 1.0 ;
835     }
836     else if(j==j1 && i!=i1)
837     {
838         temp = (y[i1][j1]-y_n[i][j]) / (y_n[i1][j1]-y_n[i][j]) ;
839     }
840     else
841     {
842         temp = (x_e[i][j]-x[i][j]) / (x[i1][j1]-x[i][j]) ;
843     }
844
845     return v_1[i][j] * (1.0-temp) + temp * v_1[i1][j1];
846 }
847
848 void caseConv2D::interfacevelocity_u()
849 {
850     unsigned int i , j ;
851
852     for (i=1;i<N_i+1;i++)
853     {
854         for (j=1;j<N_j;j++)
855         {
856             velx.u_e[i][j] = CDSscheme_u1(i , j , i , j+1) ;
857             velx.u_w[i][j] = CDSscheme_u1(i , j -1 , i , j) ;
858             velx.u_n[i][j] = CDSscheme_u1(i , j , i+1 , j) ;
859             velx.u_s[i][j] = CDSscheme_u1(i -1 , j , i , j) ;
860         }
861     }
862 }
863
864 void caseConv2D::interfacevelocity_v()
865 {
866     unsigned int i , j ;
867
868     for (i=1;i<N_i;i++)
869     {
870         for (j=1;j<N_j+1;j++)
871         {
872             vely.v_e[i][j] = CDSscheme_v1(i , j , i , j+1) ;
873             vely.v_w[i][j] = CDSscheme_v1(i , j -1 , i , j) ;
874             vely.v_n[i][j] = CDSscheme_v1(i , j , i+1 , j) ;

```

```

875     vely.v_s[i][j] = CDSscheme_v1(i-1,j,i,j) ;
876 }
877 }
878 }
879
880 void caseConv2D::massflow_x()
881 {
882     unsigned int i , j;
883
884     for (i=1;i<N_i+1;i++)
885     {
886         for (j=1;j<N_j;j++)
887         {
888             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]) ;
889             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]) ;
890             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]
891                 - x_e[i][j] ) ) ;
892             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
893                 +1] - x_e[i][j] ) ) ;
894         }
895     }
896
897 void caseConv2D::massflow_y()
898 {
899     unsigned int i , j;
900
901     for (i=1;i<N_i+1;i++)
902     {
903         for (j=1;j<N_j+1;j++)
904         {
905             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]
906                 - y_n[i][j] ) ) ;
907             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
908                 +1][j] - y_n[i][j] ) ) ;
909             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] ) ;
910             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] ) ;
911         }
912     }
913
914 void caseConv2D::grad_u()
915 {
916     unsigned int i , j ;
917
918     for (i=1;i<N_i+1;i++)
919     {
920         for (j=1;j<N_j;j++)
921         {
922             gradient_u.grad_e[i][j] = (u_1[i][j+1] - u_1[i][j]) / (x_e[i][j+1] - x_e[i][j]) * (
923                 y_n[i][j] - y_s[i][j]) ;
924             gradient_u.grad_w[i][j] = (u_1[i][j] - u_1[i][j-1]) / (x_e[i][j] - x_e[i][j-1]) * (
925                 y_n[i][j] - y_s[i][j]) ;
926             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
927                 +1] - x[i][j]) ;
928             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
929                 +1] - x[i][j]) ;
930         }
931     }
932
933 void caseConv2D::grad_v()
934 {
935     unsigned int i , j ;
936
937     for (i=1;i<N_i+1;i++)
938     {
939         for (j=1;j<N_j+1;j++)
940         {
941             gradient_v.grad_e[i][j] = (v_1[i][j+1] - v_1[i][j]) / (x[i][j+1] - x[i][j]) * (y[i
942                 +1][j] - y[i][j]) ;
943             gradient_v.grad_w[i][j] = (v_1[i][j] - v_1[i][j-1]) / (x[i][j] - x[i][j-1]) * (y[i
944                 +1][j] - y[i][j]) ;

```



```

938     gradient_v.grad_n[i][j] = (v_1[i+1][j] - v_1[i][j]) / (y_n[i+1][j] - y_n[i][j]) * (
939         x_e[i][j] - x_w[i][j]) ;
940     gradient_v.grad_s[i][j] = (v_1[i][j] - v_1[i-1][j]) / (y_n[i][j] - y_n[i-1][j]) * (
941         x_e[i][j] - x_w[i][j]) ;
942 }
943 }
944 void caseConv2D::EvalR_u()
945 {
946     unsigned int i , j ;
947     double temp = 0.0 , temp1 = 0.0 , temp2 = 0.0 , temp3 = 0.0 ;
948
949     for (i=1;i<N_i+1;i++)
950     {
951         for (j=1;j<N_j;j++)
952         {
953             temp1 = gradient_u.grad_e[i][j] + gradient_u.grad_n[i][j] - gradient_u.grad_w[i][j] -
954                 gradient_u.grad_s[i][j] ;
955             temp2 = mass_x.F_e[i][j] * velx.u_e[i][j] + mass_x.F_n[i][j] * velx.u_n[i][j] -
956                 mass_x.F_w[i][j] * velx.u_w[i][j] - mass_x.F_s[i][j] * velx.u_s[i][j] ;
957             temp = (x_e[i][j]-x[i][j]) / (x[i][j+1]-x[i][j]) ;
958             temp3 = Pr * ( T_0[i][j] * (1.0-temp) + temp * T_0[i][j+1] ) * cos (angle*PI/180) ;
959             R_ul[i][j] = ( Pr*temp1/sqrt(Ra) - temp2 ) / ( (x[i][j+1]-x[i][j])*(y_n[i][j]-y_s[i][j]) ) + temp3 ;
960         }
961     }
962 }
963
964 void caseConv2D::EvalR_v()
965 {
966     unsigned int i , j ;
967     double temp = 0.0 , temp1 = 0.0 , temp2 = 0.0 , temp3 = 0.0 ;
968
969     for (i=1;i<N_i;i++)
970     {
971         for (j=1;j<N_j+1;j++)
972         {
973             temp1 = gradient_v.grad_e[i][j] + gradient_v.grad_n[i][j] - gradient_v.grad_w[i][j] -
974                 gradient_v.grad_s[i][j] ;
975             temp2 = mass_y.F_e[i][j] * vely.v_e[i][j] + mass_y.F_n[i][j] * vely.v_n[i][j] -
976                 mass_y.F_w[i][j] * vely.v_w[i][j] - mass_y.F_s[i][j] * vely.v_s[i][j] ;
977             temp = (y_n[i][j]-y[i][j]) / (y[i+1][j]-y[i][j]) ;
978             temp3 = Pr * ( T_0[i][j] * (1.0-temp) + temp * T_0[i+1][j] ) * sin (angle*PI/180) ;
979             R_vl[i][j] = ( Pr*temp1/sqrt(Ra) - temp2 ) / ( (x_e[i][j] - x_w[i][j])*(y[i+1][j]-y[i][j]) ) + temp3 ;
980         }
981     }
982 }
983
984 void caseConv2D::EvalVelocity_p()
985 {
986     unsigned int i , j ;
987
988     for (i=1;i<N_i+1;i++)
989     {
990         for (j=1;j<N_j;j++)
991         {
992             u_p2[i][j] = u_1[i][j] + At * ( 3.0/2.0 * R_ul[i][j] - 1.0/2.0 * R_u0[i][j] ) ;
993         }
994     }
995     for (i=1;i<N_i;i++)
996     {
997         for (j=1;j<N_j+1;j++)
998         {
999             v_p2[i][j] = v_1[i][j] + At * ( 3.0/2.0 * R_vl[i][j] - 1.0/2.0 * R_v0[i][j] ) ;
1000         }
1001     }
1002 }

```

