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

ESEIAAT - UPC

Study for the computational resolution of conservation equations of mass, momentum and energy. Possible application to different aeronautical and industrial engineering problems: Case 1B

Attachment B - C++ codes

Author: Laura Pla Olea

Director: Carlos David Perez Segarra

Co-Director: Asensio Oliva Llena

Degree: Grau en Enginyeria en Tecnologies Aeroespacials

Delivery date: 10-06-2017

Contents

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

1 Four materials problem

```
#include<iostream>
#include<math.h>
#include<fstream>

using namespace std;

// Dimensions
const int M1 = 40;
const int M2 = 30;
const int M3 = 10;
const int N1 = 50;
const int N2 = 60;

// Definition of types
typedef double matrix[M1+M2+M3][N1+N2];


// FUNCTIONS
void horizontal_coordinates (double dx1, double dx2, double xvc[], double x[]);
void vertical_coordinates (double dy1, double dy2, double dy3, double yvc[], double y[]);
void volume (double *xvc, double *yvc, int N, int M, matrix& V);
void surface (double *yvc, int M, double Sx[]);
void properties (double *x, double *y, const float p[3][2], const float rhod[4], const float cpd[4],
    const float lamd[4], matrix& rho, matrix& cp, matrix& lambda);
void harmonic_mean (matrix lambda, double* x, double* y, double* xvc, double* yvc, int N, int M,
    matrix& lambdaw, matrix& lambdae, matrix& lambdas, matrix& lambdan);
void search_index (float point, double *x, int Number, int &ipoint, int& ip);
void constant_coefficients (double *x, double *y, double *xvc, double *yvc, double *Sx, double *Sy,
    matrix V, float dt, float beta, float alpha, matrix rho, matrix cp, matrix lambda, matrix
    lambdaw, matrix lambdae, matrix lambdas, matrix lambdan, matrix& ap, matrix& aw, matrix& ae,
    matrix& as, matrix& an);
void bp_coefficients (double *x, double *y, double *xvc, double *yvc, double *Sx, double *Sy, double
    Sytotal, matrix V, float dt, float beta, float alpha, float Qtop, float qv, float Tbottom, float
    Tgleft, float Tright, float Trightant, matrix Tant, matrix rho, matrix cp, matrix lambda, matrix
    lambdaw, matrix lambdae, matrix lambdas, matrix lambdan, matrix& bp);
void Gauss_Seidel (matrix ap, matrix aw, matrix ae, matrix as, matrix an, matrix bp, float fr, float
    delta, int N, int M, matrix& T);
double double_interpolation (float x, float y, double T11, double T12, double T21, double T22,
```

```
      double x1, double x2, double y1, double y2);
void print_matrix (matrix T, int N, int M);
void output_file (double* Tpoint1, double* Tpoint2, int Time, float dt);


int main(){

        // DATA
        // Coordinates
        const float p[3][2] = {
        {0.50,0.40},
        {0.50,0.70},
        {1.10,0.80}
        }; // [m]

        // Physical properties
        const float rhod[4] = {1500.00,1600.00,1900.00,2500.00}; // [kg/m^3]
        const float cpd[4] = {750.00,770.00,810.00,930.00}; // [J/(kgK)]
        const float lamd[4] = {170.00,140.00,200.00,140.00}; // [W/(mK)]

        // Boundary conditions
        const float Tbottom = 23.00; // [C]
        const float Qtop = 60.00; // [W/m]
        const float Tgleft = 33.00; // [C]
        const float alpha = 9.00; // [W/(m^2K)]
        const float Tright0 = 8.00; // Initial temperature on the right [C]
        const float variationright = 0.005; // Variation of the temperature on the right [s/C]
        const float T0 = 8.00; // Initial temperature [C]
        const float qv = 0; // Internal heat [W/m^3]

        // Results (coordinates)
        const float point[2][2] = {
        {0.65,0.56},
        {0.74,0.72}
        }; // Points to be studied [m]

        // Mathematical properties
        const int Time = 5001; // Time discretization
        const float beta = 0.5;
        const float tfinal = 5000; // Time of the simulation
        const float delta = 0.001; // Precision of the simulation
        const float fr = 1.2; // Relaxation factor


        cout<<"Program started"<<endl;

        // PREVIOUS CALCULATIONS

        float L1,L2,H1,H2,H3; // Dimensions
        L1 = p[0][0];
```

```
79          L2 = p[2][0]-L1;
80          H1 = p[0][1];
81          H2 = p[1][1]-H1;
82          H3 = p[2][1]-H1-H2;
83
84          double dx1, dx2, dy1, dy2, dy3, dt; // Increments of space and time
85          dt = tfinal/(Time-1); // Increment of time
86          dx1 = L1/N1; // Increments in the horizontal direction
87          dx2 = L2/N2;
88          dy1 = H1/M1; // Increments in the vertical direction
89          dy2 = H2/M2;
90          dy3 = H3/M3;
91
92          // Coordinates
93          double xvc[N1+N2+1],yvc[M1+M2+M3+1]; // Coordinates of the faces
94          double x[N1+N2],y[M1+M2+M3]; // Coordinates of the nodes
95          xvc[0] = 0;
96           horizontal_coordinates (dx1, dx2, xvc, x);
97          yvc[0] = p[2][1];
98           vertical_coordinates (dy1, dy2, dy3, yvc, y);
99
100         // Surfaces and volumes
101         double Sx[M1+M2+M3], Sy[N1+N2], V[M1+M2+M3][N1+N2], Sytotal; // Surfaces and volumes
102          Sytotal = p[2][1]; // Total surface of the north face
103         volume (xvc, yvc, N1+N2, M1+M2+M3, V);
104          surface (yvc, M1+M2+M3, Sx);
105          surface (xvc, N1+N2, Sy);
106
107
108         cout<<"Calculating properties ... "<<endl;
109
110         // Density, specific heat and conductivity
111         matrix rho, cp, lambda; // Density, specific heat and conductivity
112          properties (x, y, p, rhod, cpd, lamd, rho, cp, lambda);
113
114
115         // Harmonic mean
116         matrix lambdaw, lambdae, lambdas, lambdan; // Harmonic mean
117         harmonic_mean (lambda, x, y, xvc, yvc, N1+N2, M1+M2+M3, lambdaw, lambdae, lambdas,
                lambdan);
118
119         // INITIALIZATION
120         matrix T, Tant; // Temperature and Temperature in the previous instant of time
121          float Tright, Trightant; // Temperature on the right and Temperature on the right in the
                previous instant of time
122         double Tpoint1[Time], Tpoint2[Time]; // Temperatures at the points that are going to be
                studied
123         for(int i = 0; i<N1+N2; i++)
124         {
125                 for(int j = 0; j<M1+M2+M3; j++)
```

```
126                 {
127                         T[j][i] = T0;
128                         Tant[j][i] = T0;
129                         Tpoint1[0] = T0;
130                         Tpoint2[0] = T0;
131                 }
132         }
133         Tright = Tright0;
134
135         // Searching for the points (0.65, 0.56) and (0.74, 0.72)
136         int ipoint1, jpoint1, ip1, jp1, ipoint2, jpoint2, ip2, jp2;
137     search_index (point [0][0], x, N1+N2, ipoint1, ip1);
138     search_index (point [1][0], x, N1+N2, ipoint2, ip2);
139     search_index (point [0][1], y, M1+M2+M3, jpoint1, jp1);
140     search_index (point [1][1], y, M1+M2+M3, jpoint2, jp2);
141
142
143         // CALCULATION OF CONSTANT COEFFICIENTS
144         matrix ap, ae, aw, as, an, bp; // Coefficients
145          constant_coefficients (x, y, xvc, yvc, Sx, Sy, V, dt, beta, alpha, rho, cp, lambda, lambdaw,
                 lambdae, lambdas, lambdan, ap, aw, ae, as, an);
146
147
148         cout<<"Solving..."<<endl;
149
150         float t = 0.00; // First time increment
151         double resta;
152         double MAX;
153         int k = 0;
154         while(t<=tfinal)
155         {
156                 k = k+1;
157                 t = t+dt;
158                 Trightant = Tright;
159                 Tright = Tright0+variationright*t;
160
161                 // CALCULATION OF NON-CONSTANT COEFFICIENTS
162                  bp_coefficients (x, y, xvc, yvc, Sx, Sy, Sytotal, V, dt, beta, alpha, Qtop, qv,
                         Tbottom, Tgleft, Tright, Trightant, Tant, rho, cp, lambda, lambdaw, lambdae,
                         lambdas, lambdan, bp);
163
164                 // SOLVER
165                 Gauss_Seidel (ap, aw, ae, as, an, bp, fr, delta, N1+N2, M1+M2+M3, T);
166
167                 // Assignation of the instant of time
168                 for(int i = 0; i<N1+N2; i++)
169                 {
170                         for(int j = 0; j<M1+M2+M3; j++)
171                         {
172                                 Tant[j][i] = T[j][i];
```

```
173                         }
174                 }
175
176                 // Temperature at the given points
177                 Tpoint1[k] = double_interpolation(point [0][0], point [0][1], T[jpoint1 ][ ipoint1 ], T[jp1
                        ][ ipoint1 ], T[jpoint1 ][ ip1 ], T[jp1 ][ ip1 ], x[ ipoint1 ], x[ip1 ], y[ jpoint1 ], y[jp1 ])
                        ;
178                 Tpoint2[k] = double_interpolation(point [1][0], point [1][1], T[jpoint2 ][ ipoint2 ], T[jp2
                        ][ ipoint2 ], T[jpoint2 ][ ip2 ], T[jp2 ][ ip2 ], x[ ipoint2 ], x[ip2 ], y[ jpoint2 ], y[jp2 ])
                        ;
179         }
180
181         cout<<endl<<endl<<"Final temperature:"<<endl;
182
183         // Output of the matrix temperature at the final instant of time
184         print_matrix (T, N1+N2, M1+M2+M3);
185
186     // Output file
187     cout<<"Creating file ... "<<endl;
188     output_file (Tpoint1, Tpoint2, Time, dt);
189 //        resultaats (x, y, T, N1+N2, M1+M2+M3);
190
191     cout<<"End of program"<<endl;
192
193     ofstream results ;
194     results .open("Ressultats5000.dat");
195     int N = N1+N2;
196     int M = M1+M2+M3;
197     for(int i = -1; i<N+1; i++)
198     {
199         for(int j = -1; j<M+1; j++)
200         {
201                 if(i==-1 && j==-1)
202                 {
203                         results <<0.000<<"      "<<0.800<<"   "<<(200*T[0][0]/0.005+alpha*Tgleft)/(
                                alpha+200/0.005)<<endl;
204                         }
205                         else if(i==-1 && j==M)
206                         {
207                                 results <<0.000<<"      "<<0.000<<"   "<<23.000<<endl;
208                         }
209                         else if(i==-1 && j!=-1 && j!=M)
210                         {
211                                 results <<0.000<<"      "<<y[j]<<"      "<<(lambda[j][0]*T[j
