



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

1	Four materials problem	1
2	Smith-Hutton problem	18
3	Driven cavity problem	30
4	Differentially heated cavity	47
5	Burgers' equation	67
6	Square cylinder	71

1 | Four materials problem

```

1  #include<iostream>
2  #include<math.h>
3  #include<fstream>
4
5  using namespace std;
6
7  // Dimensions
8  const int M1 = 40;
9  const int M2 = 30;
10 const int M3 = 10;
11 const int N1 = 50;
12 const int N2 = 60;
13
14 // Definition of types
15 typedef double matrix[M1+M2+M3][N1+N2];
16
17
18 // FUNCTIONS
19 void horizontal_coordinates (double dx1, double dx2, double xvc [], double x []);
20 void vertical_coordinates (double dy1, double dy2, double dy3, double yvc [], double y []);
21 void volume (double *xvc, double *yvc, int N, int M, matrix& V);
22 void surface (double *yvc, int M, double Sx[]);
23 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);
24 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);
25 void search_index (float point, double *x, int Number, int &ipoint, int& ip);
26 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);
27 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
    Tleft, float Tright, float Trightant, matrix Tant, matrix rho, matrix cp, matrix lambda, matrix
    lambdaw, matrix lambdae, matrix lambdas, matrix lambdan, matrix& bp);
28 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);
29 double double_interpolation (float x, float y, double T11, double T12, double T21, double T22,

```

```

    double x1, double x2, double y1, double y2);
30 void print_matrix (matrix T, int N, int M);
31 void output_file (double* Tpoint1, double* Tpoint2, int Time, float dt);
32
33
34 int main(){
35
36     // DATA
37     // Coordinates
38     const float p [3][2] = {
39         {0.50,0.40},
40         {0.50,0.70},
41         {1.10,0.80}
42     }; // [m]
43
44     // Physical properties
45     const float rhod[4] = {1500.00,1600.00,1900.00,2500.00}; // [kg/m^3]
46     const float cpd[4] = {750.00,770.00,810.00,930.00}; // [J/(kgK)]
47     const float lamd[4] = {170.00,140.00,200.00,140.00}; // [W/(mK)]
48
49     // Boundary conditions
50     const float Tbottom = 23.00; // [C]
51     const float Qtop = 60.00; // [W/m]
52     const float Tgleft = 33.00; // [C]
53     const float alpha = 9.00; // [W/(m^2K)]
54     const float Tright0 = 8.00; // Initial temperature on the right [C]
55     const float variationright = 0.005; // Variation of the temperature on the right [s/C]
56     const float T0 = 8.00; // Initial temperature [C]
57     const float qv = 0; // Internal heat [W/m^3]
58
59     // Results (coordinates)
60     const float point [2][2] = {
61         {0.65,0.56},
62         {0.74,0.72}
63     }; // Points to be studied [m]
64
65     // Mathematical properties
66     const int Time = 5001; // Time discretization
67     const float beta = 0.5;
68     const float tfinal = 5000; // Time of the simulation
69     const float delta = 0.001; // Precision of the simulation
70     const float fr = 1.2; // Relaxation factor
71
72
73     cout<<"Program started"<<endl;
74
75     // PREVIOUS CALCULATIONS
76
77     float L1,L2,H1,H2,H3; // Dimensions
78     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,
118                    lambdan);
119
120    // INITIALIZATION
121    matrix T, Tant; // Temperature and Temperature in the previous instant of time
122    float Tright, Trightant; // Temperature on the right and Temperature on the right in the
123    previous instant of time
124    double Tpoint1[Time], Tpoint2[Time]; // Temperatures at the points that are going to be
125    studied
126    for(int i = 0; i<N1+N2; i++)
127    {
128        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,
146                             lambdae, lambdas, lambdan, ap, aw, ae, as, an);
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,
163                         Tbottom, Tleft, Tright, Trightant, Tant, rho, cp, lambda, lambdaw, lambdae,
164                         lambdas, lambdan, bp);
165
166         // SOLVER
167         Gauss_Seidel (ap, aw, ae, as, an, bp, fr, delta, N1+N2, M1+M2+M3, T);
168
169         // Assignment of the instant of time
170         for(int i = 0; i<N1+N2; i++)
171         {
172             for(int j = 0; j<M1+M2+M3; j++)
173             {
174                 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("Resultats5000.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

```

```

361         ])/lambda[j][i]);
362         lambdan[j][i] = (y[j-1]-y[j])/((y[j-1]-yvc[j])/lambda[j-1][i]+(yvc[j]-
363         y[j])/lambda[j][i]);
364         lambdae[j][i] = lambda[j][i];
365         lambdas[j][i] = (y[j]-y[j+1])/((yvc[j+1]-y[j+1])/lambda[j+1][i]+(y[j]-
366         yvc[j+1])/lambda[j][i]);
367     }
368     else if(j==0)
369     {
370         lambdaw[j][i] = (x[i]-x[i-1])/((x[i]-xvc[i])/lambda[j][i-1]+(x[i]-xvc[i]
371         ))/lambda[j][i]);
372         lambdan[j][i] = lambda[j][i];
373         lambdae[j][i] = (x[i+1]-x[i])/((x[i+1]-xvc[i+1])/lambda[j][i+1]+(xvc[i
374         +1]-x[i])/lambda[j][i]);
375         lambdas[j][i] = (y[j]-y[j+1])/((yvc[j+1]-y[j+1])/lambda[j+1][i]+(y[j]-
376         yvc[j+1])/lambda[j][i]);
377     }
378     else if(j==M-1)
379     {
380         lambdaw[j][i] = (x[i]-x[i-1])/((x[i]-xvc[i])/lambda[j][i-1]+(x[i]-xvc[i]
381         ))/lambda[j][i]);
382         lambdan[j][i] = (y[j-1]-y[j])/((y[j-1]-yvc[j])/lambda[j-1][i]+(yvc[j]-
383         y[j])/lambda[j][i]);
384         lambdae[j][i] = (x[i+1]-x[i])/((x[i+1]-xvc[i+1])/lambda[j][i+1]+(xvc[i
385         +1]-x[i])/lambda[j][i]);
386         lambdas[j][i] = lambda[j][i];
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
403                     point we want
404                 ip = i+1; //ip is the second node closest to it (used in
405                     interpolation )
406             }
407             else
408             {
409                 ipoint = i+1;
410                 ip = i;
411             }
412         }
413     else
414     {
415         if (x[i]>point && x[i+1]<=point)
416         {
417             if (point-x[i+1]<x[i]-point)
418             {
419                 ipoint = i;
420                 ip = i+1;
421             }
422             else
423             {
424                 ipoint = i+1;
425                 ip = i;
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,
435     matrix V, float dt, float beta, float alpha, matrix rho, matrix cp, matrix lambda, matrix
436     lambdaw, matrix lambdae, matrix lambdas, matrix lambdan, matrix& ap, matrix& aw, matrix& ae,
437     matrix& as, matrix& an)
438 {
439     for(int i =0; i<N1+N2; i++)
440     {
441         for(int j = 0; j<M1+M2+M3; j++)
442         {
443             if (i==0 && j==0)
444             {
445                 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*lambdaaw[j][i]*Sx[j]/(x[i]-x[i-1]);
492         as[j][i] = beta*lambdaas[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*lambdaaw[j][i]*Sx[j]/(x[i]-x[i-1]);
500         as[j][i] = 0;
501         an[j][i] = beta*lambdaan[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*lambdaaw[j][i]*Sx[j]/(x[i]-x[i-1]);
508         as[j][i] = beta*lambdaas[j][i]*Sy[i]/(y[j]-y[j+1]);
509         an[j][i] = beta*lambdaan[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+1][i]+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

```

1  #include <iostream>
2  #include <math.h>
3  #include <fstream>
4
5  using namespace std;
6
7  // Numerical parameters
8  const int N = 200;
9  const int M = 100;
10
11 typedef double matrix[M+2][N+2];
12 typedef double mface[M+1][N+1];
13
14
15 // FUNCTIONS
16 void coordinates(float x0, float xN, float dx, int N, float xvc[], float x[]);
17 void surface(float *yvc, int M, float Sv[]);
18 void volume(float *xvc, float *yvc, int N, int M, matrix& V);
19 void velocity(float *x, float *y, int N, int M, mface& u, mface& v);
20 void mass_flow(float rho, int N, int M, float *Sv, float *Sh, float *xvc, float *yvc, mface& mflowx,
    mface& mflowy);
21 void phi_inlet_outlet(float *x, float alpha, int N, double phis[]);
22 void search_index(float point, float *x, int Number, int& ipoint, int& ip);
23 double max(double a, double b);
24 double Aoperator(string method, double P);
25 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);
26 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);
27 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);
28 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]);
29 void output_matrix(int N, int M, matrix mat);
30 void output_file(matrix T, int N);
31 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,
111             mflowy, xfinal , index, phi1);
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,
116             mflowy, xfinal , index, phi2);
117
118     cout<<"Solving rho/gamma = 1000000..."<<endl;
119     gamma = rho/1000000;
120     solver(method, rho, gamma, dt, fr, delta, Sp, Sc, phi_boundary, phis, x, y, Sh, Sv, V, mflowx,
121             mflowy, xfinal , index, phi3);
122
123     cout<<endl<<"Creating an output file..."<<endl;
124     ofstream results ;
125     results .open("Resultats.dat");
126     for(int k = 0; k<11; k++)
127     {
128         results <<xfinal[k]<<" " <<phi1[k]<<" " <<phi2[k]<<" " <<phi3[k]<<"\n";
129     }

```

```
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[j]*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
230             point we want
231             ip = i+1; //ip is the second node closest to it (used in
232             interpolation )
233         }
234     else
235     {
236         ipoint = i+1;
237         ip = i;
238     }
239 }
240 else
241 {
242     if (x[i]>point && x[i+1]<=point)
243     {
244         if (point-x[i+1]<x[i]-point)
245         {
246             ipoint = i;
247             ip = i+1;
248         }
249         else
250         {
251             ipoint = i+1;
252             ip = i;
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 Aoperator(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] = 0;
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] = 0;
347         as[j][i] = 0;
348         ap[j][i] = 1;
349     }
350     else
351     {
352         De = gamma*Sh[j]/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     // SOLVER: Gauss-Seidel
428     for(int i = 0; i<N; i++)
429     {
430         for(int j = 0; j<M; j++)
431         {
432             if(i==0 && j==0)
433             {
434                 T[j][i] = Tcalc[j][i]+fr*((ae[j][i]*Tcalc[j][i+1]+as[j][i]*
435                     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
440                     -1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
441             }
442             else if(i==0 && j!=0 && j!=M-1)
443             {
444                 T[j][i] = Tcalc[j][i]+fr*((ae[j][i]*Tcalc[j][i+1]+as[j][i]*
445                     Tcalc[j+1][i]+an[j][i]*T[j-1][i]+bp[j][i])/ap[j][i]-Tcalc
446                     [j][i]);
447             }
448             else if(i==N-1 && j==0)
449             {
450                 T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+as[j][i]*Tcalc[j
451                     +1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
452             }
453             else if(i==N-1 && j==M-1)
454             {
455                 T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+as[j][i]*Tcalc[j
456                     +1][i]+an[j][i]*T[j-1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i])
457                     ;
458             }
459             else if(i!=0 && i!=N-1 && j==0)
460             {
461                 T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+ae[j][i]*Tcalc[j
462                     ][i+1]+as[j][i]*Tcalc[j+1][i]+bp[j][i])/ap[j][i]-Tcalc[j
463                     ][i]);
464             }
465             else if(i!=0 && i!=N-1 && j==M-1)

```

```

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

```

```

505 {
506     matrix phi, phi0;
507
508     for(int i = 0; i<N+2; i++)
509     {
510         for(int j = 0; j<M+2; j++)
511         {
512             phi[j][i] = 1;
513         }
514     }
515
516
517     matrix ae, aw, an, as, ap, bp;
518     constant_coefficients (N+2, M+2, method, rho, gamma, dt, Sp, x, y, Sh, Sv, V, mflowx, mflowy,
519                             ae, aw, an, as, ap);
520
521     float resta = 1;
522
523     while(resta>delta)
524     {
525         //New increment of time
526         for(int i = 0; i<N+2; i++)
527         {
528             for(int j = 0; j<M+2; j++)
529             {
530                 phi0[j][i] = phi[j][i];
531             }
532         }
533
534         bp_coefficient (N+2, M+2, rho, dt, Sc, x, phi_boundary, phis, phi0, V, bp);
535         Gauss_Seidel (ap, aw, ae, as, an, bp, fr, delta, N+2, M+2, phi);
536
537         resta = 0;
538         for(int i = 0; i<N+2; i++)
539         {
540             for(int j = 0; j<M+2; j++)
541             {
542                 resta = max(resta, fabs(phi[j][i]-phi0[j][i]));
543             }
544         }
545     }
546
547     for(int i = 0; i<11; i++)
548     {
549         phi1[i] = interpolation ( xfinal [i], phi[M+1][index[0][i]], phi[M+1][index[1][i]], x[
550             index [0][i]], x[index [1][i]] );
551     }

```

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 = 112;
9  const int M = 112;
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 initial_conditions (int N, int M, float uref, staggx& u0, staggx& Ru0, staggy& v0, staggy& Rv0);
19 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);
20 double convective_term (double xf, double x2, double x3, double u2, double u3);
21 void intermediate_velocities (int N, int M, float rho, double 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);
22 void bp_coefficient (int N, int M, float rho, double dt, double* Sh, double* Sv, staggx up, staggy vp,
    matrix bp);
23 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);
24 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);
25 double min(double a, double b);
26 double max(double a, double b);
27 double time_step (double dtd, double* x, double* y, staggx u, staggy v);
28 double error (int N, int M, staggx u, staggy v, staggx u0, staggy v0);
29 void search_index (float point, double *x, int Number, int& ipoint, int& ip);
30 double interpolation (float x, double T1, double T2, double x1, double x2);
31 void output_files (int N, int M, float L, double* x, double* y, double* xvc, double* yvc, staggx u,
    staggy v);