```

1003
1004 void caseConv2D::EvalCoef()
1005 {
1006     unsigned int i , j;
1007
1008     for (i=2;i<N_i;i++)
1009     {
1010         for (j=2;j<N_j;j++)
1011         {
1012             coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
1013             coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
1014             coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
1015             coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
1016             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] ;
1017             coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1018                 +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1019         }
1020     }
1021 }
1022
1023 void caseConv2D::EvalCoefbc()
1024 {
1025     unsigned int i , j;
1026
1027     {
1028         i=1; j=1;
1029         coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
1030         coef.a_s[i][j] = 0.0 ;
1031         coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
1032         coef.a_w[i][j] = 0.0 ;
1033         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] ;
1034         coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1035             +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1036     }
1037
1038     {
1039         i=1; j=N_j;
1040         coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
1041         coef.a_s[i][j] = 0.0 ;
1042         coef.a_e[i][j] = 0.0 ;
1043         coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
1044         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] ;
1045         coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1046             +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1047     }
1048
1049     {
1050         i=N_i; j=1;
1051         coef.a_n[i][j] = 0.0 ;
1052         coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
1053         coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
1054         coef.a_w[i][j] = 0.0 ;
1055         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] ;
1056         coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1057             +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1058     }
1059
1060     {
1061         i=N_i; j=N_j;
1062         coef.a_n[i][j] = 0.0 ;
1063         coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
1064         coef.a_e[i][j] = 0.0 ;
1065         coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
1066         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] ;
1067         coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1068             +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1069     }
1070
1071     i=1;
1072     for (j=2;j<N_j;j++)
1073     {
1074         coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
1075         coef.a_s[i][j] = 0.0 ;

```

```

1076     coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
1077     coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
1078     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] ;
1079     coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1080                       +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1081 }
1082
1083 i=N_i;
1084 for (j=2;j<N_j;j++)
1085 {
1086     coef.a_n[i][j] = 0.0 ;
1087     coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
1088     coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
1089     coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
1090     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] ;
1091     coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1092                       +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1093 }
1094
1095 j=1;
1096 for (i=2;i<N_i;i++)
1097 {
1098     coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
1099     coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
1100     coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
1101     coef.a_w[i][j] = 0.0 ;
1102     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] ;
1103     coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1104                       +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1105 }
1106
1107 j=N_j;
1108 for (i=2;i<N_i;i++)
1109 {
1110     coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
1111     coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
1112     coef.a_e[i][j] = 0.0 ;
1113     coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
1114     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] ;
1115     coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1116                       +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1117 }
1118 }
1119
1120 void caseConv2D::Eval_coef_constantes_P()
1121 {
1122     unsigned int i , j;
1123
1124     for(i=2;i<N_i;i++)
1125     {
1126         for(j=2;j<N_j;j++)
1127         {
1128             coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
1129             coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
1130             coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
1131             coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
1132             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] ;
1133         }
1134     }
1135
1136     {
1137         i=1; j=1;
1138         coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
1139         coef.a_s[i][j] = 0.0 ;
1140         coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
1141         coef.a_w[i][j] = 0.0 ;
1142         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] ;
1143     }
1144
1145     {
1146         i=1; j=N_j;
1147         coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
1148         coef.a_s[i][j] = 0.0 ;

```

```

1149     coef.a_e[i][j] = 0.0 ;
1150     coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
1151     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] ;
1152 }
1153
1154 {
1155     i=N_i; j=1;
1156     coef.a_n[i][j] = 0.0 ;
1157     coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
1158     coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
1159     coef.a_w[i][j] = 0.0 ;
1160     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] ;
1161 }
1162
1163 {
1164     i=N_i; j=N_j;
1165     coef.a_n[i][j] = 0.0 ;
1166     coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
1167     coef.a_e[i][j] = 0.0 ;
1168     coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
1169     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] ;
1170 }
1171
1172 i=1;
1173 for (j=2;j<N_j;j++)
1174 {
1175     coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
1176     coef.a_s[i][j] = 0.0 ;
1177     coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
1178     coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
1179     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] ;
1180 }
1181
1182 i=N_i;
1183 for (j=2;j<N_j;j++)
1184 {
1185     coef.a_n[i][j] = 0.0 ;
1186     coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
1187     coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
1188     coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
1189     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] ;
1190 }
1191
1192 j=1;
1193 for (i=2;i<N_i;i++)
1194 {
1195     coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
1196     coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
1197     coef.a_e[i][j] = S_e[i][j] / (x[i][j+1] - x[i][j]) ;
1198     coef.a_w[i][j] = 0.0 ;
1199     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] ;
1200 }
1201
1202 j=N_j;
1203 for (i=2;i<N_i;i++)
1204 {
1205     coef.a_n[i][j] = S_n[i][j] / (y[i+1][j] - y[i][j]) ;
1206     coef.a_s[i][j] = S_s[i][j] / (y[i][j] - y[i-1][j]) ;
1207     coef.a_e[i][j] = 0.0 ;
1208     coef.a_w[i][j] = S_w[i][j] / (x[i][j] - x[i][j-1]) ;
1209     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] ;
1210 }
1211 }
1212
1213 void caseConv2D::Eval_coef_variables_P()
1214 {
1215     unsigned int i , j;
1216
1217     for (i=2;i<N_i;i++)
1218     {
1219         for (j=2;j<N_j;j++)
1220         {
1221             coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]

```