                                        ][0]/0.005+alpha*Tgleft)/(alpha+lambda[j][0]/0.005)<<endl;
212                         }
213                         else if(i==N && j==-1)
214                         {
215                                 results <<1.100<<"      "<<0.800<<"   "<<8+0.005*tfinal<<endl;
216                         }
```

```
217                         else if(i==N && j==M)
218                         {
219                                 results <<1.100<<"      "<<0.000<<"   "<<8+0.005*tfinal<<endl;
220                         }
221                         else if(i==N && j!=-1 && j!=M)
222                         {
223                                 results <<1.100<<"      "<<y[j]<<"      "<<8+0.005*tfinal<<endl;
224                         }
225                         else if(j==-1 && i!=-1 && i!=N)
226                         {
227                                 results <<x[i]<<"       "<<0.800<<"   "<<T[0][i]+Qtop*0.005/(1.10*
                                        lambda[0][i]*0.005)<<endl;
228                         }
229                         else if(j==M && i!=-1 && i!=N)
230                         {
231                                 results <<x[i]<<"       "<<0.000<<"   "<<23.000<<endl;
232                         }
233                 else
234                 {
235                         results <<x[i]<<"       "<<y[j]<<"      "<<T[j][i]<<endl;
236                         }
237                 }
238                 results <<endl;
239         }
240     results.close();
241
242     return 0;
243
244 }
245
246
247
248 void horizontal_coordinates (double dx1, double dx2, double xvc[], double x[])
249 {
250         for (int i = 1; i<N1+N2+1; i++)
251         {
252                 if(i<=N1)
253                 {
254                         xvc[i] = xvc[i-1]+dx1;
255                         x[i-1] = (xvc[i-1]+xvc[i])/2;
256                 }
257                 else
258                 {
259                         xvc[i] = xvc[i-1]+dx2;
260                         x[i-1] = (xvc[i-1]+xvc[i])/2;
261                 }
262         }
263 }
264
265
```

```
266 void vertical_coordinates (double dy1, double dy2, double dy3, double yvc [], double y [])
267 {
268         for (int j = 1; j<M1+M2+M3+1; j++)
269         {
270                 if (j<=M3)
271                 {
272                         yvc[j] = yvc[j-1]-dy3;
273                         y[j-1] = (yvc[j-1]+yvc[j])/2;
274                 }
275                 else if (j>M3 && j<=M2+M3)
276                 {
277                         yvc[j] = yvc[j-1]-dy2;
278                         y[j-1] = (yvc[j-1]+yvc[j])/2;
279                 }
280                 else
281                 {
282                         yvc[j] = yvc[j-1]-dy1;
283                         y[j-1] = (yvc[j-1]+yvc[j])/2;
284                 }
285         }
286 }
287
288
289 void volume (double *xvc, double *yvc, int N, int M, matrix& V)
290 {
291         for(int i = 0; i<N; i++)
292         {
293                 for(int j = 0; j<M; j++)
294                 {
295                         V[j][i] = fabs(xvc[i+1]-xvc[i])*fabs(yvc[j]-yvc[j+1]); // Volume
296                 }
297         }
298 }
299
300
301 void surface (double *yvc, int M, double Sx[])
302 {
303         for(int j = 0; j<M; j++)
304         {
305                 Sx[j] = fabs(yvc[j]-yvc[j+1]);
306         }
307 }
308
309
310 void properties (double *x, double *y, const float p [3][2], const float rhod [4], const float cpd [4],
        const float lamd [4], matrix& rho, matrix& cp, matrix& lambda)
311 {
312         for(int i = 0; i<N1+N2; i++)
313         {
314                 for(int j = 0; j<M1+M2+M3; j++)
```

```
315                 {
316                         if(x[i]<=p[0][0] && y[j]<=p[0][1])
317                         {
318                                 rho[j][i] = rhod[0];
319                                 cp[j][i] = cpd[0];
320                                 lambda[j][i] = lamd[0];
321                         }
322                         else if(x[i]<=p[0][0] && y[j]>p[0][1])
323                         {
324                                 rho[j][i] = rhod[2];
325                                 cp[j][i] = cpd[2];
326                                 lambda[j][i] = lamd[2];
327                         }
328                         else if(x[i]>p[0][0] && y[j]<=p[1][1])
329                         {
330                                 rho[j][i] = rhod[1];
331                                 cp[j][i] = cpd[1];
332                                 lambda[j][i] = lamd[1];
333                         }
334                         else
335                         {
336                                 rho[j][i] = rhod[3];
337                                 cp[j][i] = cpd[3];
338                                 lambda[j][i] = lamd[3];
339                         }
340                 }
341         }
342 }
343
344
345 void harmonic_mean (matrix lambda, double* x, double* y, double* xvc, double* yvc, int N, int M,
        matrix& lambdaw, matrix& lambdae, matrix& lambdas, matrix& lambdan)
346 {
347         for(int i = 0; i<N; i++)
348         {
349                 for(int j = 0; j<M; j++)
350                 {
351                         if(i==0)
352                         {
353                                 lambdaw[j][i] = lambda[j][i];
354                                 lambdan[j][i] = (y[j-1]-y[j])/((y[j-1]-yvc[j])/lambda[j-1][i]+(yvc[j]-
                                        y[j])/lambda[j][i]);
355                                 lambdae[j][i] = (x[i+1]-x[i])/((x[i+1]-xvc[i+1])/lambda[j][i+1]+(xvc[i
                                        +1]-x[i])/lambda[j][i]);
356                                 lambdas[j][i] = (y[j]-y[j+1])/((yvc[j+1]-y[j+1])/lambda[j+1][i]+(y[j]-
                                        yvc[j+1])/lambda[j][i]);
357                         }
358                         else if(i==N-1)
359                         {
360                                 lambdaw[j][i] = (x[i]-x[i-1])/((x[i]-xvc[i])/lambda[j][i-1]+(x[i]-xvc[i
```

```
                                ])/lambda[j][i]);
361                             lambdan[j][i] = (y[j-1]-y[j])/((y[j-1]-yvc[j])/lambda[j-1][i]+(yvc[j]-
                                    y[j])/lambda[j][i]);
362                             lambdae[j][i] = lambda[j][i];
363                             lambdas[j][i] = (y[j]-y[j+1])/((yvc[j+1]-y[j+1])/lambda[j+1][i]+(y[j]-
                                    yvc[j+1])/lambda[j][i]);
364                         }
365                         else if(j==0)
366                         {
367                             lambdaw[j][i] = (x[i]-x[i-1])/((x[i]-xvc[i])/lambda[j][i-1]+(x[i]-xvc[i
                                    ])/lambda[j][i]);
368                             lambdan[j][i] = lambda[j][i];
369                             lambdae[j][i] = (x[i+1]-x[i])/((x[i+1]-xvc[i+1])/lambda[j][i+1]+(xvc[i
                                    +1]-x[i])/lambda[j][i]);
370                             lambdas[j][i] = (y[j]-y[j+1])/((yvc[j+1]-y[j+1])/lambda[j+1][i]+(y[j]-
                                    yvc[j+1])/lambda[j][i]);
371                         }
372                         else if(j==M-1)
373                         {
374                             lambdaw[j][i] = (x[i]-x[i-1])/((x[i]-xvc[i])/lambda[j][i-1]+(x[i]-xvc[i
                                    ])/lambda[j][i]);
375                             lambdan[j][i] = (y[j-1]-y[j])/((y[j-1]-yvc[j])/lambda[j-1][i]+(yvc[j]-
                                    y[j])/lambda[j][i]);
376                             lambdae[j][i] = (x[i+1]-x[i])/((x[i+1]-xvc[i+1])/lambda[j][i+1]+(xvc[i
                                    +1]-x[i])/lambda[j][i]);
377                             lambdas[j][i] = lambda[j][i];
378                         }
379                         else
380                         {
381                             lambdaw[j][i] = (x[i]-x[i-1])/((x[i]-xvc[i])/lambda[j][i-1]+(x[i]-xvc[i
                                    ])/lambda[j][i]);
382                             lambdan[j][i] = (y[j-1]-y[j])/((y[j-1]-yvc[j])/lambda[j-1][i]+(yvc[j]-
                                    y[j])/lambda[j][i]);
383                             lambdae[j][i] = (x[i+1]-x[i])/((x[i+1]-xvc[i+1])/lambda[j][i+1]+(xvc[i
                                    +1]-x[i])/lambda[j][i]);
384                             lambdas[j][i] = (y[j]-y[j+1])/((yvc[j+1]-y[j+1])/lambda[j+1][i]+(y[j]-
                                    yvc[j+1])/lambda[j][i]);
385                         }
386                     }
387             }
388 }
389
390
391 // Searching the index of the node closest to a given point (and the second closest)
392 void search_index (float point, double *x, int Number, int &ipoint, int& ip)
393 {
394         for(int i = 0; i<Number-1; i++)
395     {
396         if(x[i+1]-x[i]>0)
397         {
```

```
398                 if(x[i]<=point && x[i+1]>point)
399                 {
400                         if(point-x[i]<x[i+1]-point)
401                                 {
402                                         ipoint = i; //ipoint is the index of the node closest to the
                                                point we want
403                                         ip = i+1; //ip is the second node closest to it (used in
                                                interpolation )
404                                 }
405                                 else
406                                 {
407                                         ipoint = i+1;
408                                         ip = i;
409                                 }
410                         }
411                 }
412                 else
413                 {
414                         if(x[i]>point && x[i+1]<=point)
415                 {
416                         if(point-x[i+1]<x[i]-point)
417                         {
418                                 ipoint = i;
419                                 ip = i+1;
420                                 }
421                                 else
422                                 {
423                                         ipoint = i+1;
424                                         ip = i;
425                                 }