```



```

32
33
34 int main()
35 {
36     int Re = 10000; // Reynolds number
37     float L = 1; // Length of the cavity
38     float rho = 1; // Density
39     float uref = 1; // Reference velocity
40     double mu = rho*uref*L/Re; // Viscosity
41
42     float delta = 2e-4; // Precision of the simulation (as the Re increases it is recommended to
        use 5e-5, 1e-4, 2e-4...)
43     float fr = 1.2; // Relaxation factor
44
45     cout<<"Program started"<<endl;
46     cout<<"Re="<<Re<<endl<<endl;
47
48     // Coordinates
49     double xvc[N+1], yvc[M+1], x[N+2], y[M+2];
50     coordinates(L, N, xvc, x);
51     coordinates(L, M, yvc, y);
52
53     // Surfaces
54     double Sh[N+2], Sv[M+2];
55     matrix V;
56     surface(xvc, N+2, Sh); // Horizontal surface
57     surface(yvc, M+2, Sv); // Vertical surface
58     volume(xvc, yvc, N+2, M+2, V); // Volume
59
60
61     // Properties that are going to be calculated
62     matrix p; // Values in the nodes (pressure)
63     staggx u, u0, Ru0; // Values in the points given by the staggered meshes ( velocities )
64     staggy v, v0, Rv0;
65
66     // Inicialization
67     initial_conditions (N, M, uref, u0, Ru0, v0, Rv0);
68
69     // Calculation of the constant coefficients that are used to determine the pressure
70     matrix ae, aw, an, as, ap, bp;
71     constant_coefficients (N+2, M+2, x, y, Sv, Sh, ae, aw, an, as, ap);
72
73     // Time step (CFL condition)
74     double resta = 1;
75     double dtd = 0.2*rho*pow(x[2]-xvc[1],2)/mu;
76     double dtc = 0.35*fabs(x[2]-xvc[1])/uref;
77     double dt = min(dtd, dtc);
78
79     staggx up, Ru; // Intermediate velocities
80     staggy vp, Rv;

```

```

81
82     cout<<"Solving..."<<endl;
83     // Fractional Step Method
84     while(resta>delta)
85     {
86         // STEP 1: INTERMEDIATE VELOCITY
87         intermediate_velocities (N, M, rho, mu, delta, dt, x, y, xvc, yvc, Sh, Sv, V, u0, v0,
            Ru0, Rv0, Ru, Rv, up, vp);
88
89
90         // STEP 2: PRESSURE
91         bp_coefficient (N+2, M+2, rho, dt, Sh, Sv, up, vp, bp);
92         Gauss_Seidel (ap, aw, ae, as, an, bp, fr, delta, N+2, M+2, p);
93
94
95         // STEP 3: VELOCITY
96         velocities (N, M, rho, dt, uref, x, y, p, up, vp, u, v);
97
98
99         // STEP 4: TIME STEP
100        dt = time_step (dtd, x, y, u, v);
101
102
103        // Comprovation
104        resta = error (N, M, u, v, u0, v0);
105
106        // New time step
107        for(int i = 0; i<N+1; i++)
108        {
109            for(int j = 0; j<M+2; j++)
110            {
111                u0[j][i] = u[j][i];
112                Ru0[j][i] = Ru[j][i];
113            }
114        }
115        for(int i = 0; i<N+2; i++)
116        {
117            for(int j = 0; j<M+1; j++)
118            {
119                v0[j][i] = v[j][i];
120                Rv0[j][i] = Rv[j][i];
121            }
122        }
123    }
124
125    // Results
126    cout<<endl<<"Creating some output files..."<<endl;
127    output_files (N, M, L, x, y, xvc, yvc, u, v);
128
129    return 0;

```

```
130 }
131
132
133 // Coordinates of the control volumes (x -> nodes, xvc -> faces)
134 void coordinates(float L, int N, double xvc[], double x[])
135 {
136     double dx = L/N;
137     xvc[0] = 0;
138     x[0] = 0;
139     for(int i = 0; i<N; i++)
140     {
141         xvc[i+1] = xvc[i]+dx;
142         x[i+1] = (xvc[i+1]+xvc[i])/2;
143     }
144     x[N+1] = L;
145 }
146
147
148 // Surfaces of the control volumes
149 void surface(double *yvc, int M, double Sv[])
150 {
151     for(int j = 0; j<M-1; j++)
152     {
153         Sv[j+1] = fabs(yvc[j]-yvc[j+1]);
154     }
155     Sv[0] = 0;
156     Sv[M-1] = 0;
157 }
158
159
160 // Volume of each control volume
161 void volume(double *xvc, double *yvc, int N, int M, matrix& V)
162 {
163     for(int i = 0; i<N; i++)
164     {
165         for(int j = 0; j<M; j++)
166         {
167             if (i==N-1 || j==M-1 || i==0 || j==0)
168             {
169                 V[j][i] = 0;
170             }
171             else
172             {
173                 V[j][i] = fabs(xvc[i]-xvc[i-1])*fabs(yvc[j]-yvc[j-1]);
174             }
175         }
176     }
177 }
178
179
```

```

180 // Initial conditions of the problem
181 void initial_conditions (int N, int M, float uref, staggx& u0, staggx& Ru0, staggy& v0, staggy& Rv0)
182 {
183     for(int j = 0; j<M+2; j++)
184     {
185         for(int i = 0; i<N+1; i++)
186         {
187             if(j==M+1 && i!=0 && i!=N)
188             {
189                 u0[j][i] = uref; // Horizontal velocity at n
190             }
191             else
192             {
193                 u0[j][i] = 0; // Horizontal velocity at n
194             }
195             Ru0[j][i] = 0; // R ( horizontal ) at n-1
196         }
197     }
198     for(int j = 0; j<M+1; j++)
199     {
200         for(int i = 0; i<N+2; i++)
201         {
202             v0[j][i] = 0; // Vertical velocity at n
203             Rv0[j][i] = 0; // R ( vertical ) at n-1
204         }
205     }
206 }
207
208
209 // Calculation of the constant coefficients (ae, aw, an, as, ap) of the Poisson equation (pressure)
210 void constant_coefficients (int N, int M, double *x, double *y, double *Sv, double *Sh, matrix& ae,
211                             matrix& aw, matrix& an, matrix& as, matrix& ap)
212 {
213     for(int i = 0; i<N; i++)
214     {
215         for(int j = 0; j<M; j++)
216         {
217             if(j==M-1 && i!=0 && i!=N-1)
218             {
219                 ae[j][i] = 0;
220                 aw[j][i] = 0;
221                 an[j][i] = 0;
222                 as[j][i] = 1;
223                 ap[j][i] = 1;
224             }
225             else if(i==0 && j==0)
226             {
227                 ae[j][i] = 1;
228                 aw[j][i] = 0;
229                 an[j][i] = 1;

```

```

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

```

```

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

```

```

327
328         De = mu*Sv[j]/fabs(xvc[i+1]-xvc[i]);
329         Dw = mu*Sv[j]/fabs(xvc[i]-xvc[i-1]);
330         Dn = mu*Sh[i]/fabs(y[j+1]-y[j]);
331         Ds = mu*Sh[i]/fabs(y[j]-y[j-1]);
332
333         // R ( horizontal )
334         if (i==0 || i==N || j==0 || j==M+1)
335         {
336             Ru[j][i] = 0;
337         }
338         else
339         {
340             Ru[j][i] = (De*(u0[j][i+1]-u0[j][i])+Dn*(u0[j+1][i]-u0[j][i])-Dw*(u0[
                    j][i]-u0[j][i-1])-Ds*(u0[j][i]-u0[j-1][i])-(mflowe*ue+mflown*un
                    -mfloww*uw-mflows*us))/V[j][i];
341         }
342
343         // Intermediate velocity ( horizontal )
344         up[j][i] = u0[j][i]+dt*(1.5*Ru[j][i]-0.5*u0[j][i])/rho;
345     }
346 }
347
348 double ve, vw, vn, vs;
349
350 for(int i = 0; i<N+2; i++)
351 {
352     for(int j = 0; j<M+1; j++)
353     {
354         // Mass flow terms (rho*v*S)
355         mflowe = (rho*u0[j+1][i]+rho*u0[j][i])*Sv[j]/2;
356         mfloww = (rho*u0[j+1][i-1]+rho*u0[j][i-1])*Sv[j]/2;
357         mflown = (rho*v0[j][i]+rho*v0[j+1][i])*Sh[i]/2;
358         mflows = (rho*v0[j][i]+rho*v0[j-1][i])*Sh[i]/2;
359
360
361         // VERTICAL
362         ve = convective_term (xvc[i], x[i], x[i+1], v0[j][i], v0[j][i+1]);
363         vw = convective_term (xvc[i-1], x[i], x[i-1], v0[j][i], v0[j][i-1]);
364         vn = convective_term (y[j+1], yvc[j], yvc[j+1], v0[j][i], v0[j+1][i]);
365         vs = convective_term (y[j], yvc[j], yvc[j-1], v0[j][i], v0[j-1][i]);
366
367         De = mu*Sv[j]/fabs(x[i+1]-x[i]);
368         Dw = mu*Sv[j]/fabs(x[i]-x[i-1]);
369         Dn = mu*Sh[i]/fabs(yvc[j+1]-yvc[j]);
370         Ds = mu*Sh[i]/fabs(yvc[j]-yvc[j-1]);
371
372         // R ( vertical )
373         if (i==0 || i==N+1 || j==0 || j==M)
374         {

```

```

375         Rv[j][i] = 0;
376     }
377     else
378     {
379         Rv[j][i] = (De*(v0[j][i+1]-v0[j][i])+Dn*(v0[j+1][i]-v0[j][i])-Dw*(v0[
            j][i]-v0[j][i-1])-Ds*(v0[j][i]-v0[j-1][i])-(mflowe*ve+mflown*vn-
            mfloww*vw-mflows*vs))/V[j][i];
380     }
381
382     // Intermediate velocity ( vertical )
383     vp[j][i] = v0[j][i]+dt*(1.5*Rv[j][i]-0.5*Rv0[j][i])/rho;
384 }
385 }
386 }
387
388
389 // Calculation of the bp coefficient of the Poisson equation (pressure)
390 void bp_coefficient (int N, int M, float rho, double dt, double* Sh, double* Sv, staggx up, staggy vp,
    matrix bp)
391 {
392     for(int i = 0; i<N; i++)
393     {
394         for(int j = 0; j<M; j++)
395         {
396             if (i==0 || j==0 || i==N-1 || j==M-1)
397             {
398                 bp[j][i] = 0;
399             }
400             else
401             {
402                 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;
403             }
404         }
405     }
406 }
407
408
409 // Solver (using Gauss-Seidel)
410 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)
411 {
412     double Tcalc[M][N]; // Temperature calculated in the previous iteration
413     for(int i = 0; i<N; i++)
414     {
415         for(int j = 0; j<M; j++)
416         {
417             Tcalc[j][i] = T[j][i];
418         }
419     }

```



```

420
421 double MAX = 1; // Maximum value of the difference between T and Tcalc
422 double resta = 1; // Difference between T and Tcalc
423
424 while(MAX>delta)
425 {
426
427     // SOLVER: Gauss–Seidel
428     for(int i = 0; i<N; i++)
429     {
430         for(int j = 0; j<M; j++)
431         {
432             if(i==0 && j==M-1)
433             {
434                 T[j][i] = Tcalc[j][i]+fr*((ae[j][i]*Tcalc[j][i+1]+as[j][i]*T[j]
435                     -1)[i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
436             }
437             else if(i==0 && j==0)
438             {
439                 T[j][i] = Tcalc[j][i]+fr*((ae[j][i]*Tcalc[j][i+1]+an[j][i]*
440                     Tcalc[j+1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
441             }
442             else if(i==0 && j!=0 && j!=M-1)
443             {
444                 T[j][i] = Tcalc[j][i]+fr*((ae[j][i]*Tcalc[j][i+1]+as[j][i]*T[j]
445                     -1)[i]+an[j][i]*Tcalc[j+1][i]+bp[j][i])/ap[j][i]-Tcalc[j]
446                     [i]);
447             }
448             else if(i==N-1 && j==M-1)
449             {
450                 T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+as[j][i]*T[j-1][
451                     i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
452             }
453             else if(i==N-1 && j==0)
454             {
455                 T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+an[j][i]*Tcalc[j
456                     +1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
457             }
458             else if(i==N-1 && j!=0 && j!=M-1)
459             {
460                 T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+as[j][i]*T[j-1][
461                     i]+an[j][i]*Tcalc[j+1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i])
462                     ;
463             }
464             else if(i!=0 && i!=N-1 && j==M-1)
465             {
466                 T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+ae[j][i]*Tcalc[j
467                     ][i+1]+as[j][i]*T[j-1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i])
468                     ;
469             }
470         }
471     }
472
473     MAX = resta;
474     resta = 0;
475 }