```

1222         +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1223     }
1224 }
1225
1226 {
1227     i=1; j=1;
1228     coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1229         +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1230 }
1231
1232 {
1233     i=1; j=N_j;
1234     coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1235         +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1236 }
1237
1238 {
1239     i=N_i; j=1;
1240     coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1241         +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1242 }
1243
1244 {
1245     i=N_i; j=N_j;
1246     coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1247         +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1248 }
1249
1250 i=1;
1251 for (j=2;j<N_j;j++)
1252 {
1253     coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1254         +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1255 }
1256
1257 i=N_i;
1258 for (j=2;j<N_j;j++)
1259 {
1260     coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1261         +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1262 }
1263
1264 j=1;
1265 for (i=2;i<N_i;i++)
1266 {
1267     coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1268         +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1269 }
1270
1271 j=N_j;
1272 for (i=2;i<N_i;i++)
1273 {
1274     coef.b_p[i][j] = - ( v_p2[i][j] * S_n[i][j] - v_p2[i-1][j] * S_s[i][j]
1275         +u_p2[i][j] * S_e[i][j] - u_p2[i][j-1] * S_w[i][j] ) ;
1276 }
1277 }
1278
1279 void caseConv2D::solveGS()
1280 {
1281     unsigned int i , j;
1282     unsigned int iter;
1283     double deltap_ = epsilon_pT + 1.0;
1284
1285     for(iter=1; deltap_ > epsilon_pT && iter<max_iter ; iter++)
1286     {
1287         for(i=1;i<N_i+1;i++)
1288         {
1289             for(j=1;j<N_j+1;j++)
1290             {
1291                 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]
1292                     * 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]
1293                     + coef.b_p[i][j] );
1294             }
1295         }
1296     }

```

```

1293     }
1294     deltap_ = delta_inc( p_1 , p_1sup );
1295     p_1sup = p_1 ;
1296 }
1297 if (iter >= max_iter) cerr << "Warning: max number of iterations achieved: " << deltap_ <<
    endl;
1298 assert (deltap_ < epsilon_pT);
1299 }
1300
1301 void caseConv2D::solveTDMA()
1302 {
1303     unsigned int i , j;
1304     unsigned int iter;
1305     double deltap_ = epsilon_pT + 1.0;
1306     vector<double> P_x(N_j+1,0.0) , R_x(N_j+1,0.0) , b_x(N_j+1,0.0);
1307     vector<double> P_y(N_i+1,0.0) , R_y(N_i+1,0.0) , b_y(N_i+1,0.0);
1308
1309     for (iter=1; deltap_ > epsilon_pT && iter < max_iter ; iter++)
1310     {
1311
1312         for (i=1; i < N_i+1; i++)
1313         {
1314             for (j=1; j < N_j+1; j++)
1315             {
1316                 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] ;
1317             }
1318
1319             j=1;
1320             P_x[j] = coef.a_e[i][j] / coef.a_p[i][j];
1321             R_x[j] = b_x[j] / coef.a_p[i][j];
1322
1323             for (j=2; j < N_j+1; j++)
1324             {
1325                 P_x[j] = coef.a_e[i][j] / (coef.a_p[i][j] - coef.a_w[i][j]*P_x[j-1]);
1326                 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]);
1327             }
1328
1329             for (j=N_j; j >= 1; j--)
1330             {
1331                 p_1[i][j] = P_x[j]*p_1[i][j+1] + R_x[j];
1332             }
1333         }
1334
1335     p_1sup = p_1 ;
1336
1337     for (j=1; j < N_j+1; j++)
1338     {
1339         for (i=1; i < N_i+1; i++)
1340         {
1341             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] ;
1342         }
1343
1344         i=1;
1345         P_y[i] = coef.a_n[i][j] / coef.a_p[i][j];
1346         R_y[i] = b_y[i] / coef.a_p[i][j];
1347
1348         for (i=2; i < N_i+1; i++)
1349         {
1350             P_y[i] = coef.a_n[i][j] / (coef.a_p[i][j] - coef.a_s[i][j]*P_y[i-1]);
1351             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]);
1352         }
1353
1354         for (i=N_i; i >= 1; i--)
1355         {
1356             p_1[i][j] = P_y[i]*p_1[i+1][j] + R_y[i];
1357         }
1358     }
1359
1360     deltap_ = delta_inc( p_1 , p_1sup );

```

```

1361     p_1sup = p_1 ;
1362 }
1363 if (iter >= max_iter) cerr << "Warning: max number of iterations achieved: " << deltap_ <<
    endl;
1364 assert (deltap_ < epsilon_pT);
1365
1366 }
1367
1368 void caseConv2D::Evaldiffusion_T()
1369 {
1370     unsigned int i , j ;
1371
1372     for (i=1; i<N_i+1; i++)
1373     {
1374         for (j=1; j<N_j+1; j++)
1375         {
1376             diff_T.D_e[i][j] = S_e[i][j] / ( ( x[i][j+1]-x[i][j] ) * ( sqrt(Ra) ) ) ;
1377             diff_T.D_w[i][j] = S_w[i][j] / ( ( x[i][j]-x[i][j-1] ) * ( sqrt(Ra) ) ) ;
1378             diff_T.D_n[i][j] = S_n[i][j] / ( ( y[i+1][j]-y[i][j] ) * ( sqrt(Ra) ) ) ;
1379             diff_T.D_s[i][j] = S_s[i][j] / ( ( y[i][j]-y[i-1][j] ) * ( sqrt(Ra) ) ) ;
1380         }
1381     }
1382 }
1383
1384 void caseConv2D::Evalmassflow_T()
1385 {
1386     unsigned int i , j ;
1387
1388     for (i=1; i<N_i+1; i++)
1389     {
1390         for (j=1; j<N_j+1; j++)
1391         {
1392             mass_T.F_e[i][j] = u_2[i][j] * S_e[i][j] ;
1393             mass_T.F_w[i][j] = u_2[i][j-1] * S_w[i][j] ;
1394             mass_T.F_n[i][j] = v_2[i][j] * S_n[i][j] ;
1395             mass_T.F_s[i][j] = v_2[i-1][j] * S_s[i][j] ;
1396         }
1397     }
1398 }
1399
1400 void caseConv2D::Evalpeplet_T()
1401 {
1402     unsigned int i , j ;
1403
1404     for (i=1; i<N_i+1; i++)
1405     {
1406         for (j=1; j<N_j+1; j++)
1407         {
1408             pe_T.Pe_e[i][j] = mass_T.F_e[i][j] / diff_T.D_e[i][j] ;
1409             pe_T.Pe_w[i][j] = mass_T.F_w[i][j] / diff_T.D_w[i][j] ;
1410             pe_T.Pe_n[i][j] = mass_T.F_n[i][j] / diff_T.D_n[i][j] ;
1411             pe_T.Pe_s[i][j] = mass_T.F_s[i][j] / diff_T.D_s[i][j] ;
1412         }
1413     }
1414 }
1415
1416 void caseConv2D::EvalCoef_T()
1417 {
1418     unsigned int i , j ;
1419
1420     for (i=2; i<N_i; i++)
1421     {
1422         for (j=2; j<N_j; j++)
1423         {
1424             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[
1425                 i][j])) ? 0.0 : (-mass_T.F_n[i][j])) ;
1426             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[
1427                 i][j])) ? 0.0 : (mass_T.F_s[i][j])) ;
1428             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[
1429                 i][j])) ? 0.0 : (-mass_T.F_e[i][j])) ;
1430             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[
1431                 i][j])) ? 0.0 : (mass_T.F_w[i][j])) ;
1432             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.

```