426                         }
427                 }
428         }
429
430 }
431
432
433 // Calculation of the constant coefficients
434 void constant_coefficients (double *x, double *y, double *xvc, double *yvc, double *Sx, double *Sy,
        matrix V, float dt, float beta, float alpha, matrix rho, matrix cp, matrix lambda, matrix
        lambdaw, matrix lambdae, matrix lambdas, matrix lambdan, matrix& ap, matrix& aw, matrix& ae,
        matrix& as, matrix& an)
435 {
436         for(int i =0; i<N1+N2; i++)
437         {
438                 for(int j = 0; j<M1+M2+M3; j++)
439                 {
440                         if(i==0 && j==0)
441                         {
442                                 ae[j][i] = beta*lambdae[j][i]*Sx[j]/(x[i+1]-x[i]);
```

```
443                 aw[j][i] = 0;
444                 as[j][i] = beta*lambdas[j][i]*Sy[i]/(y[j]-y[j+1]);
445                 an[j][i] = 0;
446                 ap[j][i] = ae[j][i]+aw[j][i]+as[j][i]+an[j][i]+rho[j][i]*cp[j][i]*V[j
                        ][i]/dt+beta*Sx[j]/(1/alpha+(x[i]-xvc[i])/lambda[j][i]);
447             }
448             else if(i==0 && j!=0 && j!=M1+M2+M3-1)
449             {
450                 ae[j][i] = beta*lambdae[j][i]*Sx[j]/(x[i+1]-x[i]);
451                 aw[j][i] = 0;
452                 as[j][i] = beta*lambdas[j][i]*Sy[i]/(y[j]-y[j+1]);
453                 an[j][i] = beta*lambdan[j][i]*Sy[i]/(y[j-1]-y[j]);
454                 ap[j][i] = ae[j][i]+aw[j][i]+as[j][i]+an[j][i]+rho[j][i]*cp[j][i]*V[j
                        ][i]/dt+beta*Sx[j]/(1/alpha+(x[i]-xvc[i])/lambda[j][i]);
455             }
456             else if(i==0 && j==M1+M2+M3-1)
457             {
458                 ae[j][i] = beta*lambdae[j][i]*Sx[j]/(x[i+1]-x[i]);
459                 aw[j][i] = 0;
460                 as[j][i] = 0;
461                 an[j][i] = beta*lambdan[j][i]*Sy[i]/(y[j-1]-y[j]);
462                 ap[j][i] = ae[j][i]+aw[j][i]+as[j][i]+an[j][i]+rho[j][i]*cp[j][i]*V[j
                        ][i]/dt+beta*Sx[j]/(1/alpha+(x[i]-xvc[i])/lambda[j][i])+beta*
                        lambda[j][i]/(y[j]-yvc[j+1])*Sy[i];
463             }
464             else if(i==N1+N2-1 && j==0)
465             {
466                 ae[j][i] = 0;
467                 aw[j][i] = beta*lambdaw[j][i]*Sx[j]/(x[i]-x[i-1]);
468                 as[j][i] = beta*lambdas[j][i]*Sy[i]/(y[j]-y[j+1]);
469                 an[j][i] = 0;
470                 ap[j][i] = ae[j][i]+aw[j][i]+as[j][i]+an[j][i]+rho[j][i]*cp[j][i]*V[j
                        ][i]/dt+beta*lambda[j][i]*Sx[j]/(xvc[i+1]-x[i]);
471             }
472             else if(i==N1+N2-1 && j==M1+M2+M3-1)
473             {
474                 ae[j][i] = 0;
475                 aw[j][i] = beta*lambdaw[j][i]*Sx[j]/(x[i]-x[i-1]);
476                 as[j][i] = 0;
477                 an[j][i] = beta*lambdan[j][i]*Sy[i]/(y[j-1]-y[j]);
478                 ap[j][i] = ae[j][i]+aw[j][i]+as[j][i]+an[j][i]+rho[j][i]*cp[j][i]*V[j
                        ][i]/dt+beta*lambda[j][i]*Sx[j]/(xvc[i+1]-x[i])+beta*lambda[j][i
                        ]/(y[j]-yvc[j+1])*Sy[i];
479             }
480             else if(i==N1+N2-1 && j!=0 && j!=M1+M2+M3-1)
481             {
482                 ae[j][i] = 0;
483                 aw[j][i] = beta*lambdaw[j][i]*Sx[j]/(x[i]-x[i-1]);
484                 as[j][i] = beta*lambdas[j][i]*Sy[i]/(y[j]-y[j+1]);
485                 an[j][i] = beta*lambdan[j][i]*Sy[i]/(y[j-1]-y[j]);
```

```
486                         ap[j][i] = ae[j][i]+aw[j][i]+as[j][i]+an[j][i]+rho[j][i]*cp[j][i]*V[j
                                ][i]/dt+beta*lambda[j][i]*Sx[j]/(xvc[i+1]-x[i]);
487                     }
488                     else if(i!=0 && i!=N1+N2-1 && j==0)
489                     {
490                         ae[j][i] = beta*lambdae[j][i]*Sx[j]/(x[i+1]-x[i]);
491                         aw[j][i] = beta*lambdaw[j][i]*Sx[j]/(x[i]-x[i-1]);
492                         as[j][i] = beta*lambdas[j][i]*Sy[i]/(y[j]-y[j+1]);
493                         an[j][i] = 0;
494                         ap[j][i] = ae[j][i]+aw[j][i]+as[j][i]+an[j][i]+rho[j][i]*cp[j][i]*V[j
                                ][i]/dt;
495                     }
496                     else if(i!=0 && i!=N1+N2-1 && j==M1+M2+M3-1)
497                     {
498                         ae[j][i] = beta*lambdae[j][i]*Sx[j]/(x[i+1]-x[i]);
499                         aw[j][i] = beta*lambdaw[j][i]*Sx[j]/(x[i]-x[i-1]);
500                         as[j][i] = 0;
501                         an[j][i] = beta*lambdan[j][i]*Sy[i]/(y[j-1]-y[j]);
502                         ap[j][i] = ae[j][i]+aw[j][i]+as[j][i]+an[j][i]+rho[j][i]*cp[j][i]*V[j
                                ][i]/dt+beta*lambda[j][i]*Sy[i]/(y[j]-yvc[j+1]);
503                     }
504                     else
505                     {
506                         ae[j][i] = beta*lambdae[j][i]*Sx[j]/(x[i+1]-x[i]);
507                         aw[j][i] = beta*lambdaw[j][i]*Sx[j]/(x[i]-x[i-1]);
508                         as[j][i] = beta*lambdas[j][i]*Sy[i]/(y[j]-y[j+1]);
509                         an[j][i] = beta*lambdan[j][i]*Sy[i]/(y[j-1]-y[j]);
510                         ap[j][i] = ae[j][i]+aw[j][i]+as[j][i]+an[j][i]+rho[j][i]*cp[j][i]*V[j
                                ][i]/dt;
511                     }
512                 }
513             }
514 }
515
516
517 // Calculation of non-constant coefficients
518 void bp_coefficients (double *x, double *y, double *xvc, double *yvc, double *Sx, double *Sy, double
        Sytotal, matrix V, float dt, float beta, float alpha, float Qtop, float qv, float Tbottom, float
        Tgleft, float Tright, float Trightant, matrix Tant, matrix rho, matrix cp, matrix lambda, matrix
        lambdaw, matrix lambdae, matrix lambdas, matrix lambdan, matrix& bp)
519 {
520         for(int i =0; i<N1+N2; i++)
521                 {
522                     for(int j = 0; j<M1+M2+M3; j++)
523                     {
524                         if(i==0 && j==0)
525                         {
526                             bp[j][i] = rho[j][i]*cp[j][i]*Tant[j][i]*V[j][i]/dt+(1-beta)*
                                    ((Tgleft-Tant[j][i])*Sx[j]/(1/alpha+(x[i]-xvc[i])/lambda[j
                                    ][i])+lambdae[j][i]*(Tant[j][i+1]-Tant[j][i])*Sx[j]/(x[i
```

```
                +1]-x[i])+lambdas[j][i]*(Tant[j+1][i]-Tant[j][i])*Sy[i]/(
                y[j]-y[j+1]))+beta*Tgleft*Sx[j]/(1/alpha+(x[i]-xvc[i])/
                lambda[j][i])+Qtop*Sy[i]/Sytotal+qv*V[j][i];
527         }
528         else if(i==0 && j!=0 && j!=M1+M2+M3-1)
529         {
530             bp[j][i] = rho[j][i]*cp[j][i]*Tant[j][i]*V[j][i]/dt+(1-beta)*
                ((Tgleft-Tant[j][i])*Sx[j]/(1/alpha+(x[i]-xvc[i])/lambda[j
                ][i])+lambdae[j][i]*(Tant[j][i+1]-Tant[j][i])*Sx[j]/(x[i
                +1]-x[i])+lambdas[j][i]*(Tant[j+1][i]-Tant[j][i])*Sy[i]/(
                y[j]-y[j+1])+lambdan[j][i]*(Tant[j-1][i]-Tant[j][i])*Sy[i
                ]/(y[j-1]-y[j]))+beta*Tgleft*Sx[j]/(1/alpha+(x[i]-xvc[i])/
                lambda[j][i])+qv*V[j][i];
531         }
532         else if(i==0 && j==M1+M2+M3-1)
533         {
534             bp[j][i] = rho[j][i]*cp[j][i]*Tant[j][i]*V[j][i]/dt+(1-beta)*
                ((Tgleft-Tant[j][i])*Sx[j]/(1/alpha+(x[i]-xvc[i])/lambda[j
                ][i])+lambdae[j][i]*(Tant[j][i+1]-Tant[j][i])*Sx[j]/(x[i
                +1]-x[i])+lambda[j][i]*(Tbottom-Tant[j][i])/(y[j]-yvc[j
                +1])*Sy[i]+lambdan[j][i]*(Tant[j-1][i]-Tant[j][i])*Sy[i]/(
                y[j-1]-y[j]))+beta*lambda[j][i]*Tbottom/(y[j]-yvc[j+1])*
                Sy[i]+beta*Tgleft*Sx[j]/(1/alpha+(x[i]-xvc[i])/lambda[j][i
                ])+qv*V[j][i];
535         }
536         else if(i==N1+N2-1 && j==0)
537         {
538             bp[j][i] = rho[j][i]*cp[j][i]*Tant[j][i]*V[j][i]/dt+(1-beta)*
                (lambdaw[j][i]*(Tant[j][i-1]-Tant[j][i])*Sx[j]/(x[i]-x[i
                -1])+lambda[j][i]*(Trightant-Tant[j][i])*Sx[j]/(xvc[i
                +1]-x[i])+lambdas[j][i]*(Tant[j+1][i]-Tant[j][i])*Sy[i]/(
                y[j]-y[j+1]))+Qtop*Sy[i]/Sytotal+beta*lambda[j][i]*Tright
                *Sx[j]/(xvc[i+1]-x[i])+qv*V[j][i];
539         }
540         else if(i==N1+N2-1 && j==M1+M2+M3-1)
541         {
542             bp[j][i] = rho[j][i]*cp[j][i]*Tant[j][i]*V[j][i]/dt+(1-beta)*
                (lambdaw[j][i]*(Tant[j][i-1]-Tant[j][i])*Sx[j]/(x[i]-x[i
                -1])+lambda[j][i]*(Trightant-Tant[j][i])*Sx[j]/(xvc[i
                +1]-x[i])+lambda[j][i]*(Tbottom-Tant[j][i])/(y[j]-yvc[j
                +1])*Sy[i]+lambdan[j][i]*(Tant[j-1][i]-Tant[j][i])*Sy[i]/(
                y[j-1]-y[j]))+beta*lambda[j][i]*Tright*Sx[j]/(xvc[i+1]-x
                [i])+beta*lambda[j][i]*Tbottom/(y[j]-yvc[j+1])*Sy[i]+qv*V[
                j][i];
543         }
544         else if(i==N1+N2-1 && j!=0 && j!=M1+M2+M3-1)
545         {
546             bp[j][i] = rho[j][i]*cp[j][i]*Tant[j][i]*V[j][i]/dt+(1-beta)*
                (lambdaw[j][i]*(Tant[j][i-1]-Tant[j][i])*Sx[j]/(x[i]-x[i
                -1])+lambda[j][i]*(Trightant-Tant[j][i])*Sx[j]/(xvc[i
```

```
                                        +1]-x[i])+lambdas[j][i]*(Tant[j+1][i]-Tant[j][i])*Sy[i]/(
                                        y[j]-y[j+1])+lambdan[j][i]*(Tant[j-1][i]-Tant[j][i])*Sy[i
                                        ]/(y[j-1]-y[j]))+beta*lambda[j][i]*Tright*Sx[j]/(xvc[i
                                        +1]-x[i])+qv*V[j][i];
547                             }
548                             else if(i!=0 && i!=N1+N2-1 && j==0)
549                             {
550                             bp[j][i] = rho[j][i]*cp[j][i]*Tant[j][i]*V[j][i]/dt+(1-beta)*(
                                    lambdaw[j][i]*(Tant[j][i-1]-Tant[j][i])*Sx[j]/(x[i]-x[i-1])+
                                    lambdae[j][i]*(Tant[j][i+1]-Tant[j][i])*Sx[j]/(x[i+1]-x[i])+
                                    lambdas[j][i]*(Tant[j+1][i]-Tant[j][i])*Sy[i]/(y[j]-y[j+1]))+Qtop
                                    *Sy[i]/Sytotal+qv*V[j][i];
551                             }
552                             else if(i!=0 && i!=N1+N2-1 && j==M1+M2+M3-1)
553                             {