```

```

460         else if (i!=0 && i!=N-1 && j==0)
461         {
462             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]);
463         }
464         else
465         {
466             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]);
467         }
468     }
469 }
470
471 // Comprovation
472 MAX = 0;
473 for(int i = 0; i<N; i++)
474 {
475     for(int j = 0; j<M; j++)
476     {
477         resta = fabs(Tcalc[j][i]-T[j][i]);
478
479         if (resta>MAX)
480         {
481             MAX = resta;
482         }
483     }
484 }
485
486 // New assignation
487 for(int i = 0; i<N; i++)
488 {
489     for(int j = 0; j<M; j++)
490     {
491         Tcalc[j][i] = T[j][i];
492     }
493 }
494 }
495 }
496
497
498 // Calculation of the velocity with the correction of pressure
499 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)
500 {
501     // Horizontal velocity at n+1
502     for(int i = 0; i<N+1; i++)
503     {
504         for(int j = 0; j<M+2; j++)

```

```

505         {
506             if (i==0 || i==N || j==0)
507             {
508                 u[j][i] = 0;
509             }
510             else if (j==M+1)
511             {
512                 u[j][i] = uref;
513             }
514             else
515             {
516                 u[j][i] = up[j][i] - dt*(p[j][i+1] - p[j][i]) / (rho*fabs(x[i+1] - x[i]));
517             }
518         }
519     }
520
521     // Vertical velocity at n+1
522     for(int i = 0; i < N+2; i++)
523     {
524         for(int j = 0; j < M+1; j++)
525         {
526             if (j==0 || j==M || i==0 || i==N+1)
527             {
528                 v[j][i] = 0;
529             }
530             else
531             {
532                 v[j][i] = vp[j][i] - dt*(p[j+1][i] - p[j][i]) / (rho*fabs(y[j+1] - y[j]));
533             }
534         }
535     }
536 }
537
538
539 // Returns the minimum value
540 double min(double a, double b)
541 {
542     if (a > b)
543     {
544         return b;
545     }
546     else
547     {
548         return a;
549     }
550 }
551
552
553 // Returns the maximum value
554 double max(double a, double b)

```

```

555 {
556     if(a>b)
557     {
558         return a;
559     }
560     else
561     {
562         return b;
563     }
564 }
565
566
567 // Calculation of the proper time step (CFL condition)
568 double time_step (double dtd, double* x, double* y, staggx u, staggy v)
569 {
570     double dt;
571     double dtc = 100;
572
573     for(int i = 1; i<N; i++)
574     {
575         for(int j = 1; j<M+1; j++)
576         {
577             dtc = min(dtc, 0.35*fabs(x[i+1]-x[i])/fabs(u[j][i]));
578         }
579     }
580     for(int i = 1; i<N+1; i++)
581     {
582         for(int j = 1; j<M; j++)
583         {
584             dtc = min(dtc, 0.35*fabs(y[j+1]-y[j])/fabs(v[j][i]));
585         }
586     }
587     dt = min(dtc, dtd);
588     return dt;
589 }
590
591
592 // Difference between the previous and the actual time step
593 double error (int N, int M, staggx u, staggy v, staggx u0, staggy v0)
594 {
595     double resta = 0;
596     for(int i = 0; i<N+1; i++)
597     {
598         for(int j = 0; j<M+2; j++)
599         {
600             resta = max(resta, fabs(u[j][i]-u0[j][i]));
601         }
602     }
603     for(int i = 0; i<N+2; i++)
604     {

```

```

605         for(int j = 0; j<M+1; j++)
606         {
607             resta = max(resta, fabs(v[j][i]-v0[j][i]));
608         }
609     }
610     return resta;
611 }
612
613
614 // Searching the index of the node closest to a given point (and the second closest)
615 void search_index (float point, double *x, int Number, int& ipoint, int& ip)
616 {
617     for(int i = 0; i<Number-1; i++)
618     {
619         if(x[i+1]-x[i]>0)
620         {
621             if(x[i]<=point && x[i+1]>point)
622             {
623                 if (point-x[i]<x[i+1]-point)
624                 {
625                     ipoint = i; //ipoint is the index of the node closest to the
626                             point we want
627                     ip = i+1; //ip is the second node closest to it (used in
628                             interpolation )
629                 }
630                 else
631                 {
632                     ipoint = i+1;
633                     ip = i;
634                 }
635             }
636         }
637         else
638         {
639             if(x[i]>point && x[i+1]<=point)
640             {
641                 if (point-x[i+1]<x[i]-point)
642                 {
643                     ipoint = i;
644                     ip = i+1;
645                 }
646                 else
647                 {
648                     ipoint = i+1;
649                     ip = i;
650                 }
651             }
652         }
653     }
654 }

```

```

653 }
654
655
656 // Linear interpolation
657 double interpolation (float x, double T1, double T2, double x1, double x2)
658 {
659     double result ;
660     result = T1+(T2-T1)*(x-x1)/(x2-x1);
661     return result ;
662 }
663
664
665 // Output of the results
666 void output_files (int N, int M, float L, double* x, double* y, double* xvc, double* yvc, staggx u,
667     staggy v)
668 {
669     // Horizontal coordinates
670     ofstream xx;
671     xx.open("x.dat");
672     for(int i = 0; i<N+2; i++)
673     {
674         xx<<x[i]<<endl;
675     }
676     xx.close();
677
678     // Vertical coordinates
679     ofstream yy;
680     yy.open("y.dat");
681     for(int j = M+1; j>=0; j--)
682     {
683         yy<<y[j]<<endl;
684     }
685     yy.close();
686
687     // Horizontal velocities
688     ofstream resultats ;
689     resultats .open("Resultats.dat");
690     for(int i = 0; i<N+1; i++)
691     {
692         for(int j = 0; j<M+2; j++)
693         {
694             resultats <<xvc[i]<<" " <<y[j]<<" " <<u[j][i]<<endl;
695         }
696         resultats <<endl;
697     }
698     resultats .close();
699
700     // Vertical velocities
701     ofstream resltats ;
702     resltats .open("Resltats.dat");

```

```

702     for(int i = 0; i<N+2; i++)
703     {
704         for(int j = 0; j<M+1; j++)
705         {
706             resvtats <<x[i]<<"    "<<yvc[j]<<"    "<<v[j][i]<<endl;
707         }
708         resvtats <<endl;
709     }
710     resvlt . close();
711
712     // Matrix of horizontal velocities
713     ofstream result ;
714     result .open("Matrixu.dat");
715     for(int j = M+1; j>=0; j--)
716     {
717         for(int i = 0; i<N+2; i++)
718         {
719             if(i==0 || i==N+1)
720             {
721                 result <<u[j][i]<<"    ";
722             }
723             else
724             {
725                 result <<convective_term (x[i], xvc[i-1], xvc[i], u[j][i-1], u[j][i])
726                     <<" ";
727             }
728             result <<endl;
729         }
730         result . close();
731
732         // Matrix of vertical velocities
733         ofstream resvlt ;
734         resvlt .open("Matrixv.dat");
735         for(int j = M+1; j>=0; j--)
736         {
737             for(int i = 0; i<N+2; i++)
738             {
739                 if(j==0 && j==M+1)
740                 {
741                     resvlt <<v[j][i]<<"    ";
742                 }
743                 else
744                 {
745                     resvlt <<convective_term (y[j], yvc[j-1], yvc[j], v[j-1][i], v[j][i])
746                         <<" ";
747                 }
748             }
749             resvlt <<endl;
750         }
751     }

```

```
750     resvlt . close();
751
752
753     // Searching the indexes to interpolate
754     int ipoint, ip, jpoint, jp;
755     search_index (L/2, xvc, N+1, ipoint, ip);
756     search_index (L/2, yvc, M+1, jpoint, jp);
757
758     // Horizontal velocity in the central vertical line
759     ofstream resultsu ;
760     resultsu .open("u.dat");
761     for(int i = M+1; i>=0; i--)
762     {
763         resultsu <<y[i]<<"      "<<interpolation(L/2, u[i][ ipoint ], u[i][ ip ], xvc[ ipoint ], xvc[ ip])<<
764         endl;
765     }
766     resultsu . close();
767
768     // Vertical velocity in the central horizontal line
769     ofstream resultsv ;
770     resultsv .open("v.dat");
771     for(int i = N+1; i>=0; i--)
772     {
773         resultsv <<x[i]<<"      "<<interpolation(L/2, v[ jpoint ][ i ], u[ jp ][ i ], yvc[ jpoint ], yvc[ jp])<<
774         endl;
775     }
776     resultsv . close();
777 }
```


4 | Differentially heated cavity

```

1  #include<iostream>
2  #include<math.h>
3  #include<fstream>
4
5  using namespace std;
6
7  // Numerical parameters
8  const int N = 50;
9  const int M = 50;
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 initial_conditions (int N, int M, staggx& u0, staggx& Ru0, staggy& v0, staggy& Rv0, matrix& T0);
19 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);
20 void temperature_coefficients (int N, int M, double dt, double* x, double* y, double* Sv, double* Sh,
    matrix V, staggx u, staggy v, matrix T0, matrix &aTe, matrix &aTw, matrix &aTn, matrix &aTs,
    matrix &aTp, matrix &bTp);
21 double convective_term (double xf, double x2, double x3, double u2, double u3);
22 void intermediate_velocities (int N, int M, float Pr, int Ra, double dt, double* x, double* y, double
    *xvc, double* yvc, double* Sh, double* Sv, matrix V, matrix T0, staggx u0, staggy v0, staggx Ru0,
    staggy Rv0, staggx &Ru, staggy &Rv, staggx &up, staggy &vp);
23 void bp_coefficient (int N, int M, double dt, double* Sh, double* Sv, staggx up, staggy vp, matrix bp)
    ;
24 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);
25 void velocities (int N, int M, double dt, double* x, double* y, matrix p, staggx up, staggy vp, staggx
    &u, staggy &v);
26 double min(double a, double b);
27 double max(double a, double b);
28 double time_step (double dtd, double* x, double* y, staggx u, staggy v);
29 double error (int N, int M, staggx u, staggy v, staggx u0, staggy v0, matrix T, matrix T0);
30 void heat_flux(int N, int M, double* x, staggx u, matrix T, matrix Q);

```

```

31 void Nusselt(int N, int M, double* x, double* yvc, matrix Q, double Nu[]);
32 void maximum_planes (int N, int M, double* x, double* y, staggx u, staggy v);
33 void output_files (int N, int M, float L, double* x, double* y, double* xvc, double* yvc, staggx u,
    staggy v, matrix T, double* Nu);
34
35
36 int main()
37 {
38     float Pr = 0.71; // Prandtl number
39     int Ra = 1e6; // Rayleigh number
40     float L = 1; // Length of the cavity
41
42     float delta = 1e-4; // Precision of the simulation
43     float fr = 1.2; // Relaxation factor
44
45     cout<<"Program started"<<endl;
46     cout<<"Pr="<<Pr<<endl<<endl;
47     cout<<"Ra="<<Ra<<endl<<endl;
48
49     // Coordinates
50     double xvc[N+1], yvc[M+1], x[N+2], y[M+2];
51     coordinates(L, N, xvc, x);
52     coordinates(L, M, yvc, y);
53
54     // Surfaces
55     double Sh[N+2], Sv[M+2];
56     matrix V;
57     surface(xvc, N+2, Sh); // Horizontal surface
58     surface(yvc, M+2, Sv); // Vertical surface
59     volume(xvc, yvc, N+2, M+2, V); // Volume
60
61
62     // Properties that are going to be calculated
63     matrix p, T, T0, Q; // Values in the nodes (pressure)
64     double Nu[N+2]; // Nusselt number
65     staggx u, u0, Ru0; // Values in the points given by the staggered meshes ( velocities )
66     staggy v, v0, Rv0;
67
68     // Inicialization
69     initial_conditions (N, M, u0, Ru0, v0, Rv0, T0);
70
71     matrix aTe, aTw, aTn, aTs, aTp, bTp;
72
73     // Calculation of the constant coefficients that are used to determine the pressure
74     matrix ae, aw, an, as, ap, bp;
75     constant_coefficients (N+2, M+2, x, y, Sv, Sh, ae, aw, an, as, ap);
76
77     // Time step (CFL condition)
78     double resta = 1;
79     double dtd = 0.2*pow(x[2]-xvc[1],2)/Pr;