```

1429         a_w[i][j] + Vol[i][j] / At ;
1430     coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
1431 }
1432 }
1433
1434 void caseConv2D::Eval_boundary_CoeffbcT_temperature(enum T_BccValues bcc[4] , const vector
1435 <double> &value)
1436 {
1437     unsigned int i , j ;
1438     switch (bcc[0]) //left
1439     {
1440     case Neumann:
1441         j=0;
1442         for (i=0;i<N_i+2;i++)
1443         {
1444             T_1sup[i][j] = 0.0;
1445             T_1[i][j] = 0.0;
1446         }
1447         {
1448             i=1; j=1; coef_T.a_w[i][j] = 0.0 ;
1449         }
1450         j=1;
1451         for (i=2;i<N_i;i++)
1452         {
1453             coef_T.a_w[i][j] = 0.0 ;
1454         }
1455         {
1456             i=N_i; j=1; coef_T.a_w[i][j] = 0.0 ;
1457         }
1458     break;
1459     case Dirichlet:
1460         j=0;
1461         for (i=0;i<N_i+2;i++)
1462         {
1463             T_1sup[i][j] = value[0];
1464             T_1[i][j] = value[0];
1465         }
1466         {
1467             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>(
1468                 mass_T.F_w[i][j]))?0.0:(mass_T.F_w[i][j])) ;
1469         }
1470         j=1;
1471         for (i=2;i<N_i;i++)
1472         {
1473             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
1474                 [i][j]))?0.0:(mass_T.F_w[i][j])) ;
1475         }
1476         {
1477             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]) +
1478                 ((0.0>(mass_T.F_w[i][j]))?0.0:(mass_T.F_w[i][j])) ;
1479         }
1480     break;
1481     default:
1482         assert(false);
1483     break;
1484 }
1485 switch (bcc[1])//bottom
1486 {
1487     case Neumann:
1488         i=0;
1489         for (j=0;j<N_j+2;j++)
1490         {
1491             T_1sup[i][j] = 0.0;
1492             T_1[i][j] = 0.0;
1493         }
1494         {
1495             i=1; j=1; coef_T.a_s[i][j] = 0.0 ;
1496         }
1497     }

```



```

1497     i=1; j=N_j; coef_T.a_s[i][j] = 0.0 ;
1498 }
1499 i=1;
1500 for (j=2;j<N_j;j++)
1501 {
1502     coef_T.a_s[i][j] = 0.0 ;
1503 }
1504
1505 break;
1506 case Dirichlet:
1507     i=0;
1508     for (j=0;j<N_j+2;j++)
1509     {
1510         T_1sup[i][j] = value[1];
1511         T_1[i][j] = value[1];
1512     }
1513     {
1514         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>(
1515             mass_T.F_s[i][j]))?0.0:(mass_T.F_s[i][j])) ;
1516     }
1517     {
1518         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]) +
1519             ((0.0>(mass_T.F_s[i][j]))?0.0:(mass_T.F_s[i][j])) ;
1520     }
1521     i=1;
1522     for (j=2;j<N_j;j++)
1523     {
1524         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
1525             [i][j]))?0.0:(mass_T.F_s[i][j])) ;
1526     }
1527     break;
1528 default:
1529     assert(false);
1530     break;
1531 }
1532 switch (bcc[2])//right
1533 {
1534 case Neumann:
1535     j=N_j+1;
1536     for (i=0;i<N_i+2;i++)
1537     {
1538         T_1sup[i][j] = 0.0;
1539         T_1[i][j] = 0.0;
1540     }
1541     {
1542         i=N_i; j=N_j; coef_T.a_e[i][j] = 0.0 ;
1543     }
1544     j=N_j;
1545     for (i=2;i<N_i;i++)
1546     {
1547         coef_T.a_e[i][j] = 0.0 ;
1548     }
1549     {
1550         i=1; j=N_j; coef_T.a_e[i][j] = 0.0 ;
1551     }
1552     break;
1553 case Dirichlet:
1554     j=N_j+1;
1555     for (i=0;i<N_i+2;i++)
1556     {
1557         T_1sup[i][j] = value[2];
1558         T_1[i][j] = value[2];
1559     }
1560     {
1561         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]) +
1562             ((0.0>(-mass_T.F_e[i][j]))?0.0:(-mass_T.F_e[i][j])) ;
1563     }
1564     j=N_j;
1565     for (i=2;i<N_i;i++)
1566     {
1567         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])) ;
1566     }
1567     {
1568         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])) ;
1569     }
1570
1571     break;
1572 default:
1573     assert(false);
1574     break;
1575 }
1576 switch (bcc[3])//top
1577 {
1578     case Neumann:
1579         i=N_i+1;
1580         for (j=0;j<N_j+2;j++)
1581         {
1582             T_1sup[i][j] = 0.0;
1583             T_1[i][j] = 0.0;
1584         }
1585         {
1586             i=N_i; j=1; coef_T.a_n[i][j] = 0.0 ;
1587         }
1588         i=N_i;
1589         for (j=2;j<N_j;j++)
1590         {
1591             coef_T.a_n[i][j] = 0.0 ;
1592         }
1593         {
1594             i=N_i; j=N_j; coef_T.a_n[i][j] = 0.0 ;
1595         }
1596
1597         break;
1598     case Dirichlet:
1599         i=N_i+1;
1600         for (j=0;j<N_j+2;j++)
1601         {
1602             T_1sup[i][j] = value[3];
1603             T_1[i][j] = value[3];
1604         }
1605         {
1606             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])) ;
1607         }
1608         i=N_i;
1609         for (j=2;j<N_j;j++)
1610         {
1611             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])) ;
1612         }
1613         {
1614             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])) ;
1615         }
1616
1617         break;
1618     default:
1619         assert(false);
1620         break;
1621 }
1622
1623 {
1624     i=1; j=1;
1625     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])) ;
1626     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])) ;
1627     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 ;
1628     coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
1629 }
1630

```