554                                     bp[j][i] = rho[j][i]*cp[j][i]*Tant[j][i]*V[j][i]/dt+(1-beta)*
                                            (lambdaw[j][i]*(Tant[j][i-1]-Tant[j][i])*Sx[j]/(x[i]-x[i
                                            -1])+lambdae[j][i]*(Tant[j][i+1]-Tant[j][i])*Sx[j]/(x[i
                                            +1]-x[i])+lambda[j][i]*(Tbottom-Tant[j][i])*Sy[i]/(y[j]-
                                            yvc[j+1])+lambdan[j][i]*(Tant[j-1][i]-Tant[j][i])*Sy[i]/(y
                                            [j-1]-y[j]))+beta*lambda[j][i]*Tbottom*Sy[i]/(y[j]-yvc[j
                                            +1])+qv*V[j][i];
555                             }
556                             else
557                             {
558                                     bp[j][i] = rho[j][i]*cp[j][i]*Tant[j][i]*V[j][i]/dt+(1-beta)*
                                            (lambdaw[j][i]*(Tant[j][i-1]-Tant[j][i])*Sx[j]/(x[i]-x[i
                                            -1])+lambdae[j][i]*(Tant[j][i+1]-Tant[j][i])*Sx[j]/(x[i
                                            +1]-x[i])+lambdas[j][i]*(Tant[j+1][i]-Tant[j][i])*Sy[i]/(
                                            y[j]-y[j+1])+lambdan[j][i]*(Tant[j-1][i]-Tant[j][i])*Sy[i
                                            ]/(y[j-1]-y[j]))+qv*V[j][i];
559                             }
560                     }
561             }
562 }
563
564
565 // Solver (using Gauss-Seidel)
566 void Gauss_Seidel (matrix ap, matrix aw, matrix ae, matrix as, matrix an, matrix bp, float fr, float
        delta, int N, int M, matrix& T)
567 {
568         double Tcalc[M][N]; // Temperature calculated in the previous iteration
569         for(int i = 0; i<N; i++)
570         {
571                 for(int j = 0; j<M; j++)
572                 {
573                         Tcalc[j][i] = T[j][i];
574                 }
575         }
576
```

```
577     double MAX = 1; // Maximum value of the difference between T and Tcalc
578     double resta = 1; // Difference between T and Tcalc
579
580     while(MAX>delta)
581     {
582
583         // SOLVER: Gauss-Seidel
584         for(int i = 0; i<N; i++)
585         {
586             for(int j = 0; j<M; j++)
587             {
588                 if(i==0 && j==0)
589                 {
590                     T[j][i] = Tcalc[j][i]+fr*((ae[j][i]*Tcalc[j][i+1]+as[j][i]*
                            Tcalc[j+1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
591                 }
592                 else if(i==0 && j==M-1)
593                 {
594                     T[j][i] = Tcalc[j][i]+fr*((ae[j][i]*Tcalc[j][i+1]+an[j][i]*T[j
                            -1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
595                 }
596                 else if(i==0 && j!=0 && j!=M-1)
597                 {
598                     T[j][i] = Tcalc[j][i]+fr*((ae[j][i]*Tcalc[j][i+1]+as[j][i]*
                            Tcalc[j+1][i]+an[j][i]*T[j-1][i]+bp[j][i])/ap[j][i]-Tcalc
                            [j][i]);
599                 }
600                 else if(i==N-1 && j==0)
601                 {
602                     T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+as[j][i]*Tcalc[j
                            +1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
603                 }
604                 else if(i==N-1 && j==M-1)
605                 {
606                     T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+an[j][i]*T[j-1][
                            i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
607                 }
608                 else if(i==N && j!=0 && j!=M-1)
609                 {
610                     T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+as[j][i]*Tcalc[j
                            +1][i]+an[j][i]*T[j-1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i])
                            ;
611                 }
612                 else if(i!=0 && i!=N-1 && j==0)
613                 {
614                     T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+ae[j][i]*Tcalc[j
                            ][i+1]+as[j][i]*Tcalc[j+1][i]+bp[j][i])/ap[j][i]-Tcalc[j
                            ][i]);
615                 }
616                 else if(i!=0 && i!=N-1 && j==M-1)
```

```
617                                 {
618                                         T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+ae[j][i]*Tcalc[j
                                                ][i+1]+an[j][i]*T[j-1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i])
                                                ;
619                                 }
620                                 else
621                                 {
622                                         T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+ae[j][i]*Tcalc[j
                                                ][i+1]+as[j][i]*Tcalc[j+1][i]+an[j][i]*T[j-1][i]+bp[j][i
                                                ])/ap[j][i]-Tcalc[j][i]);
623                                 }
624                         }
625                 }
626
627                 // Comprovation
628                 MAX = 0;
629                 for(int i = 0; i<N; i++)
630                 {
631                         for(int j = 0; j<M; j++)
632                         {
633                                 resta = fabs(Tcalc[j][i]-T[j][i]);
634
635                                 if(resta>MAX)
636                                 {
637                                         MAX = resta;
638                                 }
639                         }
640                 }
641
642                 // New assignation
643                 for(int i = 0; i<N; i++)
644                 {
645                         for(int j = 0; j<M; j++)
646                         {
647                                 Tcalc[j][i] = T[j][i];
648                         }
649                 }
650         }
651 }
652
653
654 // Double interpolation
655 double double_interpolation (float x, float y, double T11, double T12, double T21, double T22,
        double x1, double x2, double y1, double y2)
656 {
657         double result1, result2, finalresult;
658          result1 = T11+(T21-T11)*(x-x1)/(x2-x1);
659          result2 = T12+(T22-T12)*(x-x1)/(x2-x1);
660          finalresult = result1+(result2-result1)*(y-y1)/(y2-y1);
661         return finalresult;
```

```
662 }
663
664
665 // Print matrix
666 void print_matrix (matrix T, int N, int M)
667 {
668         for(int j = 0; j<M; j++)
669     {
670         for(int i = 0; i<N; i++)
671         {
672             cout<<T[j][i]<<"   "; // display the current element out of the array
673         }
674                 cout<<endl; // go to a new line
675     }
676 }
677
678 // Create an output file with the results
679 void output_file (double* Tpoint1, double* Tpoint2, int Time, float dt)
680 {
681         ofstream puntss;
682     puntss.open("Punts.dat");
683     float t = 0;
684     for(int k = 0; k<Time; k++)
685     {
686         puntss<<t<<" "<<Tpoint1[k]<<"       "<<Tpoint2[k]<<"\n";
687         t = t+dt;
688         }
689     puntss.close();
690 }
```

2 | Smith-Hutton problem

```
#include <iostream>
#include <math.h>
#include<fstream>

using namespace std;

// Numerical parameters
const int N = 200;
const int M = 100;

typedef double matrix[M+2][N+2];
typedef double mface[M+1][N+1];


// FUNCTIONS
void coordinates(float x0, float xN, float dx, int N, float xvc[], float x[]);
void surface(float *yvc, int M, float Sv[]);
void volume(float *xvc, float *yvc, int N, int M, matrix& V);
void velocity(float *x, float *y, int N, int M, mface& u, mface& v);
void mass_flow(float rho, int N, int M, float *Sv, float *Sh, float *xvc, float *yvc, mface& mflowx,
    mface& mflowy);
void phi_inlet_outlet(float *x, float alpha, int N, double phis[]);
void search_index (float point, float *x, int Number, int& ipoint, int& ip);
double max(double a, double b);
double Aperator(string method, double P);
void constant_coefficients (int N, int M, string method, float rho0, float gamma, float dt, float Sp,
    float *x, float *y, float *Sh, float *Sv, matrix V, mface mflowx, mface mflowy, matrix& ae,
    matrix& aw, matrix& an, matrix& as, matrix& ap);
void bp_coefficient (int N, int M, float rho0, float dt, float Sc, float *x, double phi_boundary,
    double *phis, matrix phi0, matrix V, matrix& bp);
void Gauss_Seidel (matrix ap, matrix aw, matrix ae, matrix as, matrix an, matrix bp, float fr, float
    delta, int N, int M, matrix& T);
void solver(string method, float rho, float gamma, float dt, float fr, float delta, float Sp, float Sc
    , double phi_boundary, double *phis, float *x, float *y, float *Sh, float *Sv, matrix V, mface
    mflowx, mface mflowy, float *xfinal, int index[2][11], double phi1[11]);
void output_matrix(int N, int M, matrix mat);
void output_file (matrix T, int N);
double interpolation(float x, double T1, double T2, double x1, double x2);
```

```
32
33
34 int main(){
35
36         cout<<"Program started"<<endl<<endl;
37
38         // DATA
39         float alpha = 10; // Angle [degrees]
40         float rho = 1; // Density
41         float Sc = 0; // Source term = Sc+Sp*phi
42         float Sp = 0;
43         string method = "EDS";
44
45         float delta = 0.000000001; // Precision of the simulation
46         float fr = 1.1; // Relaxation factor
47
48
49         // PREVIOUS CALCULATIONS
50
51         // Increments
52         float dx, dy, dt;
53         dx = 2.0/N;
54         dy = 1.0/M;
55         dt = 1;
56
57         // Coordinates
58         float xvc[N+1], yvc[M+1]; // Coordinates of the faces
59         float x[N+2], y[M+2]; // Coordinates of the nodes
60
61
62         coordinates(-1, 1, dx, N+1, xvc, x);
63         coordinates(1, 0, -dy, M+1, yvc, y);
64
65
66         // Surfaces and volumes
67         float Sh[N+2], Sv[M+2];
68         matrix V;
69         surface(yvc, M+2, Sv);
70         surface(xvc, N+2, Sh);
71         volume(xvc, yvc, N+2, M+2, V);
72
73
74         // Mass flow on the faces
75         mface mflowx, mflowy;
76         mass_flow(rho, N+1, M+1, Sv, Sh, xvc, yvc, mflowx, mflowy);
77
78
79         // Boundary conditions
80         double phi_boundary, phis[N+1];