```

```

80     double dtc = 0.35*fabs(x[2]-xvc[1]);
81     double dt = min(dtd, dtc);
82
83     staggx up, Ru; // Intermediate velocities
84     staggy vp, Rv;
85
86     cout<<"Solving..."<<endl;
87     // Fractional Step Method
88     while(resta>delta)
89     {
90         // STEP 1: INTERMEDIATE VELOCITY
91         intermediate_velocities (N, M, Pr, Ra, dt, x, y, xvc, yvc, Sh, Sv, V, T0, u0, v0, Ru0,
92                                   Rv0, Ru, Rv, up, vp);
93
94         // STEP 2: PRESSURE
95         bp_coefficient (N+2, M+2, dt, Sh, Sv, up, vp, bp);
96         Gauss_Seidel (ap, aw, ae, as, an, bp, fr, delta, N+2, M+2, p);
97
98         // STEP 3: VELOCITY
99         velocities (N, M, dt, x, y, p, up, vp, u, v);
100
101
102         // STEP 4: TEMPERATURE
103         temperature_coefficients (N, M, dt, x, y, Sv, Sh, V, u, v, T0, aTe, aTw, aTn, aTs, aTp,
104                                   bTp);
105         Gauss_Seidel (aTp, aTw, aTe, aTs, aTn, bTp, fr, delta, N+2, M+2, T);
106
107         // STEP 5: TIME STEP
108         dt = time_step (dtd, x, y, u, v);
109
110
111         // Comprovation
112         resta = error (N, M, u, v, u0, v0, T, T0);
113
114         // New time step
115         for(int i = 0; i<N+1; i++)
116         {
117             for(int j = 0; j<M+2; j++)
118             {
119                 u0[j][i] = u[j][i];
120                 Ru0[j][i] = Ru[j][i];
121             }
122         }
123         for(int i = 0; i<N+2; i++)
124         {
125             for(int j = 0; j<M+1; j++)
126             {
127

```

```

128             v0[j][i] = v[j][i];
129             Rv0[j][i] = Rv[j][i];
130         }
131     }
132     for(int j = 0; j<M+2; j++)
133     {
134         for(int i = 0; i<N+2; i++)
135         {
136             T0[j][i] = T[j][i];
137         }
138     }
139 }
140
141 // Results
142 heat_flux(N, M, x, u, T, Q);
143 Nusselt(N, M, x, y, Q, Nu);
144 cout<<endl<<"Creating some output files..."<<endl;
145 output_files (N, M, L, x, y, xvc, yvc, u, v, T, Nu);
146 maximum_planes (N, M, x, y, u, v);
147
148 return 0;
149 }
150
151
152 // Coordinates of the control volumes (x -> nodes, xvc -> faces)
153 void coordinates(float L, int N, double xvc[], double x[])
154 {
155     double dx = L/N;
156     xvc[0] = 0;
157     x[0] = 0;
158     for(int i = 0; i<N; i++)
159     {
160         xvc[i+1] = xvc[i]+dx;
161         x[i+1] = (xvc[i+1]+xvc[i])/2;
162     }
163     x[N+1] = L;
164 }
165
166
167 // Surfaces of the control volumes
168 void surface(double *yvc, int M, double Sv[])
169 {
170     for(int j = 0; j<M-1; j++)
171     {
172         Sv[j+1] = fabs(yvc[j]-yvc[j+1]);
173     }
174     Sv[0] = 0;
175     Sv[M-1] = 0;
176 }
177

```

```

178
179 // Volume of each control volume
180 void volume(double *xvc, double *yvc, int N, int M, matrix& V)
181 {
182     for(int i = 0; i<N; i++)
183     {
184         for(int j = 0; j<M; j++)
185         {
186             if(i==N-1 || j==M-1)
187             {
188                 V[j][i] = 0;
189             }
190             else
191             {
192                 V[j][i] = fabs(xvc[i]-xvc[i-1])*fabs(yvc[j]-yvc[j-1]);
193             }
194         }
195     }
196 }
197
198
199 // Initial conditions
200 void initial_conditions (int N, int M, staggx& u0, staggx& Ru0, staggy& v0, staggy& Rv0, matrix& T0)
201 {
202     for(int j = 0; j<M+2; j++)
203     {
204         for(int i = 0; i<N+1; i++)
205         {
206             u0[j][i] = 0; // Horizontal velocity at n
207             Ru0[j][i] = 0; // R ( horizontal ) at n-1
208         }
209     }
210     for(int j = 0; j<M+1; j++)
211     {
212         for(int i = 0; i<N+2; i++)
213         {
214             v0[j][i] = 0; // Vertical velocity at n
215             Rv0[j][i] = 0; // R ( vertical ) at n-1
216         }
217     }
218     for(int j = 0; j<M+2; j++)
219     {
220         for(int i = 0; i<N+2; i++)
221         {
222             if(i==0)
223             {
224                 T0[j][i] = 1;
225             }
226             else
227             {

```

```

228         T0[j][i] = 0;
229     }
230 }
231 }
232 }
233
234
235 // Calculation of the constant coefficients (ae, aw, an, as, ap) of the Poisson equation (pressure)
236 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)
237 {
238     for(int i = 0; i<N; i++)
239     {
240         for(int j = 0; j<M; j++)
241         {
242             if(j==M-1 && i!=0 && i!=N-1)
243             {
244                 ae[j][i] = 0;
245                 aw[j][i] = 0;
246                 an[j][i] = 0;
247                 as[j][i] = 1;
248                 ap[j][i] = 1;
249             }
250             else if(i==0 && j==0)
251             {
252                 ae[j][i] = 1;
253                 aw[j][i] = 0;
254                 an[j][i] = 1;
255                 as[j][i] = 0;
256                 ap[j][i] = 1;
257             }
258             else if(i==0 && j==M-1)
259             {
260                 ae[j][i] = 1;
261                 aw[j][i] = 0;
262                 an[j][i] = 0;
263                 as[j][i] = 1;
264                 ap[j][i] = 1;
265             }
266             else if(i==0 && j!=0 && j!=M-1)
267             {
268                 ae[j][i] = 1;
269                 aw[j][i] = 0;
270                 an[j][i] = 0;
271                 as[j][i] = 0;
272                 ap[j][i] = 1;
273             }
274             else if(i==N-1 && j==0)
275             {
276                 ae[j][i] = 0;

```

```

277         aw[j][i] = 1;
278         an[j][i] = 1;
279         as[j][i] = 0;
280         ap[j][i] = 1;
281     }
282     else if(i==N-1 && j==M-1)
283     {
284         ae[j][i] = 0;
285         aw[j][i] = 1;
286         an[j][i] = 0;
287         as[j][i] = 1;
288         ap[j][i] = 1;
289     }
290     else if(i==N-1 && j!=0 && j!=M-1)
291     {
292         ae[j][i] = 0;
293         aw[j][i] = 1;
294         an[j][i] = 0;
295         as[j][i] = 0;
296         ap[j][i] = 1;
297     }
298     else if(j==0 && i!=0 && i!=N-1)
299     {
300         ae[j][i] = 0;
301         aw[j][i] = 0;
302         an[j][i] = 1;
303         as[j][i] = 0;
304         ap[j][i] = 1;
305     }
306     else
307     {
308         ae[j][i] = Sv[j]/fabs(x[i+1]-x[i]);
309         aw[j][i] = Sv[j]/fabs(x[i]-x[i-1]);
310         an[j][i] = Sh[i]/fabs(y[j+1]-y[j]);
311         as[j][i] = Sh[i]/fabs(y[j]-y[j-1]);
312         ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i];
313     }
314 }
315 }
316 }
317
318
319 // Coefficients used to calculate the temperature
320 void temperature_coefficients(int N, int M, double dt, double* x, double* y, double* Sv, double* Sh,
    matrix V, staggx u, staggy v, matrix T0, matrix &aTe, matrix &aTw, matrix &aTn, matrix &aTs,
    matrix &aTp, matrix &bTp)
321 {
322     double Fe, Fw, Fn, Fs;
323     double De, Dw, Dn, Ds;
324

```

```

325     for(int j = 0; j<M+2; j++)
326     {
327         for(int i = 0; i<N+2; i++)
328         {
329             // Mass flow terms (v*S)
330             Fe = u[j][i]*Sv[j];
331             Fw = u[j][i-1]*Sv[j];
332             Fn = v[j][i]*Sh[i];
333             Fs = v[j-1][i]*Sh[i];
334
335             // Areas and distances
336             De = Sv[j]/fabs(x[i+1]-x[i]);
337             Dw = Sv[j]/fabs(x[i]-x[i-1]);
338             Dn = Sh[i]/fabs(y[j+1]-y[j]);
339             Ds = Sh[i]/fabs(y[j+1]-y[j]);
340
341             if(i==0)
342             {
343                 aTe[j][i] = 0;
344                 aTw[j][i] = 0;
345                 aTn[j][i] = 0;
346                 aTs[j][i] = 0;
347                 aTp[j][i] = 1;
348                 bTp[j][i] = 1;
349             }
350             else if(i==N+1)
351             {
352                 aTe[j][i] = 0;
353                 aTw[j][i] = 0;
354                 aTn[j][i] = 0;
355                 aTs[j][i] = 0;
356                 aTp[j][i] = 1;
357                 bTp[j][i] = 0;
358             }
359             else if(j==0 && i!=0 && i!=N+1)
360             {
361                 aTe[j][i] = 0;
362                 aTw[j][i] = 0;
363                 aTn[j][i] = 1;
364                 aTs[j][i] = 0;
365                 aTp[j][i] = 1;
366                 bTp[j][i] = 0;
367             }
368             else if(j==M+1 && i!=0 && i!=N+1)
369             {
370                 aTe[j][i] = 0;
371                 aTw[j][i] = 0;
372                 aTn[j][i] = 0;
373                 aTs[j][i] = 1;
374                 aTp[j][i] = 1;

```



```

375         bTp[j][i] = 0;
376     }
377     else
378     {
379         aTe[j][i] = De-0.5*Fe;
380         aTw[j][i] = Dw+0.5*Fw;
381         aTn[j][i] = Dn-0.5*Fn;
382         aTs[j][i] = Ds+0.5*Fs;
383         aTp[j][i] = aTe[j][i]+aTw[j][i]+aTn[j][i]+aTs[j][i]+V[j][i]/dt;
384         bTp[j][i] = T0[j][i]*V[j][i]/dt;
385     }
386 }
387 }
388 }
389
390
391 // Computation of the velocity in the convective term using CDS
392 double convective_term (double xf, double x2, double x3, double u2, double u3)
393 {
394     // 2 refers to node P, 3 to node E
395     double u;
396     u = u2+fabs(x2-xf)*(u3-u2)/fabs(x3-x2);
397
398     return u;
399 }
400
401
402 // Calculation of the intermediate velocities
403 void intermediate_velocities (int N, int M, float Pr, int Ra, double dt, double* x, double* y, double
    *xvc, double* yvc, double* Sh, double* Sv, matrix V, matrix T0, staggy u0, staggy v0, staggy Ru0,
    staggy Rv0, staggy &Ru, staggy &Rv, staggy &up, staggy &vp)
404 {
405     double mflowe, mfloww, mflown, mflows;
406     double ue, uw, un, us;
407
408     for(int i = 0; i<N+1; i++)
409     {
410         for(int j = 0; j<M+2; j++)
411         {
412             // Mass flow terms (v*S)
413             mflowe = (u0[j][i+1]+u0[j][i])*Sv[j]/2;
414             mfloww = (u0[j][i-1]+u0[j][i])*Sv[j]/2;
415             mflown = (v0[j][i]+v0[j][i+1])*Sh[i]/2;
416             mflows = (v0[j-1][i]+v0[j-1][i+1])*Sh[i]/2;
417
418
419             // HORIZONTAL
420             ue = convective_term (x[i+1], xvc[i], xvc[i+1], u0[j][i], u0[j][i+1]);
421             uw = convective_term (x[i], xvc[i], xvc[i-1], u0[j][i], u0[j][i-1]);
422             un = convective_term (yvc[j], y[j], y[j+1], u0[j][i], u0[j+1][i]);

```

```

423         us = convective_term (yvc[j-1], y[j], y[j-1], u0[j][i], u0[j-1][i]);
424
425
426         // R ( horizontal )
427         if (i==0 || i==N || j==0 || j==M+1)
428         {
429             Ru[j][i] = 0;
430         }
431         else
432         {
433             Ru[j][i] = (Pr*(u0[j][i+1]-u0[j][i])*Sv[j]/fabs(xvc[i+1]-xvc[i])+Pr*(
                        u0[j+1][i]-u0[j][i])*Sh[i]/fabs(y[j+1]-y[j])-Pr*(u0[j][i]-u0[j][i-1])*Sv[j]/
                        fabs(xvc[i]-xvc[i-1])-Pr*(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];
434         }
435
436         // Intermediate velocity ( horizontal )
437         up[j][i] = u0[j][i]+dt*(1.5*Ru[j][i]-0.5*Ru0[j][i]);
438     }
439 }
440
441 double ve, vw, vn, vs;
442
443 for(int i = 0; i<N+2; i++)
444 {
445     for(int j = 0; j<M+1; j++)
446     {
447         // Mass flow terms (v*S)
448         mflowe = (u0[j+1][i]+u0[j][i])*Sv[j]/2;
449         mfloww = (u0[j+1][i-1]+u0[j][i-1])*Sv[j]/2;
450         mflown = (v0[j][i]+v0[j+1][i])*Sh[i]/2;
451         mflows = (v0[j][i]+v0[j-1][i])*Sh[i]/2;
452
453
454         // VERTICAL
455         ve = convective_term (xvc[i], x[i], x[i+1], v0[j][i], v0[j][i+1]);
456         vw = convective_term (xvc[i-1], x[i], x[i-1], v0[j][i], v0[j][i-1]);
457         vn = convective_term (y[j+1], yvc[j], yvc[j+1], v0[j][i], v0[j+1][i]);
458         vs = convective_term (y[j], yvc[j], yvc[j-1], v0[j][i], v0[j-1][i]);
459
460         // R ( vertical )
461         if (i==0 || i==N+1 || j==0 || j==M)
462         {
463             Rv[j][i] = 0;
464         }
465         else
466         {
467             Rv[j][i] = (Pr*(v0[j][i+1]-v0[j][i])*Sv[j]/fabs(x[i+1]-x[i])+Pr*(v0[j
                        +1][i]-v0[j][i])*Sh[i]/fabs(yvc[j+1]-yvc[j])-Pr*(v0[j][i]-v0[j][i-1])*Sv[j]/
                        fabs(xvc[i]-xvc[i-1])-Pr*(v0[j][i]-v0[j-1][i])*Sh[i]/
                        fabs(y[j]-y[j-1])-(mflowe*ue+mflown*un-mfloww*uw-mflows*us
                        ))/V[j][i];