```

1631     j=1;
1632     for (i=2;i<N_i;i++)
1633     {
1634         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
1635         ][j]))?0.0:(-mass_T.F_n[i][j]));
1636         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][
1637         j]))?0.0:(mass_T.F_s[i][j]));
1638         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
1639         ][j]))?0.0:(-mass_T.F_e[i][j]));
1640         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[
1641         i][j] + Vol[i][j] / At ;
1642         coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
1643     }
1644
1645     {
1646         i=N_i; j=1;
1647         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][
1648         j]))?0.0:(mass_T.F_s[i][j]));
1649         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
1650         ][j]))?0.0:(-mass_T.F_e[i][j]));
1651         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[
1652         i][j] + Vol[i][j] / At ;
1653         coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
1654     }
1655
1656     i=N_i;
1657     for (j=2;j<N_j;j++)
1658     {
1659         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][
1660         j]))?0.0:(mass_T.F_s[i][j]));
1661         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
1662         ][j]))?0.0:(-mass_T.F_e[i][j]));
1663         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][
1664         j]))?0.0:(mass_T.F_w[i][j]));
1665         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[
1666         i][j] + Vol[i][j] / At ;
1667         coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
1668     }
1669
1670     {
1671         i=N_i; j=N_j;
1672         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][
1673         j]))?0.0:(mass_T.F_s[i][j]));
1674         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][
1675         j]))?0.0:(mass_T.F_w[i][j]));
1676         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[
1677         i][j] + Vol[i][j] / At ;
1678         coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
1679     }
1680
1681     j=N_j;
1682     for (i=2;i<N_i;i++)
1683     {
1684         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
1685         ][j]))?0.0:(-mass_T.F_n[i][j]));
1686         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][
1687         j]))?0.0:(mass_T.F_s[i][j]));
1688         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][
1689         j]))?0.0:(mass_T.F_w[i][j]));
1690         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[
1691         i][j] + Vol[i][j] / At ;
1692         coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
1693     }
1694
1695     {
1696         i=1; j=N_j;
1697         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
1698         ][j]))?0.0:(-mass_T.F_n[i][j]));
1699         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][
1700         j]))?0.0:(mass_T.F_w[i][j]));
1701         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[
1702         i][j] + Vol[i][j] / At ;
1703         coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
1704     }

```

```

1683     }
1684
1685     i=1;
1686     for (j=2;j<N_j;j++)
1687     {
1688         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[
1689             ])[j]))?0.0:(-mass_T.F_n[i][j]));
1690         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[
1691             ])[j]))?0.0:(-mass_T.F_e[i][j]));
1692         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][
1693             ])[j]))?0.0:(mass_T.F_w[i][j]));
1694         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[
1695             i][j] + Vol[i][j] / At ;
1696         coef_T.b_p[i][j] = T_0[i][j] * Vol[i][j] / At ;
1697     }
1698 }
1699
1700 void caseConv2D::solveTDMA_Temperature()
1701 {
1702     unsigned int i , j;
1703     unsigned int iter;
1704     double deltaT = epsilon_pT + 1.0;
1705     vector<double> P_x(N_j+1,0.0) , R_x(N_j+1,0.0) , b_x(N_j+1,0.0);
1706     vector<double> P_y(N_i+1,0.0) , R_y(N_i+1,0.0) , b_y(N_i+1,0.0);
1707
1708     for(iter=1; deltaT>epsilon_pT && iter<max_iter ; iter++)
1709     {
1710         for(i=1;i<N_i+1;i++)
1711         {
1712             for(j=1;j<N_j+1;j++)
1713             {
1714                 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] *
1715                     T_1sup[i-1][j] ;
1716             }
1717             j=1;
1718             P_x[j] = coef_T.a_e[i][j] / coef_T.a_p[i][j];
1719             R_x[j] = b_x[j] / coef_T.a_p[i][j];
1720
1721             for (j=2;j<N_j+1;j++)
1722             {
1723                 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]);
1724                 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]
1725                     ]*P_x[j-1]);
1726             }
1727
1728             for (j=N_j;j>=1;j--)
1729             {
1730                 T_1[i][j] = P_x[j] * T_1[i][j+1] + R_x[j];
1731             }
1732
1733             T_1sup = T_1 ;
1734
1735             for(j=1;j<N_j+1;j++)
1736             {
1737                 for(i=1;i<N_i+1;i++)
1738                 {
1739                     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] *
1740                         T_1sup[i][j-1] ;
1741                 }
1742                 i=1;
1743                 P_y[i] = coef_T.a_n[i][j] / coef_T.a_p[i][j];
1744                 R_y[i] = b_y[i] / coef_T.a_p[i][j];
1745
1746                 for (i=2;i<N_i+1;i++)
1747                 {
1748                     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]);
1749                     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]
1750                         ]*P_y[i-1]);
1751                 }
1752             }
1753         }
1754     }

```