81         phi_inlet_outlet(x, alpha, N+2, phis);
```

```
82          phi_boundary = 1-tanh(alpha);
83
84
85          // Output coordinates
86          float  xfinal [11];
87          int  index [2][11];
88          xfinal [0] = 0;
89          for(int  i = 0; i<11; i++)
90          {
91                  if(i==0)
92                  {
93                          xfinal [i] = 0;
94                  }
95                  else
96                  {
97                          xfinal [i] = xfinal [i-1]+0.1;
98                  }
99                  search_index ( xfinal [i], x, N+2, index[0][i], index [1][ i]);
100         }
101         index [0][10] = N+2;
102
103
104         // Resolution
105         float  gamma;
106         double phi1 [11], phi2 [11], phi3 [11];
107
108         cout<<"Solving rho/gamma = 10..."<<endl;
109         gamma = rho/10;
110         solver(method, rho, gamma, dt, fr, delta, Sp, Sc, phi_boundary, phis, x, y, Sh, Sv, V, mflowx,
                mflowy, xfinal , index, phi1);
111
112
113         cout<<"Solving rho/gamma = 1000..."<<endl;
114         gamma = rho/1000;
115         solver(method, rho, gamma, dt, fr, delta, Sp, Sc, phi_boundary, phis, x, y, Sh, Sv, V, mflowx,
                mflowy, xfinal , index, phi2);
116
117         cout<<"Solving rho/gamma = 1000000..."<<endl;
118         gamma = rho/1000000;
119         solver(method, rho, gamma, dt, fr, delta, Sp, Sc, phi_boundary, phis, x, y, Sh, Sv, V, mflowx,
                mflowy, xfinal , index, phi3);
120
121
122         cout<<endl<<"Creating an output file..."<<endl;
123         ofstream  results ;
124     results .open("Resultats.dat");
125     for(int  k = 0; k<11; k++)
126     {
127          results <<xfinal[k]<<"  "<<phi1[k]<<" "<<phi2[k]<<" "<<phi3[k]<<"\n";
128         }
```

```
129     results.close();
130
131     return 0;
132 }
133
134
135
136 void coordinates(float x0, float xN, float dx, int N, float xvc[], float x[])
137 {
138         xvc[0] = x0;
139         x[0] = xvc[0];
140         for(int i = 0; i<N; i++)
141         {
142                 xvc[i+1] = xvc[i]+dx;
143                 x[i+1] = (xvc[i+1]+xvc[i])/2;
144         }
145         x[N] = xN;
146         xvc[0] = x0+dx/2;
147         xvc[N-1] = xN-dx/2;
148 }
149
150
151 void surface(float *yvc, int M, float Sv[])
152 {
153         for(int j = 0; j<M; j++)
154         {
155                 Sv[j] = fabs(yvc[j]-yvc[j+1]);
156                 if(j==M-1)
157                 {
158                         Sv[j] = Sv[j-1];
159                 }
160         }
161 }
162
163
164 void volume(float *xvc, float *yvc, int N, int M, matrix& V)
165 {
166         for(int i = 0; i<N; i++)
167         {
168                 for(int j = 0; j<M; j++)
169                 {
170                         V[j][i] = fabs(xvc[i]-xvc[i-1])*fabs(yvc[j-1]-yvc[j]);
171                 }
172         }
173 }
174
175
176 void velocity(float *x, float *y, int N, int M, mface& u, mface& v)
177 {
178         for(int i = 0; i<N; i++)
```

```
179         {
180                 for(int j = 0; j<M; j++)
181                 {
182                         u[j][i] = 2*y[j]*(1-pow(x[i],2));
183                         v[j][i] = -2*x[i]*(1-pow(y[j],2));
184                 }
185         }
186 }
187
188
189 void mass_flow(float rho, int N, int M, float *Sv, float *Sh, float *xvc, float *yvc, mface& mflowx,
        mface& mflowy)
190 {
191         for(int i = 0; i<N; i++)
192         {
193                 for(int j = 0; j<M; j++)
194                 {
195                         mflowx[j][i] = rho*Sv[j]*2*yvc[j]*(1-pow(xvc[i],2));
196                         mflowy[j][i] = -rho*Sh[i]*2*xvc[i]*(1-pow(yvc[j],2));
197                 }
198         }
199 }
200
201
202 void phi_inlet_outlet(float *x, float alpha, int N, double phis[])
203 {
204         for(int i = 0; i<N; i++)
205         {
206                 if(x[i]<=0)
207                 {
208                         phis[i] = 1+tanh(alpha*(2*x[i]+1));
209                 }
210                 else
211                 {
212                         phis[i] = 0;
213                 }
214         }
215 }
216
217
218 // Searching the index of the node closest to a given point (and the second closest)
219 void search_index (float point, float *x, int Number, int& ipoint, int& ip)
220 {
221         for(int i = 0; i<Number-1; i++)
222     {
223         if(x[i+1]-x[i]>0)
224         {
225                 if(x[i]<=point && x[i+1]>point)
226                 {
227                         if(point-x[i]<x[i+1]-point)
```

```
228                                     {
229                                             ipoint = i; //ipoint is the index of the node closest to the
                                                    point we want
230                                             ip = i+1; //ip is the second node closest to it (used in
                                                    interpolation )
231                                     }
232                                     else
233                                     {
234                                             ipoint = i+1;
235                                             ip = i;
236                                     }
237                             }
238                     }
239                     else
240                     {
241                             if(x[i]>point && x[i+1]<=point)
242                     {
243                             if(point-x[i+1]<x[i]-point)
244                             {
245                                     ipoint = i;
246                                     ip = i+1;
247                                     }
248                                     else
249                                     {
250                                             ipoint = i+1;
251                                             ip = i;
252                                     }
253                             }
254                     }
255             }
256
257 }
258
259
260 double max(double a, double b)
261 {
262             if(a>b)
263             {
264                     return a;
265             }
266             else
267             {
268                     return b;
269             }
270 }
271
272
273 double Aperator(string method, double P)
274 {
275             double A;
```

```
276         if(method=="CDS") // Central Differencing Scheme
277         {
278                 A = 1-0.5*fabs(P);
279         }
280         else if(method=="UDS") // Upwind Differencing Scheme
281         {
282                 A = 1;
283         }
284         else if(method=="HDS") // Hybrid Differencing Scheme
285         {
286                 A = max(0,1-0.5*fabs(P));
287         }
288         else if(method=="PLDS") // Power Law Differencing Scheme
289         {
290                 A = max(0,pow(1-0.1*fabs(P),5));
291         }
292         else if(method=="EDS") // Exponential Differencing Scheme
293         {
294                 A = fabs(P)/(exp(fabs(P))-1);
295         }
296         return A;
297 }
298
299
300 void constant_coefficients (int N, int M, string method, float rho0, float gamma, float dt, float Sp,
        float *x, float *y, float *Sh, float *Sv, matrix V, mface mflowx, mface mflowy, matrix& ae,
        matrix& aw, matrix& an, matrix& as, matrix& ap)
301 {
302         double De, Dw, Dn, Ds;
303         double Pe, Pw, Pn, Ps;
304         double Fe, Fw, Fn, Fs;
305
306         for(int i = 0; i<N; i++)
307         {
308                 for(int j = 0; j<M; j++)
309                 {
310                         if(j==M-1 && x[i]>=0)
311                         {
312                                 ae[j][i] = 0;
313                                 aw[j][i] = 0;
314                                 an[j][i] = 1;
315                                 as[j][i] = 0;
316                                 ap[j][i] = 1;
317                         }
318                         else if(j==M-1 && x[i]<0)
319                         {
320                                 ae[j][i] = 0;
321                                 aw[j][i] = 0;
322                                 an[j][i] = 0;
323                                 as[j][i] = 0;
```

```
324                         ap[j][i] = 1;
325                     }
326                     else if(i==0)
327                     {
328                         ae[j][i] = 0;
329                         aw[j][i] = 0;
330                         an[j][i] = 0;
331                         as[j][i] = 0;
332                         ap[j][i] = 1;
333                     }
334                     else if(j==0)
335                     {
336                         ae[j][i] = 0;
337                         aw[j][i] = 0;
338                         an[j][i] = 1;
339                         as[j][i] = 0;
340                         ap[j][i] = 1;
341                     }
342                     else if(i==N-1)
343                     {
344                         ae[j][i] = 0;
345                         aw[j][i] = 0;
346                         an[j][i] = 1;
347                         as[j][i] = 0;
348                         ap[j][i] = 1;
349                     }
350                     else
351                     {
352                         De = gamma*Sh[i]/fabs(x[i+1]-x[i]);
353                         Dw = gamma*Sh[i-1]/fabs(x[i]-x[i-1]);
354                         Dn = gamma*Sv[j-1]/fabs(y[j-1]-y[j]);
355                         Ds = gamma*Sv[j]/fabs(y[j]-y[j+1]);
356                         Fe = mflowx[j-1][i];
357                         Fw = mflowx[j-1][i-1];
358                         Fn = mflowy[j-1][i-1];
359                         Fs = mflowy[j][i-1];
360                         Pe = Fe/De;
361                         Pw = Fw/Dw;
362                         Pn = Fn/Dn;
363                         Ps = Fs/Ds;
364                         ae[j][i] = De*Aperator(method,Pe)+max(-Fe,0);
365                         aw[j][i] = Dw*Aperator(method,Pw)+max(Fw,0);
366                         an[j][i] = Dn*Aperator(method,Pn)+max(-Fn,0);
367                         as[j][i] = Ds*Aperator(method,Ps)+max(Fs,0);
368                         ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i]+rho0*V[j][i]/dt-Sp*V[j
                                ][i];