```

```

-1])*Sv[j]/fabs(x[i]-x[i-1])-Pr*(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]+Pr*Ra*(T0[j][i]+T0[j+1][i])/2;
468     }
469
470     // Intermediate velocity ( vertical )
471     vp[j][i] = v0[j][i]+dt*(1.5*Rv[j][i]-0.5*Rv0[j][i]);
472 }
473 }
474 }
475
476
477 // Calculation of the bp coefficient of the Poisson equation ( pressure )
478 void bp_coefficient (int N, int M, double dt, double* Sh, double* Sv, staggx up, staggy vp, matrix bp)
479 {
480     for(int i = 0; i<N; i++)
481     {
482         for(int j = 0; j<M; j++)
483         {
484             if(i==0 || j==0 || i==N-1 || j==M-1)
485             {
486                 bp[j][i] = 0;
487             }
488             else
489             {
490                 bp[j][i] = -(up[j][i]*Sv[j]+vp[j][i]*Sh[i]-up[j][i-1]*Sv[j]-vp[j-1][i]
491                             *Sh[i])/dt;
492             }
493         }
494     }
495
496
497 // Solver (using Gauss-Seidel)
498 void Gauss_Seidel (matrix ap, matrix aw, matrix ae, matrix as, matrix an, matrix bp, float fr, float
499 delta, int N, int M, matrix& T)
500 {
501     double Tcalc[M][N]; // Temperature calculated in the previous iteration
502     for(int i = 0; i<N; i++)
503     {
504         for(int j = 0; j<M; j++)
505         {
506             Tcalc[j][i] = T[j][i];
507         }
508     }
509
510     double MAX = 1; // Maximum value of the difference between T and Tcalc
511     double resta = 1; // Difference between T and Tcalc
512
513     while(MAX>delta)

```

```

513 {
514
515 // SOLVER: Gauss–Seidel
516 for(int i = 0; i<N; i++)
517 {
518     for(int j = 0; j<M; j++)
519     {
520         if(i==0 && j==M-1)
521         {
522             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]);
523         }
524         else if(i==0 && j==0)
525         {
526             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]);
527         }
528         else if(i==0 && j!=0 && j!=M-1)
529         {
530             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]);
531         }
532         else if(i==N-1 && j==M-1)
533         {
534             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]);
535         }
536         else if(i==N-1 && j==0)
537         {
538             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]);
539         }
540         else if(i==N-1 && j!=0 && j!=M-1)
541         {
542             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]);
543         }
544         else if(i!=0 && i!=N-1 && j==M-1)
545         {
546             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]);
547         }
548         else if(i!=0 && i!=N-1 && j==0)
549         {
550             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]);

```

```

551         }
552         else
553         {
554             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]);
555         }
556     }
557 }
558
559 // Comprovation
560 MAX = 0;
561 for(int i = 0; i < N; i++)
562 {
563     for(int j = 0; j < M; j++)
564     {
565         resta = fabs(Tcalc[j][i] - T[j][i]);
566
567         if (resta > MAX)
568         {
569             MAX = resta;
570         }
571     }
572 }
573
574 // New assignation
575 for(int i = 0; i < N; i++)
576 {
577     for(int j = 0; j < M; j++)
578     {
579         Tcalc[j][i] = T[j][i];
580     }
581 }
582 }
583 }
584
585
586 // Calculation of the velocity with the pressure correction
587 void velocities (int N, int M, double dt, double* x, double* y, matrix p, staggx up, staggy vp, staggx
588 &u, staggy &v)
589 {
590     // Horizontal velocity at n+1
591     for(int i = 0; i < N+1; i++)
592     {
593         for(int j = 0; j < M+2; j++)
594         {
595             if (i==0 || i==N || j==0 || j==M+1)
596             {
597                 u[j][i] = 0;
598             }
599         }
600     }

```

```

598         else
599         {
600             u[j][i] = up[j][i] - dt*(p[j][i+1] - p[j][i]) / (fabs(x[i+1] - x[i]));
601         }
602     }
603 }
604
605 // Vertical velocity at n+1
606 for(int i = 0; i < N+2; i++)
607 {
608     for(int j = 0; j < M+1; j++)
609     {
610         if(j==0 || j==M || i==0 || i==N+1)
611         {
612             v[j][i] = 0;
613         }
614         else
615         {
616             v[j][i] = vp[j][i] - dt*(p[j+1][i] - p[j][i]) / (fabs(y[j+1] - y[j]));
617         }
618     }
619 }
620 }
621
622
623 // Returns the minimum value
624 double min(double a, double b)
625 {
626     if(a > b)
627     {
628         return b;
629     }
630     else
631     {
632         return a;
633     }
634 }
635
636
637 // Returns the maximum value
638 double max(double a, double b)
639 {
640     if(a > b)
641     {
642         return a;
643     }
644     else
645     {
646         return b;
647     }

```

```

648 }
649
650
651 // Calculation of the proper time step (CFL condition)
652 double time_step (double dtd, double* x, double* y, staggx u, staggy v)
653 {
654     double dt;
655     double dtc = 100;
656
657     for(int i = 1; i<N; i++)
658     {
659         for(int j = 1; j<M+1; j++)
660         {
661             dtc = min(dtc, 0.35*fabs(x[i+1]-x[i])/fabs(u[j][i]));
662         }
663     }
664     for(int i = 1; i<N+1; i++)
665     {
666         for(int j = 1; j<M; j++)
667         {
668             dtc = min(dtc, 0.35*fabs(y[j+1]-y[j])/fabs(v[j][i]));
669         }
670     }
671     dt = min(dtc, dtd);
672     return dt;
673 }
674
675
676 // Difference between the previous and the actual time step
677 double error (int N, int M, staggx u, staggy v, staggx u0, staggy v0, matrix T, matrix T0)
678 {
679     double resta = 0;
680     for(int i = 0; i<N+1; i++)
681     {
682         for(int j = 0; j<M+2; j++)
683         {
684             resta = max(resta, fabs(u[j][i]-u0[j][i]));
685         }
686     }
687     for(int i = 0; i<N+2; i++)
688     {
689         for(int j = 0; j<M+1; j++)
690         {
691             resta = max(resta, fabs(v[j][i]-v0[j][i]));
692         }
693     }
694     for(int j = 0; j<M+2; j++)
695     {
696         for (int i = 0; i<N+2; i++)
697     {

```

```

698         resta = max(resta, fabs(T[j][i]-T0[j][i]));
699     }
700 }
701
702 cout<<resta<<endl;
703 return resta;
704 }
705
706
707 // Heat flux in the horizontal direction at any point in the cavity
708 void heat_flux(int N, int M, double* x, staggx u, matrix T, matrix Q)
709 {
710     for(int j = 0; j<M+2; j++)
711     {
712         for(int i = 0; i<N+1; i++)
713         {
714             if(i==0)
715             {
716                 Q[j][i] = u[j][i]*T[j][i]-(T[j][i+1]-T[j][i])/fabs(x[i+1]-x[i]);
717             }
718             else if(i==N)
719             {
720                 Q[j][i] = Q[j][0];
721             }
722             else
723             {
724                 Q[j][i] = 0.5*u[j][i]*(T[j][i]+T[j][i+1])-(T[j][i+1]-T[j][i])/fabs(x[i
725                     +1]-x[i]);
726             }
727         }
728     }
729
730
731 // Computation of the Nusselt numbers
732 void Nusselt(int N, int M, double* x, double* yvc, matrix Q, double Nu[])
733 {
734     for(int i = 0; i<N+1; i++)
735     {
736         Nu[i] = 0;
737         for(int j = 0; j<M+1; j++)
738         {
739             Nu[i] = Nu[i]+(yvc[j+1]-yvc[j])*Q[j][i];
740         }
741     }
742
743
744     double Numax = -100;
745     double Numin = 100;
746     double Nuavg = 0;

```



```

747     double Nu0 = Nu[0];
748     double Nu12 = (Nu[N/2+1]+Nu[N/2+2])/2;
749     int jmax, jmin;
750
751     for(int i = 0; i<N+2; i++)
752     {
753         Nuavg = Nuavg+(x[i+1]-x[i])*Nu[i];
754     }
755     for(int j = 0; j<M+2; j++)
756     {
757         if (Q[j][0]>Numax)
758         {
759             Numax = Q[j][0];
760             jmax = j;
761         }
762         if (Q[j][0]<Numin)
763         {
764             Numin = Q[j][0];
765             jmin = j;
766         }
767     }
768     cout<<endl<<endl;
769     cout<<"Nu average = "<<Nuavg<<endl;
770     cout<<"Nu0 = "<<Nu0<<endl;
771     cout<<"Nu1/2 = "<<Nu12<<endl;
772     cout<<"Nu max = "<<Numax<<" at y = "<<yvc[jmax]<<endl;
773     cout<<"Nu min = "<<Numin<<" at y = "<<yvc[jmin]<<endl;
774 }
775
776
777 // Maximum velocity at the central horizontal and vertical planes
778 void maximum_planes (int N, int M, double* x, double* y, staggx u, staggy v)
779 {
780     double umax = 0, vmax = 0;
781     int imax, jmax;
782     double uavg, vavg;
783     for(int j = 0; j<M+2; j++)
784     {
785         uavg = (u[j][N/2+1]+u[j][N/2])/2;
786         if (uavg>umax)
787         {
788             umax = uavg;
789             jmax = j;
790         }
791     }
792     for(int i = 0; i<N+2; i++)
793     {
794         vavg = (v[M/2+1][i]+v[M/2][i])/2;
795         if (vavg>vmax)
796         {

```

```

797             vmax = vavg;
798             imax = i;
799         }
800     }
801
802     cout<<"u max = "<<umax<<" at y = "<<y[jmax]<<endl;
803     cout<<"v max = "<<vmax<<" at x = "<<x[imax]<<endl;
804 }
805
806
807 // Output of the results
808 void output_files (int N, int M, float L, double* x, double* y, double* xvc, double* yvc, staggx u,
809                 staggy v, matrix T, double* Nu)
810 {
811     // Horizontal coordinates
812     ofstream xx;
813     xx.open("x.dat");
814     for(int i = 0; i<N+2; i++)
815     {
816         xx<<x[i]<<endl;
817     }
818     xx.close();
819
820     // Vertical coordinates
821     ofstream yy;
822     yy.open("y.dat");
823     for(int j = M+1; j>=0; j--)
824     {
825         yy<<y[j]<<endl;
826     }
827     yy.close();
828
829     // Horizontal velocities
830     ofstream resultats ;
831     resultats .open("Resultats.dat");
832     for(int i = 0; i<N+1; i++)
833     {
834         for(int j = 0; j<M+2; j++)
835         {
836             resultats <<xvc[i]<<" "<<y[j]<<" "<<u[j][i]<<endl;
837         }
838         resultats <<endl;
839     }
840     resultats .close();
841
842     // Vertical velocities
843     ofstream resltats ;
844     resltats .open("Resltats.dat");
845     for(int i = 0; i<N+2; i++)
846     {

```

```

846         for(int j = 0; j<M+1; j++)
847         {
848             resvtats <<x[i]<<"    "<<yvc[j]<<"    "<<v[j][i]<<endl;
849         }
850         resvtats <<endl;
851     }
852     resvtats . close();
853
854     // Matrix of horizontal velocities
855     ofstream result ;
856     result .open("Matrixu.dat");
857     for(int j = M+1; j>=0; j--)
858     {
859         for(int i = 0; i<N+2; i++)
860         {
861             if(i==0 || i==N+1)
862             {
863                 result <<u[j][i]<<"    ";
864             }
865             else
866             {
867                 result <<convective_term (x[i], xvc[i-1], xvc[i], u[j][i-1], u[j][i])
868                     <<" ";
869             }
870         }
871         result <<endl;
872     }
873     result . close();
874
875     // Matrix of vertical velocities
876     ofstream resvlt ;
877     resvlt .open("Matrixv.dat");
878     for(int j = M+1; j>=0; j--)
879     {
880         for(int i = 0; i<N+2; i++)
881         {
882             if(j==0 && j==M+1)
883             {
884                 resvlt <<v[j][i]<<"    ";
885             }
886             else
887             {
888                 resvlt <<convective_term (y[j], yvc[j-1], yvc[j], v[j-1][i], v[j][i])
889                     <<" ";
890             }
891         }
892         resvlt <<endl;
893     }
894     resvlt . close();