```

1748
1749     for (i=N_i; i>=1; i--)
1750     {
1751         T_1[i][j] = P_y[i] * T_1[i+1][j] + R_y[i];
1752     }
1753 }
1754
1755 deltaT = delta_inc( T_1 , T_1sup );
1756 T_1sup = T_1 ;
1757 }
1758 if (iter>=max_iter) cerr << "Warning: max number of iterations achieved:" << deltaT <<
    endl;
1759 assert (deltaT < epsilon_pT);
1760 }
1761
1762 void caseConv2D::solveGS_Temp()
1763 {
1764     unsigned int i , j;
1765     unsigned int iter;
1766     double deltaT = epsilon_pT + 1.0;
1767
1768     for (iter=1; deltaT > epsilon_pT && iter<max_iter ; iter++)
1769     {
1770         for (i=1; i<N_i+1; i++)
1771         {
1772             for (j=1; j<N_j+1; j++)
1773             {
1774                 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]
1775                     + 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] );
1776             }
1777         }
1778         deltaT = delta_inc( T_1 , T_1sup );
1779         T_1sup = T_1 ;
1780     }
1781     if (iter>=max_iter) cerr << "Warning: max number of iterations achieved:" << deltaT <<
        endl;
1782     assert (deltaT < epsilon_pT);
1783 }
1784
1785 void caseConv2D::EvalTime()
1786 {
1787     double v_condx_max=0.0 , v_condy_max=0.0 , At_conv=0.0;
1788     double mu_condx_max=0.0 , mu_condy_max=0.0 , At_visc=0.0;
1789
1790     unsigned int i , j;
1791
1792     for (i=1; i<N_i+1; i++)
1793     {
1794         for (j=1; j<N_j; j++)
1795         {
1796             v_condx_max = (v_condx_max > fabs(u_1[i][j]/(x[i][j+1] - x[i][j])) )
1797             ? v_condx_max : fabs(u_1[i][j]/(x[i][j+1] - x[i][j])) ;
1798         }
1799     }
1800
1801     for (i=1; i<N_i; i++)
1802     {
1803         for (j=1; j<N_j+1; j++)
1804         {
1805             v_condy_max = (v_condy_max > fabs(v_1[i][j]/(y[i+1][j] - y[i][j])) )
1806             ? v_condy_max : fabs(v_1[i][j]/(y[i+1][j] - y[i][j])) ;
1807         }
1808     }
1809
1810     for (i=1; i<N_i+1; i++)
1811     {
1812         for (j=1; j<N_j+1; j++)
1813         {
1814             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])) )
1815             ? 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]

```

```

        j));
1816
1817     mu_condy_max = (mu_condy_max > fabs(mu[i][j]/rho[i][j]/(y_n[i][j] - y_s[i][j]))/(y_n[
1818     ? 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][
1819     }
1820   }
1821
1822   At_conv = (v_condx_max>v_condy_max) ? 0.35/v_condx_max : 0.35/v_condy_max;
1823   At_visc = (mu_condx_max>mu_condy_max) ? 0.2/mu_condx_max : 0.2/mu_condy_max;
1824
1825   At = (At_conv<At_visc) ? At_conv : At_visc ;
1826
1827   t_2 += At;
1828 }
1829
1830 void caseConv2D::EvalVelocityfield()
1831 {
1832   unsigned int i , j;
1833
1834   for (i=1;i<N_i+1;i++)
1835   {
1836     for (j=1;j<N_j;j++)
1837     {
1838
1839       u_2[i][j] = u_p2[i][j] - ( p_1[i][j+1] - p_1[i][j] )/(x[i][j+1] - x[i][j]);
1840     }
1841   }
1842
1843   for (i=1;i<N_i;i++)
1844   {
1845     for (j=1;j<N_j+1;j++)
1846     {
1847       v_2[i][j] = v_p2[i][j] - ( p_1[i+1][j] - p_1[i][j] )/(y[i+1][j] - y[i][j]);
1848     }
1849   }
1850 }
1851
1852 void caseConv2D::Massconservation()
1853 {
1854   unsigned int i , j;
1855
1856   for (i=1;i<N_i+1;i++)
1857   {
1858     for (j=1;j<N_j+1;j++)
1859     {
1860       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]
1861       - v_2[i-1][j] * S_s[i][j])/Vol[i][j] < 1E-1);
1862     }
1863   }
1864
1865 double caseConv2D::GetConvergence()
1866 {
1867   unsigned int i , j;
1868   double maximum=0.0;
1869
1870   maximum = delta_inc ( u_1 , u_2 );
1871   maximum = (maximum > delta_inc ( v_1 , v_2 ) ) ? maximum : delta_inc ( v_1 , v_2 ) ;
1872   maximum = (maximum > delta_inc ( T_0 , T_1 ) ) ? maximum : delta_inc ( T_0 , T_1 ) ;
1873
1874   return maximum/At;
1875 }
1876
1877 void caseConv2D::NewStep()
1878 {
1879
1880   u_0 = u_1 ;
1881   v_0 = v_1 ;
1882
1883   u_1 = u_2 ;
1884   v_1 = v_2 ;

```

```

1885
1886     p_0 = p_1 ;
1887
1888     R_u0 = R_u1 ;
1889     R_v0 = R_v1 ;
1890
1891     T_0 = T_1 ;
1892 }
1893
1894 void caseConv2D::saveValues(double conv , unsigned int iter)
1895 {
1896     unsigned int i,j;
1897     string outputname;
1898     string temp;
1899
1900     stringstream ss(stringstream::in | stringstream::out);
1901     ss.setf(ios::fixed);
1902     ss.precision(0);
1903     ss << iter ;
1904     ss >> temp ;
1905
1906     outputname = "output_" + temp + ".m";
1907     ofstream outfile(outputname.c_str());
1908
1909     if(outfile.fail())
1910     {
1911         cerr << "Error creating output.m file ." << endl;
1912         exit(-1);
1913     }
1914
1915     outfile.setf(ios::floatfield , ios::scientific);
1916     outfile.precision(10);
1917
1918     for (i=1;i<N_i+1;i++)
1919     {
1920         for (j=1;j<N_j+1;j++)
1921         {
1922             u_out[i][j] = (u_2[i][j]+u_2[i][j-1])/2 * sqrt(Ra);
1923         }
1924     }
1925
1926     for (i=1;i<N_i+1;i++)
1927     {
1928         for (j=1;j<N_j+1;j++)
1929         {
1930             v_out[i][j] = (v_2[i][j]+v_2[i-1][j])/2 * sqrt(Ra);
1931         }
1932     }
1933
1934     double Nu_average = 0.0;
1935     j = 0;
1936     for (i=1;i<N_i+1;i++)
1937     {
1938         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 ;
1939     }
1940
1941     //Nusselt number expected
1942     double Nu_expected = 0.0 , Nu_60 = 0.0 , Nu_90 = 0.0 ;
1943
1944     if (angle <= 60.0)
1945     {
1946         Nu_expected = 1.0 + 1.44 * max(0.0, 1.0 - 1708.0/(Ra * cos (angle * PI /180))) * (1.0
1947             - 1708.0 * pow(sin(1.8*angle*PI/180.0),1.6)/(Ra * cos (angle * PI /180.0)))
1948             + max(0.0,pow((Ra * cos(angle*PI/180.0) / 5830.0 ),1.0/3.0) -1.0);
1949     }
1950     else
1951     {
1952         Nu_60 = 1.0 + 1.44 * max(0.0, 1.0 - 1708.0/(Ra * cos (60.0 * PI /180))) * (1.0 -
1953             1708.0 * pow(sin(1.8*60*PI/180.0),1.6)/(Ra * cos (60.0 * PI /180.0)))
1954             + max(0.0,pow((Ra * cos(60.0*PI/180.0) / 5830.0 ),1.0/3.0) -1.0);
1955
1956         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 )
1957             );

```