369                     }
370                 }
371             }
372 }
```

```
373
374
375 void bp_coefficient (int N, int M, float rho0, float dt, float Sc, float *x, double phi_boundary,
        double *phis, matrix phi0, matrix V, matrix& bp)
376 {
377         for(int i = 0; i<N; i++)
378         {
379                 for(int j = 0; j<M; j++)
380                 {
381                         if(j==M-1 && x[i]>=0)
382                         {
383                                 bp[j][i] = 0;
384                         }
385                         else if(j==M-1 && x[i]<0)
386                         {
387                                 bp[j][i] = phis[i];
388                         }
389                         else if(i==0)
390                         {
391                                 bp[j][i] = phi_boundary;
392                         }
393                         else if(j==0)
394                         {
395                                 bp[j][i] = phi_boundary;
396                         }
397                         else if(i==N-1)
398                         {
399                                 bp[j][i] = phi_boundary;
400                         }
401                         else
402                         {
403                                 bp[j][i] = rho0*V[j][i]*phi0[j][i]/dt+Sc*V[j][i];
404                         }
405                 }
406         }
407 }
408
409
410 // Solver (using Gauss-Seidel)
411 void Gauss_Seidel (matrix ap, matrix aw, matrix ae, matrix as, matrix an, matrix bp, float fr, float
        delta, int N, int M, matrix& T)
412 {
413         double Tcalc[M][N]; // Temperature calculated in the previous iteration
414         for(int i = 0; i<N; i++)
415         {
416                 for(int j = 0; j<M; j++)
417                 {
418                         Tcalc[j][i] = T[j][i];
419                 }
420         }
```

```
421
422     double MAX = 1; // Maximum value of the difference between T and Tcalc
423     double resta = 1; // Difference between T and Tcalc
424
425     while(MAX>delta)
426     {
427
428         // SOLVER: Gauss-Seidel
429         for(int i = 0; i<N; i++)
430         {
431             for(int j = 0; j<M; j++)
432             {
433                 if(i==0 && j==0)
434                 {
435                     T[j][i] = Tcalc[j][i]+fr*((ae[j][i]*Tcalc[j][i+1]+as[j][i]*
                            Tcalc[j+1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
436                 }
437                 else if(i==0 && j==M-1)
438                 {
439                     T[j][i] = Tcalc[j][i]+fr*((ae[j][i]*Tcalc[j][i+1]+an[j][i]*T[j
                            -1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
440                 }
441                 else if(i==0 && j!=0 && j!=M-1)
442                 {
443                     T[j][i] = Tcalc[j][i]+fr*((ae[j][i]*Tcalc[j][i+1]+as[j][i]*
                            Tcalc[j+1][i]+an[j][i]*T[j-1][i]+bp[j][i])/ap[j][i]-Tcalc
                            [j][i]);
444                 }
445                 else if(i==N-1 && j==0)
446                 {
447                     T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+as[j][i]*Tcalc[j
                            +1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
448                 }
449                 else if(i==N-1 && j==M-1)
450                 {
451                     T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+an[j][i]*T[j-1][
                            i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
452                 }
453                 else if(i==N && j!=0 && j!=M-1)
454                 {
455                     T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+as[j][i]*Tcalc[j
                            +1][i]+an[j][i]*T[j-1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i])
                            ;
456                 }
457                 else if(i!=0 && i!=N-1 && j==0)
458                 {
459                     T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+ae[j][i]*Tcalc[j
                            ][i+1]+as[j][i]*Tcalc[j+1][i]+bp[j][i])/ap[j][i]-Tcalc[j
                            ][i]);
460                 }
```

```
461                             else if(i!=0 && i!=N-1 && j==M-1)
462                             {
463                                     T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+ae[j][i]*Tcalc[j
                                            ][i+1]+an[j][i]*T[j-1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i])
                                            ;
464                             }
465                             else
466                             {
467                                     T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+ae[j][i]*Tcalc[j
                                            ][i+1]+as[j][i]*Tcalc[j+1][i]+an[j][i]*T[j-1][i]+bp[j][i
                                            ])/ap[j][i]-Tcalc[j][i]);
468                             }
469                     }
470             }
471
472             // Comprovation
473             MAX = 0;
474             for(int i = 0; i<N; i++)
475             {
476                     for(int j = 0; j<M; j++)
477                     {
478                             resta = fabs(Tcalc[j][i]-T[j][i]);
479
480                             if(resta>MAX)
481                             {
482                                     MAX = resta;
483                             }
484                     }
485             }
486
487             // New assignation
488             for(int i = 0; i<N; i++)
489             {
490                     for(int j = 0; j<M; j++)
491                     {
492                             Tcalc[j][i] = T[j][i];
493                     }
494             }
495     }
496 }
497
498
499 double interpolation(float x, double T1, double T2, double x1, double x2)
500 {
501     double result;
502      result = T1+(T2-T1)*(x-x1)/(x2-x1);
503     return result;
504 }
505
506
```

```
507 void solver(string method, float rho, float gamma, float dt, float fr, float delta, float Sp, float Sc
      , double phi_boundary, double *phis, float *x, float *y, float *Sh, float *Sv, matrix V, mface
      mflowx, mface mflowy, float *xfinal, int index[2][11], double phi1[11])
508 {
509         matrix phi,phi0;
510
511         for(int i = 0; i<N+2; i++)
512         {
513                 for(int j = 0; j<M+2; j++)
514                 {
515                         phi[j][i] = 1;
516                 }
517         }
518
519
520         matrix ae, aw, an, as, ap, bp;
521          constant_coefficients (N+2, M+2, method, rho, gamma, dt, Sp, x, y, Sh, Sv, V, mflowx, mflowy,
                ae, aw, an, as, ap);
522
523
524         float resta = 1;
525
526         while(resta>delta)
527         {
528                 //New increment of time
529                 for(int i = 0; i<N+2; i++)
530                 {
531                         for(int j = 0; j<M+2; j++)
532                         {
533                                 phi0[j][i] = phi[j][i];
534                         }
535                 }
536
537                 bp_coefficient (N+2, M+2, rho, dt, Sc, x, phi_boundary, phis, phi0, V, bp);
538                 Gauss_Seidel (ap, aw, ae, as, an, bp, fr, delta, N+2, M+2, phi);
539
540                 resta = 0;
541                 for(int i = 0; i<N+2; i++)
542                 {
543                         for(int j = 0; j<M+2; j++)
544                         {
545                                 resta = max(resta, fabs(phi[j][i]-phi0[j][i]));
546                         }
547                 }
548         }
549
550         for(int i = 0; i<11; i++)
551         {
552                 phi1[i] = interpolation(xfinal[i], phi[M+1][index[0][i]], phi[M+1][index[1][i]], x[
                      index[0][i]], x[index[1][i]]);
```

```
553             }
554 }
```

3 | Driven cavity problem

```
1  #include<iostream>
2  #include<math.h>
3  #include<fstream>
4
5  using namespace std;
6
7  // Numerical parameters
8  const int N = 100;
9  const int M = 100;
10
11 typedef double matrix[M+2][N+2];
12 typedef double staggx[M+2][N+1];
13 typedef double staggy[M+1][N+2];
14
15 void coordinates(float L, int N, double xvc[], double x[]);
16 void surface(double *yvc, int M, double Sv[]);
17 void volume(double *xvc, double *yvc, int N, int M, matrix& V);
18 void constant_coefficients (int N, int M, double *x, double *y, double *Sv, double *Sh, matrix& ae,
       matrix& aw, matrix& an, matrix& as, matrix& ap);
19 double convective_term (double xf, double x2, double x3, double u2, double u3);
20 void intermediate_velocities (int N, int M, float rho, float mu, float delta, double dt, double* x,
       double* y, double *xvc, double* yvc, double* Sh, double* Sv, matrix V, staggx u0, staggy v0,
       staggx Ru0, staggy Rv0, staggx &Ru, staggy &Rv, staggx &up, staggy &vp);
21 void bp_coefficient (int N, int M, float rho, double dt, double* Sh, double* Sv, staggx up, staggy vp,
       matrix bp);
22 void Gauss_Seidel (matrix ap, matrix aw, matrix ae, matrix as, matrix an, matrix bp, float fr, float
       delta, int N, int M, matrix& T);
23 void velocities (int N, int M, float rho, double dt, float uref, double* x, double* y, matrix p,
       staggx up, staggy vp, staggx &u, staggy &v);
24 double min(double a, double b);
25 double max(double a, double b);
26 double time_step (double dtd, double* x, double* y, staggx u, staggy v);
27 double error (int N, int M, staggx u, staggy v, staggx u0, staggy v0);
28 void search_index (float point, double *x, int Number, int& ipoint, int& ip);
29 double interpolation (float x, double T1, double T2, double x1, double x2);
30 void output_files (int N, int M, float L, double* x, double* y, double* xvc, double* yvc, staggx u,
       staggy v);
31
```

```
32
33 int main()
34 {
35         int Re = 100; // Reynolds number
36         float L = 1; // Length of the cavity
37         float rho = 1; // Density
38         float uref = 1; // Reference velocity
39         float mu = rho*uref*L/Re; // Viscosity
40
41         float delta = 1e-5; // Precision of the simulation (as the Re increases it is recommended to
               use 5e-5, 1e-4, 2e-4...)