```

```
894
895 // Temperature
896 ofstream temperature;
897 temperature.open("Temperatura.dat");
898 for(int j = M+1; j>=0; j--)
899 {
900     for(int i = 0; i<N+2; i++)
901     {
902         temperature<<T[j][i]<<" ";
903     }
904     temperature<<endl;
905 }
906 temperature.close();
907
908 // Nusselt number
909 ofstream nuss;
910 nuss.open("Nusselt.dat");
911 for(int i = 0; i<N+1; i++)
912 {
913     nuss<<xvc[i]<<" "<<Nu[i]<<endl;
914 }
915 nuss.close();
916 }
```

5 | Burgers' equation

```
1 #include<iostream>
2 #include<complex>
3 #include<math.h>
4 #include<vector>
5 #include<fstream>
6
7 using namespace std;
8
9
10 complex<double> diffusive(int k, int N, double Re, vector<complex<double> > u, bool LES, float CK);
11 complex<double> convective(int k, int N, vector<complex<double> > u);
12
13
14
15 int main()
16 {
17     const int N = 20;
18     const double Re = 40; // Reynolds number
19     bool LES = 1; // 1 is LES, 0 is DNS
20     double F = 0; // Source term (in Fourier space)
21
22     double delta = 1e-6; // Precision of the simulation
23     float CK = 0.05; // Kolgomorov constant
24     float C1 = 0.02;
25     double dt = C1*Re/pow(N,2); // Increment of time
26
27     vector<complex<double> > u(N);
28     vector<complex<double> > u0(N);
29
30     for(double k = 0; k<N; k++)
31     {
32         u0[k] = 1/(k+1); // u at n
33         u[k] = u0[k]; // u at n+1
34     }
35
36     complex<double> resta;
37     double MAX = 1;
38
```

```

39
40     double t = 0;
41
42     while(MAX>delta)
43     {
44         t = t+dt;
45
46         for(int k = 1; k<N; k++)
47         {
48             u[k] = u0[k]+(diffusive(k, N, Re, u0, LES, CK)-convective(k, N, u0)+F)*dt;
49         }
50
51         // Comprovation
52         MAX = 0;
53         for(int k = 1; k<N; k++)
54         {
55             resta = (u[k]-u0[k])/dt;
56             if(abs(resta)>MAX)
57             {
58                 MAX = abs(resta);
59             }
60         }
61
62         for(int k = 1; k<N; k++)
63         {
64             u0[k] = u[k];
65         }
66     }
67     cout<<"Steady state reached at t="<<t;
68
69     vector<double> E(N);
70     for(int k = 0; k<N; k++)
71     {
72         E[k] = abs(u[k]*conj(u[k]));
73     }
74
75     ofstream results ;
76     results .open("Results.dat");
77     for(int k = 0; k<N; k++)
78     {
79         results <<k+1<<" "<<E[k]<<endl;
80     }
81     results .close();
82
83     return 0;
84 }
85
86
87
88 // Calculation of the diffusive term

```

```

89 complex<double> diffusive(int k, int N, double Re, vector<complex<double> > u, bool LES, float CK)
90 {
91     if(!LES)
92     {
93         return -(double(k)+1)*(double(k)+1)*u[k]/Re;
94     }
95     else
96     {
97         int m = 2; // Slope of the energy spectrum
98
99         double viscosity ;
100        double eddy; // Eddy-viscosity
101        double vinf;
102        double vnon;
103        double EkN = abs(u[N-1]*conj(u[N-1])); //Energy at the cutoff frequency
104
105        vinf = 0.31*(5-m)*sqrt(3-m)*pow(CK,-3/2)/(m+1);
106        vnon = 1+34.5*exp(-3.03*N/k);
107        eddy = vinf*sqrt(EkN/N)*vnon;
108        viscosity = 1/Re+eddy;
109        return -(double(k)+1)*(double(k)+1)*u[k]*viscosity;
110    }
111 }
112
113 // Calculation of the convective term
114 complex<double> convective(int k, int N, vector<complex<double> > u)
115 {
116     complex<double> conv (0,0);
117     complex<double> i(0,1);
118
119     for(int p = -N; p<=N; p++)
120     {
121         int q = k+1-p;
122         if(q>=-N && q<=N)
123         {
124             int qu = q;
125             int pu = p;
126
127             if(qu==0 || pu==0){}
128             else if(qu<0)
129             {
130                 qu = -q;
131                 conv = conv+u[pu-1]*i*double(q)*conj(u[qu-1]);
132             }
133             else if(pu<0)
134             {
135                 pu = -p;
136                 conv = conv+conj(u[pu-1])*i*double(q)*u[qu-1];
137             }
138         }
139     }

```

```
139         else
140         {
141             conv = conv+u[pu-1]*i*double(q)*u[qu-1];
142         }
143     }
144 }
145 return conv;
146 }
```


6 | Square cylinder

```

1  #include<iostream>
2  #include<math.h>
3  #include<fstream>
4
5  using namespace std;
6
7  // Numerical parameters
8  const int N1 = 90;
9  const int N2 = 10;
10 const int N3 = 300;
11 const int N = N1+N2+N3;
12
13 const int M1 = 30;
14 const int M2 = 10;
15 const int M3 = 30;
16 const int M = M1+M2+M3;
17
18 typedef double matrix[M+2][N+2];
19 typedef double staggx[M+2][N+1];
20 typedef double staggy[M+1][N+2];
21 typedef double mtx[M+2][2];
22 typedef double mty[M+1][2];
23
24 void coordinates(float D, float l, float L, int N1, int N2, int N3, double xvc[], double x[]);
25 void surface(double *yvc, int M, double Sv[]);
26 void volume(double *xvc, double *yvc, int N, int M, matrix& V);
27 double parabolic(float umax, float H, double y);
28 void initial_conditions (int N, int M, float umax, float H, double* y, staggx& u0, staggx& Ru0, staggy
    & v0, staggy& Rv0, mtx& u00, mty& v00);
29 void constant_coefficients (int N, int M, double *x, double *y, double *xvc, double *yvc, double *Sv,
    double *Sh, matrix& ae, matrix& aw, matrix& an, matrix& as, matrix& ap);
30 double convective_term (double xf, double x2, double x3, double u2, double u3);
31 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);
32 void bp_coefficient (int N, int M, float rho, double dt, double* Sh, double* Sv, staggx up, staggy vp,
    matrix bp);
33 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);
34 void velocities (int N, int M, float rho, double dt, float umax, float H, double* x, double* y, matrix
    p, staggx up, staggy vp, staggx u0, staggy v0, mtx u00, mty v00, staggx &u, staggy &v);
35 double min(double a, double b);
36 double max(double a, double b);
37 double time_step (double dtd, double* x, double* y, staggx u, staggy v);
38 bool error (int N, int M, float delta, staggx u, staggy v, staggx u0, staggy v0);
39 void search_index (float point, double *x, int Number, int& ipoint, int& ip);
40 void output_files (int N, int M, double* x, double* y, double* xvc, double* yvc, staggx u, staggy v);
41
42 double xvc[N+1], yvc[M+1], x[N+2], y[M+2];
43 double Sh[N+2], Sv[M+2];
44 matrix V;
45 matrix p; // Values in the nodes (pressure)
46 staggx u, u0, Ru0; // Values in the points given by the staggered meshes ( velocities )
47 staggy v, v0, Rv0;
48 mtx u00;
49 mty v00;
50 staggx up, Ru; // Intermediate velocities
51 staggy vp, Rv;
52
53
54 int main()
55 {
56     int Re = 3; // Reynolds number
57     float D = 1; // Diameter of the cylinder
58     float L = 50*D; // Length of the channel
59     float H = 8*D; // Height of the channel
60     float l = L/4; // inflow length
61     float rho = 1; // Density
62     float umax = 1; // Maximum velocity of the inflow parabolic velocity profile
63     float mu = rho*umax*D/Re; // Viscosity
64
65     float delta = 1e-4; // Precision of the simulation
66     float fr = 1.2; // Relaxation factor
67
68     cout<<"Program started"<<endl;
69     cout<<"Re="<<Re<<endl<<endl;
70
71     // Coordinates
72     coordinates(D, l, L, N1, N2, N3, xvc, x);
73     coordinates(D, H/2, H, M1, M2, M3, yvc, y);
74
75     // Surfaces
76
77     surface(xvc, N+2, Sh); // Horizontal surface
78     surface(yvc, M+2, Sv); // Vertical surface
79     volume(xvc, yvc, N+2, M+2, V); // Volume
80
81

```

```

82 // Properties that are going to be calculated
83
84
85 // Inicialization
86 initial_conditions (N, M, umax, H, y, u0, Ru0, v0, Rv0, u00, v00);
87
88 // Calculation of the constant coefficients that are used to determine the pressure
89 matrix ae, aw, an, as, ap, bp;
90 constant_coefficients (N+2, M+2, x, y, xvc, yvc, Sv, Sh, ae, aw, an, as, ap);
91
92 // Time step (CFL condition)
93 double resta = 1;
94 double dtd = 0.2*rho*pow(x[2]-xvc[1],2)/mu;
95 double dtc = 0.35*fabs(x[2]-xvc[1])/umax;
96 double dt = min(dtd, dtc);
97
98
99
100 cout<<"Solving..."<<endl;
101 // Fractional Step Method
102 bool steady = false;
103 while(!steady)
104 {
105     // STEP 1: INTERMEDIATE VELOCITY
106     intermediate_velocities (N, M, rho, mu, delta, dt, x, y, xvc, yvc, Sh, Sv, V, u0, v0,
107                             Ru0, Rv0, Ru, Rv, up, vp);
108
109     // STEP 2: PRESSURE
110     bp_coefficient (N, M, rho, dt, Sh, Sv, up, vp, bp);
111     Gauss_Seidel (ap, aw, ae, as, an, bp, fr, delta, N+2, M+2, p);
112
113     // STEP 3: VELOCITY
114     velocities (N, M, rho, dt, umax, H, x, y, p, up, vp, u0, v0, u00, v00, u, v);
115
116
117     // STEP 4: TIME STEP
118     dt = time_step (dtd, x, y, u, v);
119
120
121     // Comprovation
122     steady = error (N, M, delta, u, v, u0, v0);
123
124     // New time step
125     for(int i = 0; i<N+1; i++)
126     {
127         for(int j = 0; j<M+2; j++)
128         {
129             u0[j][i] = u[j][i];
130

```

```

131             Ru0[j][i] = Ru[j][i];
132         }
133     }
134     for(int i = 0; i<N+2; i++)
135     {
136         for(int j = 0; j<M+1; j++)
137         {
138             v0[j][i] = v[j][i];
139             Rv0[j][i] = Rv[j][i];
140         }
141     }
142     for(int i = 0; i<2; i++)
143     {
144         for(int j = 0; j<M+2; j++)
145         {
146             u00[j][i] = u0[j][i+N-1];
147         }
148     }
149     for(int j = 0; j<M+1; j++)
150     {
151         for(int i = 0; i<2; i++)
152         {
153             v00[j][i] = v0[j][i+N];
154         }
155     }
156 }
157
158 // Results
159 cout<<endl<<"Creating some output files..."<<endl;
160 output_files (N, M, x, y, xvc, yvc, u, v);
161
162
163 return 0;
164 }
165
166
167 // Coordinates of the control volumes (x -> nodes, xvc -> faces)
168 void coordinates(float D, float l, float L, int N1, int N2, int N3, double xvc[], double x[])
169 {
170     double dx;
171     double dx1 = (l-D/2)/N1;
172     double dx2 = D/N2;
173     double dx3 = (L-l-D/2)/N3;
174     xvc[0] = 0;
175     x[0] = 0;
176     for(int i = 0; i<N1+N2+N3; i++)
177     {
178         if(i<N1)
179         {
180             dx = dx1;

```

```

181         }
182         else if (i < N1+N2 && i >= N1)
183         {
184             dx = dx2;
185         }
186         else
187         {
188             dx = dx3;
189         }
190         xvc[i+1] = xvc[i]+dx;
191         x[i+1] = (xvc[i]+xvc[i+1])/2;
192     }
193     x[N1+N2+N3+1] = L;
194 }
195
196
197 // Surfaces of the control volumes
198 void surface(double *yvc, int M, double Sv[])
199 {
200     for(int j = 0; j < M-1; j++)
201     {
202         Sv[j+1] = fabs(yvc[j]-yvc[j+1]);
203     }
204     Sv[0] = 0;
205     Sv[M-1] = 0;
206 }
207
208
209 // Volume of each control volume
210 void volume(double *xvc, double *yvc, int N, int M, matrix& V)
211 {
212     for(int i = 0; i < N; i++)
213     {
214         for(int j = 0; j < M; j++)
215         {
216             if (i==0 || i==N-1 || j==0 || j==M-1)
217             {
218                 V[j][i] = 0;
219             }
220             else
221             {
222                 V[j][i] = fabs(xvc[i]-xvc[i-1])*fabs(yvc[j]-yvc[j-1]);
223             }
224         }
225     }
226 }
227
228
229 // Parabolic velocity profile in the input
230 double parabolic(float umax, float H, double y)