```

1955
1956     Nu_expected = (90.0 - angle)/30.0 * Nu_60 + (angle - 60.0)/30.0 * Nu_90;
1957 }
1958
1959 outfile << "%Time:_" << t_2 << "_s" << endl;
1960 outfile << "%Derivative_value:_" << conv << endl << endl;
1961 outfile << "%Nusselt_number_at_x_0.0:_" << Nu_average << endl;
1962 outfile << "%Nusselt_number_expected" << Nu_expected << endl;
1963
1964 outfile << "u=[" << endl;
1965 for (i=1;i<N_i+1;i++)
1966 {
1967     for (j=1;j<N_j+1;j++)
1968     {
1969         outfile << u_out[i][j] << " " ;
1970     }
1971     outfile << endl ;
1972 }
1973 outfile << "];" << endl;
1974
1975 outfile << endl << endl;
1976 outfile << "v=[" << endl;
1977 for (i=1;i<N_i+1;i++)
1978 {
1979     for (j=1;j<N_j+1;j++)
1980     {
1981         outfile << v_out[i][j] << " " ;
1982     }
1983     outfile << endl ;
1984 }
1985 outfile << "];" << endl;
1986
1987 outfile << endl << endl;
1988 outfile << "p=[" << endl;
1989 for (i=1;i<N_i+1;i++)
1990 {
1991     for (j=1;j<N_j+1;j++)
1992     {
1993         outfile << p_1[i][j] / At << " " ;
1994     }
1995     outfile << endl ;
1996 }
1997 outfile << "];" << endl;
1998
1999 outfile << endl << endl;
2000 outfile << "T=[" << endl;
2001 for (i=1;i<N_i+1;i++)
2002 {
2003     for (j=1;j<N_j+1;j++)
2004     {
2005         outfile << T_1[i][j] << " " ;
2006     }
2007     outfile << endl ;
2008 }
2009 outfile << "];" << endl;
2010 outfile.close();
2011 }
2012
2013 double caseConv2D::Nu_value()
2014 {
2015     unsigned int i , j;
2016     double Nu_average = 0.0;
2017     j = 0;
2018     for (i=1;i<N_i+1;i++)
2019     {
2020         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 ;
2021     }
2022
2023     return Nu_average;
2024 }
2025
2026 double delta_inc (const vector<vector<double>> &A , const vector<vector<double>> &B )
2027 {

```



```

2028   unsigned int i , j;
2029   double temporal = 0.0 , delta = 0.0;
2030
2031   assert(A.size()==B.size() && A[1].size()==B[1].size());
2032
2033   for(i=1;i<A.size()-1;i++)
2034   {
2035       for(j=1;j<A[1].size()-1;j++)
2036       {
2037           temporal = fabs(A[i][j] - B[i][j]) ;
2038           delta = ( (delta > temporal ) ? delta : temporal );
2039       }
2040   }
2041   return delta;
2042 }

```

1.2 Numerical scheme selection program

```

1  #include <iostream>
2  #include <cmath>
3
4  using namespace std;
5
6  enum schValues {
7      CDS,
8      UDS,
9      HDS,
10     PLDS,
11     EDS
12 };
13
14 enum schValues str2schValues(string str){
15     if(str=="CDS") return(CDS);
16     else if(str=="UDS") return(UDS);
17     else if(str=="HDS") return(HDS);
18     else if(str=="PLDS") return(PLDS);
19     else if(str=="EDS") return(EDS);
20     else {
21         cerr<<"Error in " << __FUNCTION__ << ": " << str << " condition doesn't exist." << endl;
22     }
23 }
24
25 double A_scheme (double P, string scheme="CDS"){
26     schValues sch;
27
28     sch=str2schValues(scheme);
29
30     switch (sch){
31         case CDS: //Central diference
32             return 1-0.5*fabs(P);
33         case UDS: //Upwind
34             return 1;
35         case HDS: //Hybrid
36             return (0 > (1 - 0.5*fabs(P)) )? 0 : (1 - 0.5*fabs(P));
37         case PLDS: //power law
38             return ( 0 > pow((1 - 0.1*fabs(P)) , 5) )? 0 : pow((1 - 0.1*fabs(P)) , 5);
39         case EDS: //exponential
40             return fabs(P) / ( exp(fabs(P)) - 1 );
41         default:
42             cerr << "Wrong scheme" << endl;
43             break;
44     }
45 }

```

1.3 Conjugate gradient solver