42         float fr = 1.2; // Relaxation factor
43
44         cout<<"Program started"<<endl;
45         cout<<"Re="<<Re<<endl<<endl;
46
47         // Coordinates
48         double xvc[N+1], yvc[M+1], x[N+2], y[M+2];
49         coordinates(L, N, xvc, x);
50         coordinates(L, M, yvc, y);
51
52         // Surfaces
53         double Sh[N+2], Sv[M+2];
54         matrix V;
55         surface(xvc, N+2, Sh); // Horizontal surface
56         surface(yvc, M+2, Sv); // Vertical surface
57         volume(xvc, yvc, N+2, M+2, V); // Volume
58
59
60         // Properties that are going to be calculated
61         matrix p; // Values in the nodes (pressure)
62         staggx u, u0, Ru0; // Values in the points given by the staggered meshes ( velocities )
63         staggy v, v0, Rv0;
64
65         // Inicialization
66         for(int j = 0; j<M+2; j++)
67         {
68                 for(int i = 0; i<N+1; i++)
69                 {
70                         if(j==M+1 && i!=0 && i!=N)
71                         {
72                                 u0[j][i] = uref; // Horizontal velocity at n
73                         }
74                         else
75                         {
76                                 u0[j][i] = 0; // Horizontal velocity at n
77                         }
78                         Ru0[j][i] = 0; // R (horizontal) at n-1
79                 }
80         }
```

```
81      for(int j = 0; j<M+1; j++)
82      {
83          for(int i = 0; i<N+2; i++)
84          {
85              v0[j][i] = 0; // Vertical velocity at n
86              Rv0[j][i] = 0; // R ( vertical ) at n-1
87          }
88      }
89
90      // Calculation of the constant coefficients that are used to determine the pressure
91      matrix ae, aw, an, as, ap, bp;
92       constant_coefficients (N+2, M+2, x, y, Sv, Sh, ae, aw, an, as, ap);
93
94      // Time step (CFL condition)
95      double resta = 1;
96      double dtd = 0.2*rho*pow(x[2]-xvc[1],2)/mu;
97      double dtc = 0.35*fabs(x[2]-xvc[1])/uref;
98      double dt = min(dtd, dtc);
99
100     staggx up, Ru; // Intermediate velocities
101     staggy vp, Rv;
102
103     cout<<"Solving..."<<endl;
104     // Fractional Step Method
105     while(resta>delta)
106     {
107         // STEP 1 !!! : INTERMEDIATE VELOCITY
108          intermediate_velocities (N, M, rho, mu, delta, dt, x, y, xvc, yvc, Sh, Sv, V, u0, v0,
                 Ru0, Rv0, Ru, Rv, up, vp);
109
110
111         // STEP 2 !!! : PRESSURE
112          bp_coefficient (N+2, M+2, rho, dt, Sh, Sv, up, vp, bp);
113         Gauss_Seidel (ap, aw, ae, as, an, bp, fr, delta, N+2, M+2, p);
114
115
116         // STEP 3 !!! : VELOCITY
117          velocities (N, M, rho, dt, uref, x, y, p, up, vp, u, v);
118
119
120         // STEP 4 !!! : TIME STEP
121         dt = time_step (dtd, x, y, u, v);
122
123
124         // Comprovation
125          resta = error (N, M, u, v, u0, v0);
126
127         // New time step
128         for(int i = 0; i<N+1; i++)
129         {
```

```
130                         for(int j = 0; j<M+2; j++)
131                         {
132                                 u0[j][i] = u[j][i];
133                                 Ru0[j][i] = Ru[j][i];
134                         }
135                 }
136                 for(int i = 0; i<N+2; i++)
137                 {
138                         for(int j = 0; j<M+1; j++)
139                         {
140                                 v0[j][i] = v[j][i];
141                                 Rv0[j][i] = Rv[j][i];
142                         }
143                 }
144         }
145
146         // Results
147     cout<<endl<<"Creating some output files..."<<endl;
148     output_files (N, M, L, x, y, xvc, yvc, u, v);
149
150         return 0;
151 }
152
153
154 // Coordinates of the control volumes (x -> nodes, xvc -> faces)
155 void coordinates(float L, int N, double xvc[], double x[])
156 {
157         double dx = L/N;
158         xvc[0] = 0;
159         x[0] = 0;
160         for(int i = 0; i<N; i++)
161         {
162                 xvc[i+1] = xvc[i]+dx;
163                 x[i+1] = (xvc[i+1]+xvc[i])/2;
164         }
165         x[N+1] = L;
166 }
167
168
169 // Surfaces of the control volumes
170 void surface(double *yvc, int M, double Sv[])
171 {
172         for(int j = 0; j<M-1; j++)
173         {
174                 Sv[j+1] = fabs(yvc[j]-yvc[j+1]);
175         }
176         Sv[0] = 0;
177         Sv[M-1] = 0;
178 }
179
```

```
180
181 // Volume of each control volume
182 void volume(double *xvc, double *yvc, int N, int M, matrix& V)
183 {
184         for(int i = 0; i<N; i++)
185         {
186                 for(int j = 0; j<M; j++)
187                 {
188                         if(i==N-1 || j==M-1)
189                         {
190                                 V[j][i] = 0;
191                         }
192                         else
193                         {
194                                 V[j][i] = fabs(xvc[i]-xvc[i-1])*fabs(yvc[j]-yvc[j-1]);
195                         }
196                 }
197         }
198 }
199
200
201 // Calculation of the constant coefficients (ae, aw, an, as, ap) of the Poisson equation (pressure)
202 void constant_coefficients (int N, int M, double *x, double *y, double *Sv, double *Sh, matrix& ae,
        matrix& aw, matrix& an, matrix& as, matrix& ap)
203 {
204         for(int i = 0; i<N; i++)
205         {
206                 for(int j = 0; j<M; j++)
207                 {
208                         if(j==M-1 && i!=0 && i!=N-1)
209                         {
210                                 ae[j][i] = 0;
211                                 aw[j][i] = 0;
212                                 an[j][i] = 0;
213                                 as[j][i] = 1;
214                                 ap[j][i] = 1;
215                         }
216                         else if(i==0 && j==0)
217                         {
218                                 ae[j][i] = 1;
219                                 aw[j][i] = 0;
220                                 an[j][i] = 1;
221                                 as[j][i] = 0;
222                                 ap[j][i] = 1;
223                         }
224                         else if(i==0 && j==M-1)
225                         {
226                                 ae[j][i] = 1;
227                                 aw[j][i] = 0;
228                                 an[j][i] = 0;
```

```
229                 as[j][i] = 1;
230                 ap[j][i] = 1;
231             }
232             else if(i==0 && j!=0 && j!=M-1)
233             {
234                 ae[j][i] = 1;
235                 aw[j][i] = 0;
236                 an[j][i] = 0;
237                 as[j][i] = 0;
238                 ap[j][i] = 1;
239             }
240             else if(i==N-1 && j==0)
241             {
242                 ae[j][i] = 0;
243                 aw[j][i] = 1;
244                 an[j][i] = 1;
245                 as[j][i] = 0;
246                 ap[j][i] = 1;
247             }
248             else if(i==N-1 && j==M-1)
249             {
250                 ae[j][i] = 0;
251                 aw[j][i] = 1;
252                 an[j][i] = 0;
253                 as[j][i] = 1;
254                 ap[j][i] = 1;
255             }
256             else if(i==N-1 && j!=0 && j!=M-1)
257             {
258                 ae[j][i] = 0;
259                 aw[j][i] = 1;
260                 an[j][i] = 0;
261                 as[j][i] = 0;
262                 ap[j][i] = 1;
263             }
264             else if(j==0 && i!=0 && i!=N-1)
265             {
266                 ae[j][i] = 0;
267                 aw[j][i] = 0;
268                 an[j][i] = 1;
269                 as[j][i] = 0;
270                 ap[j][i] = 1;
271             }
272             else
273             {
274                 ae[j][i] = Sv[j]/fabs(x[i+1]-x[i]);
275                 aw[j][i] = Sv[j]/fabs(x[i]-x[i-1]);
276                 an[j][i] = Sh[i]/fabs(y[j+1]-y[j]);
277                 as[j][i] = Sh[i]/fabs(y[j]-y[j-1]);
278                 ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i];
```

```
279                         }
280                 }
281         }
282 }
283
284
285 // Computation of the velocity in the convective term using CDS
286 double convective_term (double xf, double x2, double x3, double u2, double u3)
287 {
288         // 2 refers to node P, 3 to node E
289         double u;
290         u = u2+fabs(x2-xf)*(u3-u2)/fabs(x3-x2);
291
292         return u;
293 }
294
295
296 // Calculation of the intermediate velocities
297 void intermediate_velocities (int N, int M, float rho, float mu, float delta, double dt, double* x,
        double* y, double *xvc, double* yvc, double* Sh, double* Sv, matrix V, staggx u0, staggy v0,
        staggx Ru0, staggy Rv0, staggx &Ru, staggy &Rv, staggx &up, staggy &vp)
298 {
299         double mflowe, mfloww, mflown, mflows;
300         double ue, uw, un, us;
301
302         for(int i = 0; i<N+1; i++)
303         {
304                 for(int j = 0; j<M+2; j++)
305                 {
306                         // Mass flow terms (rho*v*S)
307                         mflowe = (rho*u0[j][i+1]+rho*u0[j][i])*Sv[j]/2;
308                         mfloww = (rho*u0[j][i-1]+rho*u0[j][i])*Sv[j]/2;
309                         mflown = (rho*v0[j][i]+rho*v0[j][i+1])*Sh[i]/2;
310                         mflows = (rho*v0[j-1][i]+rho*v0[j-1][i+1])*Sh[i]/2;
311
312
313                         // HORIZONTAL
314                         ue = convective_term (x[i+1], xvc[i], xvc[i+1], u0[j][i], u0[j][i+1]);
315                         uw = convective_term (x[i], xvc[i], xvc[i-1], u0[j][i], u0[j][i-1]);
316                         un = convective_term (yvc[j], y[j], y[j+1], u0[j][i], u0[j+1][i]);
317                         us = convective_term (yvc[j-1], y[j], y[j-1], u0[j][i], u0[j-1][i]);
318
319
320                         // R ( horizontal )
321                         if(i==0 || i==N || j==0 || j==M+1)
322                         {
323                                 Ru[j][i] = 0;
324                         }
325                         else
326                         {
```

```
327                             Ru[j][i] = (mu*(u0[j][i+1]-u0[j][i])*Sv[j]/fabs(xvc[i+1]-xvc[i])+mu*(
                                    u0[j+1][i]-u0[j][i])*Sh[i]/fabs(y[j+1]-y[j])-mu*(u0[j][i]-u0[j][i
                                    -1])*Sv[j]/fabs(xvc[i]-xvc[i-1])-mu*(u0[j][i]-u0[j-1][i])*Sh[i]/
                                    fabs(y[j]-y[j-1])-(mflowe*ue+mflown*un-mfloww*uw-mflows*us
                                    ))/V[j][i];
328                     }
329
330                     // Intermediate velocity ( horizontal )
331                     up[j][i] = u0[j][i]+dt*(1.5*Ru[j][i]-0.5*Ru0[j][i])/rho;
332             }
333     }
334
335     double ve, vw, vn, vs;
336
337     for(int i = 0; i<N+2; i++)
338     {
339             for(int j = 0; j<M+1; j++)
340             {
341                     // Mass flow terms (rho*v*S)
342                     mflowe = (rho*u0[j+1][i]+rho*u0[j][i])*Sv[j]/2;
343                     mfloww = (rho*u0[j+1][i-1]+rho*u0[j][i-1])*Sv[j]/2;
344                     mflown = (rho*v0[j][i]+rho*v0[j+1][i])*Sh[i]/2;
345                     mflows = (rho*v0[j][i]+rho*v0[j-1][i])*Sh[i]/2;
346
347
348                     // VERTICAL
349                     ve = convective_term (xvc[i], x[i], x[i+1], v0[j][i], v0[j][i+1]);
350                     vw = convective_term (xvc[i-1], x[i], x[i-1], v0[j][i], v0[j][i-1]);
351                     vn = convective_term (y[j+1], yvc[j], yvc[j+1], v0[j][i], v0[j+1][i]);
352                     vs = convective_term (y[j], yvc[j], yvc[j-1], v0[j][i], v0[j-1][i]);
353
354                     // R ( vertical )
355                     if(i==0 || i==N+1 || j==0 || j==M)
356                     {
357                             Rv[j][i] = 0;
358                     }
359                     else
360                     {
361                             Rv[j][i] = (mu*(v0[j][i+1]-v0[j][i])*Sv[j]/fabs(x[i+1]-x[i])+mu*(v0[j
                                    +1][i]-v0[j][i])*Sh[i]/fabs(yvc[j+1]-yvc[j])-mu*(v0[j][i]-v0[j][i
                                    -1])*Sv[j]/fabs(x[i]-x[i-1])-mu*(v0[j][i]-v0[j-1][i])*Sh[i]/fabs(
                                    yvc[j]-yvc[j-1])-(mflowe*ve+mflown*vn-mfloww*vw-mflows*vs))
                                    /V[j][i];
362                     }
363
364                     // Intermediate velocity ( vertical )
365                     vp[j][i] = v0[j][i]+dt*(1.5*Rv[j][i]-0.5*Rv0[j][i])/rho;
366             }
367     }
368 }