```

```

231 {
232     return 4*umax*(y/H-y*y/(H*H));
233 }
234
235
236 // Initial conditions of the problem
237 void initial_conditions (int N, int M, float umax, float H, double* y, staggx& u0, staggx& Ru0, staggy
    & v0, staggy& Rv0, mtgx& u00, mty& v00)
238 {
239     for(int j = 0; j<M+2; j++)
240     {
241         for(int i = 0; i<N+1; i++)
242         {
243             if(i==0)
244             {
245                 u0[j][i] = parabolic(umax, H, y[j]); // Horizontal velocity at n
246             }
247             else
248             {
249                 u0[j][i] = 0; // Horizontal velocity at n
250             }
251             Ru0[j][i] = 0; // R ( horizontal ) at n-1
252         }
253     }
254     for(int j = 0; j<M+1; j++)
255     {
256         for(int i = 0; i<N+2; i++)
257         {
258             v0[j][i] = 0; // Vertical velocity at n
259             Rv0[j][i] = 0; // R ( vertical ) at n-1
260         }
261     }
262     for(int i = 0; i<2; i++)
263     {
264         for(int j = 0; j<M+2; j++)
265         {
266             u00[j][i] = 0; // Horizontal velocity at n-1
267         }
268     }
269     for(int j = 0; j<M+1; j++)
270     {
271         for(int i = 0; i<2; i++)
272         {
273             v00[j][i] = 0; // Vertical velocity at n-1
274         }
275     }
276 }
277
278
279 // Calculation of the constant coefficients (ae, aw, an, as, ap) of the Poisson equation (pressure)

```

```

280 void constant_coefficients (int N, int M, double *x, double *y, double *xvc, double *yvc, double *Sv,
    double *Sh, matrix& ae, matrix& aw, matrix& an, matrix& as, matrix& ap)
281 {
282     for(int i = 0; i<N; i++)
283     {
284         for(int j = 0; j<M; j++)
285         {
286             // Coefficients in the channel walls, input and output
287             if (j==M-1 && i!=0 && i!=N-1)
288             {
289                 ae[j][i] = 0;
290                 aw[j][i] = 0;
291                 an[j][i] = 0;
292                 as[j][i] = 1;
293                 ap[j][i] = 1;
294             }
295             else if (i==0 && j==0)
296             {
297                 ae[j][i] = 1;
298                 aw[j][i] = 0;
299                 an[j][i] = 1;
300                 as[j][i] = 0;
301                 ap[j][i] = 1;
302             }
303             else if (i==0 && j==M-1)
304             {
305                 ae[j][i] = 1;
306                 aw[j][i] = 0;
307                 an[j][i] = 0;
308                 as[j][i] = 1;
309                 ap[j][i] = 1;
310             }
311             else if (i==0 && j!=0 && j!=M-1)
312             {
313                 ae[j][i] = 1;
314                 aw[j][i] = 0;
315                 an[j][i] = 0;
316                 as[j][i] = 0;
317                 ap[j][i] = 1;
318             }
319             else if (i==N-1 && j==0)
320             {
321                 ae[j][i] = 0;
322                 aw[j][i] = 0;
323                 an[j][i] = 1;
324                 as[j][i] = 0;
325                 ap[j][i] = 1;
326             }
327             else if (i==N-1 && j==M-1)
328             {

```

```

329         ae[j][i] = 0;
330         aw[j][i] = 0;
331         an[j][i] = 0;
332         as[j][i] = 1;
333         ap[j][i] = 1;
334     }
335     else if(i==N-1 && j!=0 && j!=M-1)
336     {
337         ae[j][i] = 0;
338         aw[j][i] = 0;
339         an[j][i] = 0;
340         as[j][i] = 0;
341         ap[j][i] = 1;
342     }
343     else if(j==0 && i!=0 && i!=N-1)
344     {
345         ae[j][i] = 0;
346         aw[j][i] = 0;
347         an[j][i] = 1;
348         as[j][i] = 0;
349         ap[j][i] = 1;
350     }
351     // Coefficients in the cylinder
352     else if(i>N1+1 && i<N1+N2 && j>M1+1 && j<M1+M2)
353     {
354         ae[j][i] = 0;
355         aw[j][i] = 0;
356         an[j][i] = 0;
357         as[j][i] = 0;
358         ap[j][i] = 1;
359     }
360     // Coefficients in the points near the cylinder
361     else if(i==N1 && j>M1 && j<M1+M2+1)
362     {
363         ae[j][i] = Sv[j]/fabs(xvc[i]-x[i]);
364         aw[j][i] = Sv[j]/fabs(x[i]-x[i-1]);
365         an[j][i] = Sh[i]/fabs(y[j+1]-y[j]);
366         as[j][i] = Sh[i]/fabs(y[j]-y[j-1]);
367         ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i];
368     }
369     else if(i==N1+N2+1 && j>M1 && j<M1+M2+1)
370     {
371         ae[j][i] = Sv[j]/fabs(x[i+1]-x[i]);
372         aw[j][i] = Sv[j]/fabs(x[i]-xvc[i-1]);
373         an[j][i] = Sh[i]/fabs(y[j+1]-y[j]);
374         as[j][i] = Sh[i]/fabs(y[j]-y[j-1]);
375         ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i];
376     }
377     else if(i>N1 && i<N1+N2+1 && j==M1)
378     {

```



```

379         ae[j][i] = Sv[j]/fabs(x[i+1]-x[i]);
380         aw[j][i] = Sv[j]/fabs(x[i]-x[i-1]);
381         an[j][i] = Sh[i]/fabs(yvc[j]-y[j]);
382         as[j][i] = Sh[i]/fabs(y[j]-y[j-1]);
383         ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i];
384     }
385     else if(i>N1 && i<N1+N2+1 && j==M1+M2+1)
386     {
387         ae[j][i] = Sv[j]/fabs(x[i+1]-x[i]);
388         aw[j][i] = Sv[j]/fabs(x[i]-x[i-1]);
389         an[j][i] = Sh[i]/fabs(y[j+1]-y[j]);
390         as[j][i] = Sh[i]/fabs(y[j]-yvc[j-1]);
391         ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i];
392     }
393     // Coefficients in the channel
394     else
395     {
396         ae[j][i] = Sv[j]/fabs(x[i+1]-x[i]);
397         aw[j][i] = Sv[j]/fabs(x[i]-x[i-1]);
398         an[j][i] = Sh[i]/fabs(y[j+1]-y[j]);
399         as[j][i] = Sh[i]/fabs(y[j]-y[j-1]);
400         ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i];
401     }
402 }
403 }
404 }
405
406
407 // Computation of the velocity in the convective term using CDS
408 double convective_term (double xf, double x2, double x3, double u2, double u3)
409 {
410     // 2 refers to node P, 3 to node E
411     double u;
412     u = u2+fabs(x2-xf)*(u3-u2)/fabs(x3-x2);
413
414     return u;
415 }
416
417
418 void intermediate_velocities (int N, int M, float rho, float mu, float delta, double dt, double* x,
419     double* y, double* xvc, double* yvc, double* Sh, double* Sv, matrix V, staggx u0, staggy v0,
420     staggx Ru0, staggy Rv0, staggx &Ru, staggy &Rv, staggx &up, staggy &vp)
421 {
422     double mflowe, mfloww, mflown, mflows;
423     double ue, uw, un, us;
424
425     for(int i = 0; i<N+1; i++)
426     {

```

```

427 // Mass flow terms (rho*v*S)
428 mflowe = (rho*u0[j][i+1]+rho*u0[j][i])*Sv[j]/2;
429 mfloww = (rho*u0[j][i-1]+rho*u0[j][i])*Sv[j]/2;
430 mflown = (rho*v0[j][i]+rho*v0[j][i+1])*Sh[i]/2;
431 mflows = (rho*v0[j-1][i]+rho*v0[j-1][i+1])*Sh[i]/2;
432
433
434 // HORIZONTAL
435 ue = convective_term (x[i+1], xvc[i], xvc[i+1], u0[j][i], u0[j][i+1]);
436 uw = convective_term (x[i], xvc[i], xvc[i-1], u0[j][i], u0[j][i-1]);
437 un = convective_term (yvc[j], y[j], y[j+1], u0[j][i], u0[j+1][i]);
438 us = convective_term (yvc[j-1], y[j], y[j-1], u0[j][i], u0[j-1][i]);
439
440
441 // R ( horizontal )
442 // Channel walls, input and output
443 if (i==0 || i==N || j==0 || j==M+1)
444 {
445     Ru[j][i] = 0;
446 }
447 // In the cylinder
448 else if (i>=N1 && i<=N1+N2 && j>M1 && j<M1+M2+1)
449 {
450     Ru[j][i] = 0;
451 }
452 // Points that surround the cylinder
453 else if (j==M1 && i>=N1 && i<=N1+N2)
454 {
455     un = 0;
456     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(yvc[j]-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];
457 }
458 else if (j==M1+M2+1 && i>=N1 && i<=N1+N2)
459 {
460     us = 0;
461     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]-yvc[j-1])-(mflowe*ue+mflown*un-mfloww*uw-mflows*us))/V[j][i];
462 }
463 // Channel
464 else
465 {
466     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]/

```

```

467         fabs(y[j]-y[j-1])-(mflowe*ue+mflown*un-mfloww*uw-mflows*us
468         ))/V[j][i];
469     }
470     // Intermediate velocity ( horizontal )
471     up[j][i] = u0[j][i]+dt*(1.5*Ru[j][i]-0.5*Ru0[j][i])/rho;
472 }
473
474 double ve, vw, vn, vs;
475
476 for(int i = 0; i<N+2; i++)
477 {
478     for(int j = 0; j<M+1; j++)
479     {
480         // Mass flow terms (rho*v*S)
481         mflowe = (rho*u0[j+1][i]+rho*u0[j][i])*Sv[j]/2;
482         mfloww = (rho*u0[j+1][i-1]+rho*u0[j][i-1])*Sv[j]/2;
483         mflown = (rho*v0[j][i]+rho*v0[j+1][i])*Sh[i]/2;
484         mflows = (rho*v0[j][i]+rho*v0[j-1][i])*Sh[i]/2;
485
486
487         // VERTICAL
488         ve = convective_term(xvc[i], x[i], x[i+1], v0[j][i], v0[j][i+1]);
489         vw = convective_term(xvc[i-1], x[i], x[i-1], v0[j][i], v0[j][i-1]);
490         vn = convective_term(y[j+1], yvc[j], yvc[j+1], v0[j][i], v0[j+1][i]);
491         vs = convective_term(y[j], yvc[j], yvc[j-1], v0[j][i], v0[j-1][i]);
492
493         // R ( vertical )
494         // Channel walls, input and output
495         if(i==0 || i==N+1 || j==0 || j==M)
496         {
497             Rv[j][i] = 0;
498         }
499         // In the cylinder
500         else if(i>N1 && i<N1+N2+1 && j>=M1 && j<=M1+M2)
501         {
502             Rv[j][i] = 0;
503         }
504         // Points that surround the cylinder
505         else if(j>=M1 && j<=M1+M2 && i==N1)
506         {
507             ve = 0;
508             Rv[j][i] = (mu*(v0[j][i+1]-v0[j][i])*Sv[j]/fabs(xvc[i]-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]/
+1][i]-v0[j][i])*Sh[i]/fabs(xvc[i]-x[i-1])-mu*(v0[j][i]-v0[j-1][i])*Sh[i]/
+1][i]-v0[j][i])*Sh[i]/fabs(yvc[j]-yvc[j-1])-(mflowe*ve+mflown*vn-mfloww*vw-mflows*vs))
+1][i]-v0[j][i])*Sh[i]/fabs(yvc[j]-yvc[j-1])-(mflowe*ve+mflown*vn-mfloww*vw-mflows*vs))
509         }
510         else if(j>=M1 && j<=M1+M2 && i==N1+N2+1)

```

```

511         {
512             vw = 0;
513             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]-xvc[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];
514         }
515         // Channel
516         else
517         {
518             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]-xvc[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];
519         }
520
521         // Intermediate velocity ( vertical )
522         vp[j][i] = v0[j][i]+dt*(1.5*Rv[j][i]-0.5*Rv0[j][i])/rho;
523     }
524 }
525 }
526
527 // Calculation of the bp coefficient of the Poisson equation ( pressure )
528 void bp_coefficient (int N, int M, float rho, double dt, double* Sh, double* Sv, staggx up, staggy vp,
529                     matrix bp)
530 {
531     for(int i = 0; i<N+2; i++)
532     {
533         for(int j = 0; j<M+2; j++)
534         {
535             // Channel walls, input and output
536             if(i==0 || j==0 || i==N-1 || j==M-1)
537             {
538                 bp[j][i] = 0;
539             }
540             // Cylinder
541             else if(i>N1 && i<N1+N2+1 && j>M1 && j<M1+M2+1)
542             {
543                 bp[j][i] = 0;
544             }
545             // Channel
546             else
547             {
548                 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;
549             }
550         }