```

1  #include <iostream>
2  #include <vector>

```

```

3 #include <cmath>
4 #include <cassert>
5
6 using namespace std;
7
8 double max_residu (const vector<vector<double>> > r);
9
10 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)
11 {
12
13     unsigned int i , j;
14     unsigned int iter;
15     unsigned int N=N_x*N_y;
16     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));
17     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));
18     vector<vector<double>> x_0 = phi_0 , x(N_y+2,vector<double>(N_x+2,0.0));
19
20     for (i=1;i<N_y+1;i++)
21     {
22         for (j=1;j<N_x+1;j++)
23         {
24             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] ;
25             r_0[i][j] = r_0[i][j] - b[i][j];
26             p_0[i][j] = - r_0[i][j];
27         }
28     }
29
30
31     for(iter=1; max_residu(r_0) > epsilon && iter<max_iter ; iter++)
32     {
33         double a_0=0.0 ;
34         double temp1 = 0.0 , temp2 = 0.0;
35         for (i=1;i<N_y+1;i++)
36         {
37             for (j=1;j<N_x+1;j++)
38             {
39                 temp1 = temp1 + r_0[i][j]*r_0[i][j];
40                 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]);
41             }
42         }
43         a_0 = temp1 / temp2 ;
44
45         for (i=1;i<N_y+1;i++)
46         {
47             for (j=1;j<N_x+1;j++)
48             {
49                 x[i][j] = x_0[i][j] + a_0 * p_0[i][j];
50             }
51         }
52
53         for (i=1;i<N_y+1;i++)
54         {
55             for (j=1;j<N_x+1;j++)
56             {
57
58                 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]);
59             }
60         }
61
62         double b_1 = 0.0 ;
63         temp1 = 0.0 , temp2=0.0 ;
64         for (i=1;i<N_y+1;i++)
65         {
66             for (j=1;j<N_x+1;j++)

```

```

67     {
68         temp1 = temp1 + r_1[i][j] * r_1[i][j];
69         temp2 = temp2 + r_0[i][j] * r_0[i][j];
70     }
71 }
72 b_1 = temp1 / temp2 ;
73
74 for (i=1;i<N_y+1;i++)
75 {
76     for (j=1;j<N_x+1;j++)
77     {
78         p_1[i][j] = -r_1[i][j] + b_1 * p_0[i][j];
79     }
80 }
81
82 p_0 = p_1 ;
83 r_0 = r_1 ;
84 x_0 = x ;
85
86 }
87 if (iter>=max_iter) cerr << "Warning: max number of iterations achieved:" << max_residu(
    r_0) << endl;
88 assert (max_residu(r_0) < epsilon);
89
90 return x;
91
92 }
93
94 double max_residu (const vector<vector<double>> > r)
95 {
96     double temp = 0.0;
97     unsigned int i , j;
98
99
100    for (i=0;i<r.size();i++)
101    {
102        for (j=0;j<r[1].size();j++)
103        {
104            temp=(temp<fabs(r[i][j]))?fabs(r[i][j]):temp;
105        }
106    }
107
108    return temp;
109
110 }

```

1.4 Biconjugate gradient stabilized solver

```

1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  #include <cassert>
5
6  using namespace std;
7
8  double max_residu (const vector<vector<double>> > r);
9
10 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)
11 {
12
13     unsigned int i , j;
14     unsigned int iter;
15     unsigned int N=N_x*N_y;
16     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));
17     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));

```

```

18  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));
19  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));
20  vector<vector<double>> x_0 = phi_0 , x = phi_0;
21  double rho_0 = 1.0 , rho_1 = 0.0 , alfa = 1.0 , beta = 0.0 , w_0 = 1.0 , w_1 = 0.0 ;
22
23  for (i=1;i<N_y+1;i++)
24  {
25      for (j=1;j<N_x+1;j++)
26      {
27          x[i][j]=0.0;
28      }
29  }
30
31  for (i=1;i<N_y+1;i++)
32  {
33      for (j=1;j<N_x+1;j++)
34      {
35          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] ;
36          r_0[i][j] = - r_0[i][j] + b[i][j];
37      }
38  }
39  r_0tilde = r_0;
40
41  for(iter=1; max_residu(r_0) > epsilon && iter<max_iter ; iter++)
42  {
43      rho_1 = 0.0 ;
44      for (i=1;i<N_y+1;i++)
45      {
46          for (j=1;j<N_x+1;j++)
47          {
48              rho_1 = rho_1 + r_0tilde[i][j]*r_0[i][j];
49          }
50      }
51      beta = rho_1 / rho_0 * alfa / w_0 ;
52
53      for (i=1;i<N_y+1;i++)
54      {
55          for (j=1;j<N_x+1;j++)
56          {
57              p_1[i][j] = r_0[i][j] + beta * (p_0[i][j] - w_0*v_0[i][j]) ;
58          }
59      }
60
61      double temp2 = 0.0 ;
62      for (i=1;i<N_y+1;i++)
63      {
64          for (j=1;j<N_x+1;j++)
65          {
66              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]);
67              temp2 = temp2 + r_0tilde[i][j] * v_1[i][j] ;
68          }
69      }
70
71      alfa = rho_1 / temp2 ;
72
73      for (i=1;i<N_y+1;i++)
74      {
75          for (j=1;j<N_x+1;j++)
76          {
77              s[i][j] = r_0[i][j] - alfa * v_1[i][j];
78          }
79      }
80
81      for (i=1;i<N_y+1;i++)
82      {
83          for (j=1;j<N_x+1;j++)
84          {
85              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]);
86          }
87      }

```

```

88     }
89
90     double temp3 = 0.0 , temp4 = 0.0;
91
92     for (i=1;i<N_y+1;i++)
93     {
94         for (j=1;j<N_x+1;j++)
95         {
96             temp3 = temp3 + t[i][j] * s[i][j] ;
97             temp4 = temp4 + t[i][j] * t[i][j] ;
98         }
99     }
100     w_1 = temp3/temp4;
101
102     for (i=1;i<N_y+1;i++)
103     {
104         for (j=1;j<N_x+1;j++)
105         {
106             x[i][j] = x_0[i][j] + alfa * p_1[i][j] + w_1 * s[i][j];
107             r_1[i][j] = s[i][j] - w_1 * t[i][j] ;
108         }
109     }
110
111     p_0 = p_1 ;
112     v_0 = v_1 ;
113     r_0 = r_1 ;
114     x_0 = x ;
115     rho_0 = rho_1 ;
116     w_0 = w_1 ;
117
118 }
119 if (iter>=max_iter) cerr << "Warning: max number of iterations achieved: " << max_residu(
120     r_0) << endl;
121 assert (max_residu(r_0) < epsilon);
122 return x;
123
124 }

```

1.5 Mesh coordinates program

```

1 #include <iostream>
2 #include <cmath>
3
4 using namespace std;
5
6 double det_coord_Mesh ( double A , unsigned int N , double g , unsigned int i )
7 {
8
9     return A/2.0 * ( 1.0 + tanh( g*( 2.0*(i-1.0)/N - 1.0 ) ) / tanh(g) );
10
11 }

```

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