```

```
369
370
371 // Calculation of the bp coefficient of the Poisson equation (pressure)
372 void bp_coefficient (int N, int M, float rho, double dt, double* Sh, double* Sv, staggx up, staggy vp,
        matrix bp)
373 {
374         for(int i = 0; i<N; i++)
375         {
376                 for(int j = 0; j<M; j++)
377                 {
378                         if(i==0 || j==0 || i==N-1 || j==M-1)
379                         {
380                                 bp[j][i] = 0;
381                         }
382                         else
383                         {
384                                 bp[j][i] = -(rho*up[j][i]*Sv[j]+rho*vp[j][i]*Sh[i]-rho*up[j][i-1]*Sv[j
                                        ]-rho*vp[j-1][i]*Sh[i])/dt;
385                         }
386                 }
387         }
388 }
389
390
391 // Solver (using Gauss-Seidel)
392 void Gauss_Seidel (matrix ap, matrix aw, matrix ae, matrix as, matrix an, matrix bp, float fr, float
       delta, int N, int M, matrix& T)
393 {
394         double Tcalc[M][N]; // Temperature calculated in the previous iteration
395         for(int i = 0; i<N; i++)
396         {
397                 for(int j = 0; j<M; j++)
398                 {
399                         Tcalc[j][i] = T[j][i];
400                 }
401         }
402
403         double MAX = 1; // Maximum value of the difference between T and Tcalc
404         double resta = 1; // Difference between T and Tcalc
405
406         while(MAX>delta)
407         {
408
409                 // SOLVER: Gauss-Seidel
410                 for(int i = 0; i<N; i++)
411                 {
412                         for(int j = 0; j<M; j++)
413                         {
414                                 if(i==0 && j==M-1)
415                                 {
```

```
416                         T[j][i] = Tcalc[j][i]+fr*((ae[j][i]*Tcalc[j][i+1]+as[j][i]*T[j
                                -1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
417                     }
418                     else if(i==0 && j==0)
419                     {
420                         T[j][i] = Tcalc[j][i]+fr*((ae[j][i]*Tcalc[j][i+1]+an[j][i]*
                                Tcalc[j+1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
421                     }
422                     else if(i==0 && j!=0 && j!=M-1)
423                     {
424                         T[j][i] = Tcalc[j][i]+fr*((ae[j][i]*Tcalc[j][i+1]+as[j][i]*T[j
                                -1][i]+an[j][i]*Tcalc[j+1][i]+bp[j][i])/ap[j][i]-Tcalc[j
                                ][i]);
425                     }
426                     else if(i==N-1 && j==M-1)
427                     {
428                         T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+as[j][i]*T[j-1][
                                i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
429                     }
430                     else if(i==N-1 && j==0)
431                     {
432                         T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+an[j][i]*Tcalc[j
                                +1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
433                     }
434                     else if(i==N-1 && j!=0 && j!=M-1)
435                     {
436                         T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+as[j][i]*T[j-1][
                                i]+an[j][i]*Tcalc[j+1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i])
                                ;
437                     }
438                     else if(i!=0 && i!=N-1 && j==M-1)
439                     {
440                         T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+ae[j][i]*Tcalc[j
                                ][i+1]+as[j][i]*T[j-1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i])
                                ;
441                     }
442                     else if(i!=0 && i!=N-1 && j==0)
443                     {
444                         T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+ae[j][i]*Tcalc[j
                                ][i+1]+an[j][i]*Tcalc[j+1][i]+bp[j][i])/ap[j][i]-Tcalc[j
                                ][i]);
445                     }
446                     else
447                     {
448                         T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+ae[j][i]*Tcalc[j
                                ][i+1]+as[j][i]*T[j-1][i]+an[j][i]*Tcalc[j+1][i]+bp[j][i
                                ])/ap[j][i]-Tcalc[j][i]);
449                     }
450                 }
451             }
```

```
452
453                 // Comprovation
454                 MAX = 0;
455                 for(int i = 0; i<N; i++)
456                 {
457                         for(int j = 0; j<M; j++)
458                         {
459                                 resta = fabs(Tcalc[j][i]-T[j][i]);
460
461                                 if(resta>MAX)
462                                 {
463                                         MAX = resta;
464                                 }
465                         }
466                 }
467
468                 // New assignation
469                 for(int i = 0; i<N; i++)
470                 {
471                         for(int j = 0; j<M; j++)
472                         {
473                                 Tcalc[j][i] = T[j][i];
474                         }
475                 }
476         }
477 }
478
479
480 // Calculation of the velocity with the correction of pressure
481 void velocities (int N, int M, float rho, double dt, float uref, double* x, double* y, matrix p,
        staggx up, staggy vp, staggx &u, staggy &v)
482 {
483         // Horizontal velocity at n+1
484         for(int i = 0; i<N+1; i++)
485         {
486                 for(int j = 0; j<M+2; j++)
487                 {
488                         if(i==0 || i==N || j==0)
489                         {
490                                 u[j][i] = 0;
491                         }
492                         else if(j==M+1)
493                         {
494                                 u[j][i] = uref;
495                         }
496                         else
497                         {
498                                 u[j][i] = up[j][i]-dt*(p[j][i+1]-p[j][i])/(rho*fabs(x[i+1]-x[i]));
499                         }
500                 }
```

```
501         }
502
503         // Vertical velocity at n+1
504         for(int i = 0; i<N+2; i++)
505         {
506                 for(int j = 0; j<M+1; j++)
507                 {
508                         if(j==0 || j==M || i==0 || i==N+1)
509                         {
510                                 v[j][i] = 0;
511                         }
512                         else
513                         {
514                                 v[j][i] = vp[j][i]-dt*(p[j+1][i]-p[j][i])/(rho*fabs(y[j+1]-y[j]));
515                         }
516                 }
517         }
518 }
519
520
521 // Returns the minimum value
522 double min(double a, double b)
523 {
524         if(a>b)
525         {
526                 return b;
527         }
528         else
529         {
530                 return a;
531         }
532 }
533
534
535 // Returns the maximum value
536 double max(double a, double b)
537 {
538         if(a>b)
539         {
540                 return a;
541         }
542         else
543         {
544                 return b;
545         }
546 }
547
548
549 // Calculation of the proper time step (CFL condition)
550 double time_step (double dtd, double* x, double* y, staggx u, staggy v)
```

```
551 {
552         double dt;
553         double dtc = 100;
554 
555         for(int i = 1; i<N; i++)
556         {
557                 for(int j = 1; j<M+1; j++)
558                 {
559                         dtc = min(dtc, 0.35*fabs(x[i+1]-x[i])/fabs(u[j][i]));
560                 }
561         }
562         for(int i = 1; i<N+1; i++)
563         {
564                 for(int j = 1; j<M; j++)
565                 {
566                         dtc = min(dtc, 0.35*fabs(y[j+1]-y[j])/fabs(v[j][i]));
567                 }
568         }
569         dt = min(dtc, dtd);
570         return dt;
571 }
572 
573 
574 // Difference between the previous and the actual time step
575 double error (int N, int M, staggx u, staggy v, staggx u0, staggy v0)
576 {
577         double resta = 0;
578         for(int i = 0; i<N+1; i++)
579         {
580                 for(int j = 0; j<M+2; j++)
581                 {
582                         resta = max(resta, fabs(u[j][i]-u0[j][i]));
583                 }
584         }
585         for(int i = 0; i<N+2; i++)
586         {
587                 for(int j = 0; j<M+1; j++)
588                 {
589                         resta = max(resta, fabs(v[j][i]-v0[j][i]));
590                 }
591         }
592         return resta;
593 }
594 
595 
596 // Searching the index of the node closest to a given point (and the second closest)
597 void search_index (float point, double *x, int Number, int& ipoint, int& ip)
598 {
599         for(int i = 0; i<Number-1; i++)
600     {
```

```
601         if(x[i+1]-x[i]>0)
602         {
603                 if(x[i]<=point && x[i+1]>point)
604                 {
605                         if(point-x[i]<x[i+1]-point)
606                                 {
607                                         ipoint = i; //ipoint is the index of the node closest to the
                                                point we want
608                                         ip = i+1; //ip is the second node closest to it (used in
                                                interpolation )
609                                 }
610                                 else
611                                 {
612                                         ipoint = i+1;
613                                         ip = i;
614                                 }
615                         }
616                 }
617                 else
618                 {
619                         if(x[i]>point && x[i+1]<=point)
620                 {
621                         if(point-x[i+1]<x[i]-point)
622                         {
623                                 ipoint = i;
624                                 ip = i+1;
625                                 }
626                                 else
627                                 {
628                                         ipoint = i+1;
629                                         ip = i;
630                                 }
631                         }
632                 }
633         }
634
635 }
636
637
638 // Linear interpolation
639 double interpolation(float x, double T1, double T2, double x1, double x2)
640 {
641         double result;
642         result = T1+(T2-T1)*(x-x1)/(x2-x1);
643         return result;
644 }
645
646
647 // Output of the results
648 void output_files (int N, int M, float L, double* x, double* y, double* xvc, double* yvc, staggx u,
```

```
    staggy v)
649 {
650     // Horizontal velocities
651     ofstream resultats ;
652   resultats .open("Resultats.dat");
653     for(int j = M+1; j>=0; j--)
654     {
655         for(int i = 0; i<N+2; i++)
656         {
657             resultats <<x[i]<<"    "<<y[j]<<"    "<<u[j][i]<<endl;
658         }
659          resultats <<endl;
660     }
661      resultats . close();
662
663     // Vertical velocities
664     ofstream resvltats ;
665   resvltats .open("Resvltats.dat");
666     for(int j = M+1; j>=0; j--)
667     {
668         for(int i = 0; i<N+2; i++)
669         {
670             resvltats <<x[i]<<"    "<<y[j]<<"    "<<v[j][i]<<endl;
671         }
672          resvltats <<endl;
673     }
674      resvltats . close();
675
676
677     // Searching the indexes to interpolate
678     int ipoint, ip, jpoint, jp;
679     search_index (L/2, xvc, N+1, ipoint, ip);
680   search_index (L/2, yvc, M+1, jpoint, jp);
681
682     // Horizontal velocity in the central vertical line
683     ofstream resultsu ;
684   resultsu .open("u.dat");
685   for(int i = M+1; i>=0; i--)
686   {
687      resultsu <<y[i]<<"    "<<interpolation(L/2, u[i][ipoint], u[i][ip], xvc[ipoint], xvc[ip])<<
          endl;
688     }
689   resultsu . close();
690
691     // Vertical velocity in the central horizontal line
692     ofstream resultsv ;
693   resultsv .open("v.dat");
694   for(int i = N+1; i>=0; i--)
695   {
696      resultsv <<x[i]<<"    "<<interpolation(L/2, v[jpoint][i], u[jp][i], yvc[jpoint], yvc[jp])<<
```

```
                endl;
697             }
698         resultsv . close();
699 }
```