```

```

551     }
552 }
553
554
555 // Solver (using Gauss–Seidel)
556 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)
557 {
558     double Tcalc[M][N]; // Temperature calculated in the previous iteration
559     for(int i = 0; i<N; i++)
560     {
561         for(int j = 0; j<M; j++)
562         {
563             Tcalc[j][i] = T[j][i];
564         }
565     }
566
567     double MAX = 1; // Maximum value of the difference between T and Tcalc
568     double resta = 1; // Difference between T and Tcalc
569
570     while(MAX>delta)
571     {
572
573         // SOLVER: Gauss–Seidel
574         for(int i = 0; i<N; i++)
575         {
576             for(int j = 0; j<M; j++)
577             {
578                 if(i==0 && j==M-1)
579                 {
580                     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]);
581                 }
582                 else if(i==0 && j==0)
583                 {
584                     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]);
585                 }
586                 else if(i==0 && j!=0 && j!=M-1)
587                 {
588                     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]);
589                 }
590                 else if(i==N-1 && j==M-1)
591                 {
592                     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]);
593                 }
594                 else if(i==N-1 && j==0)

```

```

595         {
596              $T[j][i] = Tcalc[j][i] + fr * ((aw[j][i] * T[j][i-1] + an[j][i] * Tcalc[j$ 
                     $+ 1][i] + bp[j][i]) / ap[j][i] - Tcalc[j][i]);$ 
597         }
598         else if (i == N-1 && j != 0 && j != M-1)
599         {
600              $T[j][i] = Tcalc[j][i] + fr * ((aw[j][i] * T[j][i-1] + as[j][i] * T[j-1][$ 
                     $i] + an[j][i] * Tcalc[j+1][i] + bp[j][i]) / ap[j][i] - Tcalc[j][i])$ 
                    ;
601         }
602         else if (i != 0 && i != N-1 && j == M-1)
603         {
604              $T[j][i] = Tcalc[j][i] + fr * ((aw[j][i] * T[j][i-1] + ae[j][i] * Tcalc[j$ 
                     $][i+1] + as[j][i] * T[j-1][i] + bp[j][i]) / ap[j][i] - Tcalc[j][i])$ 
                    ;
605         }
606         else if (i != 0 && i != N-1 && j == 0)
607         {
608              $T[j][i] = Tcalc[j][i] + fr * ((aw[j][i] * T[j][i-1] + ae[j][i] * Tcalc[j$ 
                     $][i+1] + an[j][i] * Tcalc[j+1][i] + bp[j][i]) / ap[j][i] - Tcalc[j$ 
                     $][i]);$ 
609         }
610         else
611         {
612              $T[j][i] = Tcalc[j][i] + fr * ((aw[j][i] * T[j][i-1] + ae[j][i] * Tcalc[j$ 
                     $][i+1] + as[j][i] * T[j-1][i] + an[j][i] * Tcalc[j+1][i] + bp[j][i]$ 
                     $]) / ap[j][i] - Tcalc[j][i]);$ 
613         }
614     }
615 }
616
617 // Comprovation
618 MAX = 0;
619 for(int i = 0; i < N; i++)
620 {
621     for(int j = 0; j < M; j++)
622     {
623         resta = fabs(Tcalc[j][i] - T[j][i]);
624
625         if (resta > MAX)
626         {
627             MAX = resta;
628         }
629     }
630 }
631
632 // New assignation
633 for(int i = 0; i < N; i++)
634 {
635     for(int j = 0; j < M; j++)

```

```

636         {
637             Tcalc[j][i] = T[j][i];
638         }
639     }
640 }
641 }
642
643
644 // Calculation of the velocity with the pressure correction
645 void velocities (int N, int M, float rho, double dt, float umax, float H, double* x, double* y, matrix
        p, staggx up, staggy vp, staggx u0, staggy v0, mtx u00, mty v00, staggx &u, staggy &v)
646 {
647     // Horizontal velocity at n+1
648     for(int i = 0; i<N+1; i++)
649     {
650         for(int j = 0; j<M+2; j++)
651         {
652             // Channel walls
653             if(j==0 || j==M+1)
654             {
655                 u[j][i] = 0;
656             }
657             // Channel input
658             else if(i==0)
659             {
660                 u[j][i] = parabolic(umax, H, y[j]);
661             }
662             // Channel output
663             else if(i==N)
664             {
665                 u[j][i] = u0[j][i] - dt*umax*(1.5*(u0[j][i] - u0[j][i-1]) - 0.5*(u00[j][2] -
                        u00[j][1]))/(0.5*fabs(x[i] - x[i-1]));
666             }
667             // Cylinder
668             else if(i>=N1 && i<=N1+N2 && j>M1 && j<M1+M2+1)
669             {
670                 u[j][i] = 0;
671             }
672             // Channel
673             else
674             {
675                 u[j][i] = up[j][i] - dt*(p[j][i+1] - p[j][i])/(rho*fabs(x[i+1] - x[i]));
676             }
677         }
678     }
679
680     // Vertical velocity at n+1
681     for(int i = 0; i<N+2; i++)
682     {
683         for(int j = 0; j<M+1; j++)

```

```

684         {
685             // Channel walls and input
686             if (j==0 || j==M || i==0)
687             {
688                 v[j][i] = 0;
689             }
690             // Channel output
691             else if (i==N+1)
692             {
693                 v[j][i] = v0[j][i] - dt*umax*(1.5*(v0[j][i] - v0[j][i-1]) - 0.5*(v00[j][2] -
694                     v00[j][1])) / fabs(x[i] - x[i-1]);
695             }
696             // Cylinder
697             else if (i>N1 && i<N1+N2+1 && j>=M1 && j<=M1+M2)
698             {
699                 v[j][i] = 0;
700             }
701             else
702             {
703                 v[j][i] = vp[j][i] - dt*(p[j+1][i] - p[j][i]) / (rho*fabs(y[j+1] - y[j]));
704             }
705         }
706     }
707
708 // Returns the minimum value
709 double min(double a, double b)
710 {
711     if (a>b)
712     {
713         return b;
714     }
715     else
716     {
717         return a;
718     }
719 }
720
721 // Returns the maximum value
722 double max(double a, double b)
723 {
724     if (a>b)
725     {
726         return a;
727     }
728     else
729     {
730         return b;
731     }
732 }

```



```

733     }
734 }
735
736
737 // Calculation of the proper time step (CFL condition)
738 double time_step (double dtd, double* x, double* y, staggx u, staggy v)
739 {
740     double dt;
741     double dtc = 100;
742
743     for(int i = 1; i<N; i++)
744     {
745         for(int j = 1; j<M+1; j++)
746         {
747             if(i>=N1 && i<=N1+N2 && j>M1 && j<M1+M2+1)
748             {}
749             else
750             {
751                 dtc = min(dtc, 0.35*fabs(x[i+1]-x[i])/fabs(u[j][i]));
752             }
753         }
754     }
755     for(int i = 1; i<N+1; i++)
756     {
757         for(int j = 1; j<M; j++)
758         {
759             if(i>N1 && i<N1+N2+1 && j>=M1 && j<=M1+M2)
760             {}
761             else
762             {
763                 dtc = min(dtc, 0.35*fabs(y[j+1]-y[j])/fabs(v[j][i]));
764             }
765         }
766     }
767     dt = min(dtc, dtd);
768     return dt;
769 }
770
771
772 // Difference between the previous and the actual time step
773 bool error (int N, int M, float delta, staggx u, staggy v, staggx u0, staggy v0)
774 {
775     double resta = 0;
776     for(int i = 0; i<N+1; i++)
777     {
778         for(int j = 0; j<M+2; j++)
779         {
780             resta = max(resta, fabs(u[j][i]-u0[j][i]));
781         }
782     }

```

```

783     for(int i = 0; i<N+2; i++)
784     {
785         for(int j = 0; j<M+1; j++)
786         {
787             resta = max(resta, fabs(v[j][i]-v0[j][i]));
788         }
789     }
790     cout<<resta<<endl;
791     if (resta>delta)
792     {
793         return false;
794     }
795     else
796     {
797         return true;
798     }
799 }
800
801
802 // Searching the index of the node closest to a given point (and the second closest)
803 void search_index (float point, double *x, int Number, int& ipoint, int& ip)
804 {
805     for(int i = 0; i<Number-1; i++)
806     {
807         if (x[i+1]-x[i]>0)
808         {
809             if (x[i]<=point && x[i+1]>point)
810             {
811                 if (point-x[i]<x[i+1]-point)
812                 {
813                     ipoint = i; //ipoint is the index of the node closest to the
814                             point we want
815                     ip = i+1; //ip is the second node closest to it (used in
816                             interpolation)
817                 }
818                 else
819                 {
820                     ipoint = i+1;
821                     ip = i;
822                 }
823             }
824         }
825         else
826         {
827             if (x[i]>point && x[i+1]<=point)
828             {
829                 if (point-x[i+1]<x[i]-point)
830                 {
831                     ipoint = i;
832                     ip = i+1;
833                 }
834             }
835         }
836     }
837 }

```

```

831         }
832         else
833         {
834             ipoint = i+1;
835             ip = i;
836         }
837     }
838 }
839 }
840
841 }
842
843
844 // Output of the results
845 void output_files (int N, int M, double* x, double* y, double* xvc, double* yvc, staggx u, staggy v)
846 {
847     // Horizontal coordinates
848     ofstream xx;
849     xx.open("x.dat");
850     for(int i = 0; i<N+2; i++)
851     {
852         xx<<x[i]<<endl;
853     }
854     xx.close();
855
856     // Vertical coordinates
857     ofstream yy;
858     yy.open("y.dat");
859     for(int j = M+1; j>=0; j--)
860     {
861         yy<<y[j]<<endl;
862     }
863     yy.close();
864
865     // Horizontal velocities
866     ofstream resultats ;
867     resultats .open("Resultats.dat");
868     for(int j = M+1; j>=0; j--)
869     {
870         for(int i = 0; i<N+1; i++)
871         {
872             if (i>N1 && i<N1+N2 && j>M1+1 && j<M1+M2)
873             {
874                 resultats <<xvc[i]<<" " <<y[j]<<" " <<"nan"<<endl;
875             }
876             else if (i>N1 && i<N1+N2 && j==M1+1)
877             {
878                 resultats <<xvc[i]<<" " <<yvc[j-1]<<" " <<0<<endl;
879             }
880             else if (i>N1 && i<N1+N2 && j==M1+M2)

```

```

881         {
882             resultats <<xvc[i]<<" "<<yvc[j]<<" "<<0<<endl;
883         }
884         else
885         {
886             resultats <<xvc[i]<<" "<<y[j]<<" "<<u[j][i]<<endl;
887         }
888     }
889     resultats <<endl;
890 }
891 resultats . close();
892
893 // Vertical velocities
894 ofstream resvltats ;
895 resvltats .open("Resvltats.dat");
896 for(int j = M; j>=0; j--)
897 {
898     for(int i = 0; i<N+2; i++)
899     {
900         if(i>N1+1 && i<N1+N2 && j>M1 && j<M1+M2)
901         {
902             resvltats <<x[i]<<" "<<yvc[j]<<" "<<"nan"<<endl;
903         }
904         else if(i==N1+1 && j>M1 && j<M1+M2)
905         {
906             resvltats <<xvc[i-1]<<" "<<yvc[j]<<" "<<0<<endl;
907         }
908         else if(i==N1+N2 && j>M1 && j<M1+M2)
909         {
910             resvltats <<xvc[i]<<" "<<yvc[j]<<" "<<0<<endl;
911         }
912         else
913         {
914             resvltats <<x[i]<<" "<<yvc[j]<<" "<<v[j][i]<<endl;
915         }
916     }
917     resvltats <<endl;
918 }
919 resvltats . close();
920
921 // Matrix of horizontal velocities
922 ofstream result ;
923 result .open("Matrixu.dat");
924 for(int j = M+1; j>=0; j--)
925 {
926     for(int i = 0; i<N+2; i++)
927     {
928         if(i==0 || i==N+1)
929         {
930             result <<u[j][i]<<" ";

```

```

931         }
932     else
933     {
934         result << convective_term (x[i], xvc[i-1], xvc[i], u[j][i-1], u[j][i])
935         << " ";
936     }
937     result << endl;
938 }
939 result . close();
940
941 // Matrix of vertical velocities
942 ofstream resvlt ;
943 resvlt . open("Matrixv.dat");
944 for(int j = M+1; j>=0; j--)
945 {
946     for(int i = 0; i<N+2; i++)
947     {
948         if (j==0 && j==M+1)
949         {
950             resvlt << v[j][i] << " ";
951         }
952         else
953         {
954             resvlt << convective_term (y[j], yvc[j-1], yvc[j], v[j-1][i], v[j][i])
955             << " ";
956         }
957     }
958     resvlt << endl;
959 }
960 resvlt . close();
961
962 // Pressure matrix
963 ofstream press ;
964 press . open("Pressure.dat");
965 for(int j = M+1; j>=0; j--)
966 {
967     for(int i = 0; i<N+2; i++)
968     {
969         press << p[j][i] << " ";
970     }
971     press << endl;
972 }
973 press . close();
974
975 // Pressure
976 ofstream presultats ;
977 presultats . open("Presultats.dat");
978 for(int j = M+1; j>=0; j--)

```

```
979 {
980     for(int i = 0; i<N+2; i++)
981     {
982         if (i>N1 && i<=N1+N2 && j>M1 && j<=M1+M2)
983         {
984             presultats <<x[i]<<"    "<<y[j]<<"    "<<"nan"<<endl;
985         }
986         else
987         {
988             presultats <<x[i]<<"    "<<y[j]<<"    "<<p[j][i]<<endl;
989         }
990     }
991     presultats <<endl;
992 }
993
994 }
```