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

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

using namespace std;

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

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


// FUNCTIONS
void horizontal_coordinates (double dx1, double dx2, double xvc [], double x []) ;
void vertical_coordinates (double dy1, double dy2, double dy3, double yvc [], double y []) ;
void volume (double *xvc, double *yvc, int N, int M, matrix& V);
void surface (double *yvc, int M, double Sx[]);
void properties (double *x, double *y, const float p [3][2], const float rhod [4], const float cpd [4],
    const float lamd [4], matrix& rho, matrix& cp, matrix& lambda);
void harmonic_mean (matrix lambda, double* x, double* y, double* xvc, double* yvc, int N, int M,
    matrix& lambdaw, matrix& lambdae, matrix& lambdas, matrix& lambdan);
void search_index ( float point , double *x, int Number, int &ipoint, int& ip);
void constant_coefficients (double *x, double *y, double *xvc, double *yvc, double *Sx, double *Sy,
    matrix V, float dt , float beta , float alpha , matrix rho , matrix cp , matrix lambda, matrix
    lambdaw, matrix lambdae, matrix lambdas, matrix lambdan, matrix& ap, matrix& aw, matrix& ae,
    matrix& as, matrix& an);
void bp_coefficients (double *x, double *y, double *xvc, double *yvc, double *Sx, double *Sy, double
    Sytotal , matrix V, float dt , float beta , float alpha , float Qtop, float qv, float Tbottom, float
    Tgleft , float Tright , float Trightant , matrix Tant, matrix rho , matrix cp , matrix lambda, matrix
    lambdaw, matrix lambdae, matrix lambdas, matrix lambdan, matrix& bp);
void Gauss_Seidel (matrix ap, matrix aw, matrix ae , matrix as , matrix an, matrix bp, float fr , float
    delta , int N, int M, matrix& T);
double double_interpolation ( float x, float y , double T11, double T12, double T21, double T22,
        double x1, double x2, double y1, double y2);
void print_matrix (matrix T, int N, int M);
void output_file (double* Tpoint1, double* Tpoint2, int Time, float dt);


int main(){

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

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

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

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

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


            cout<<"Program started"<<endl;

            // PREVIOUS CALCULATIONS

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

```
    L2 = p[2][0]−L1;
    H1 = p[0][1];
    H2 = p[1][1]−H1;
    H3 = p[2][1]−H1−H2;

    double dx1, dx2, dy1, dy2, dy3, dt; // Increments of space and time
    dt = tfinal/(Time−1); // Increment of time
    dx1 = L1/N1; // Increments in the horizontal direction
    dx2 = L2/N2;
    dy1 = H1/M1; // Increments in the vertical direction
    dy2 = H2/M2;
    dy3 = H3/M3;

    // Coordinates
    double xvc[N1+N2+1],yvc[M1+M2+M3+1]; // Coordinates of the faces
    double x[N1+N2],y[M1+M2+M3]; // Coordinates of the nodes
    xvc[0] = 0;
    horizontal_coordinates (dx1, dx2, xvc, x);
    yvc[0] = p[2][1];
    vertical_coordinates (dy1, dy2, dy3, yvc, y);

    // Surfaces and volumes
    double Sx[M1+M2+M3], Sy[N1+N2], V[M1+M2+M3][N1+N2], Sytotal; // Surfaces and volumes
    Sytotal = p[2][1]; // Total surface of the north face
    volume (xvc, yvc, N1+N2, M1+M2+M3, V);
    surface (yvc, M1+M2+M3, Sx);
    surface (xvc, N1+N2, Sy);


    cout<<"Calculating properties ... "<<endl;

    // Density, specific heat and conductivity
    matrix rho, cp, lambda; // Density, specific heat and conductivity
    properties (x, y, p, rhod, cpd, lamd, rho, cp, lambda);


    // Harmonic mean
    matrix lambdaw, lambdae, lambdas, lambdan; // Harmonic mean
    harmonic_mean (lambda, x, y, xvc, yvc, N1+N2, M1+M2+M3, lambdaw, lambdae, lambdas,
        lambdan);

    // INITIALIZATION
    matrix T, Tant; // Temperature and Temperature in the previous instant of time
    float Tright, Trightant; // Temperature on the right and Temperature on the right in the
        previous instant of time
    double Tpoint1[Time], Tpoint2[Time]; // Temperatures at the points that are going to be
        studied
    for(int i = 0; i<N1+N2; i++)
    {
        for(int j = 0; j<M1+M2+M3; j++)
```

```
                {
                        T[j][i] = T0;
                        Tant[j][i] = T0;
                        Tpoint1[0] = T0;
                        Tpoint2[0] = T0;
                }
        }
        Tright = Tright0;

        // Searching for the points (0.65, 0.56) and (0.74, 0.72)
        int ipoint1, jpoint1, ip1, jp1, ipoint2, jpoint2, ip2, jp2;
    search_index (point [0][0], x, N1+N2, ipoint1, ip1);
    search_index (point [1][0], x, N1+N2, ipoint2, ip2);
    search_index (point [0][1], y, M1+M2+M3, jpoint1, jp1);
    search_index (point [1][1], y, M1+M2+M3, jpoint2, jp2);


        // CALCULATION OF CONSTANT COEFFICIENTS
        matrix ap, ae, aw, as, an, bp; // Coefficients
         constant_coefficients (x, y, xvc, yvc, Sx, Sy, V, dt, beta, alpha, rho, cp, lambda, lambdaw,
             lambdae, lambdas, lambdan, ap, aw, ae, as, an);


        cout<<"Solving..."<<endl;

        float t = 0.00; // First time increment
        double resta;
        double MAX;
        int k = 0;
        while(t<=tfinal)
        {
                k = k+1;
                t = t+dt;
                Trightant = Tright;
                Tright = Tright0+variationright*t;

                // CALCULATION OF NON-CONSTANT COEFFICIENTS
                 bp_coefficients (x, y, xvc, yvc, Sx, Sy, Sytotal, V, dt, beta, alpha, Qtop, qv,
                     Tbottom, Tgleft, Tright, Trightant, Tant, rho, cp, lambda, lambdaw, lambdae,
                     lambdas, lambdan, bp);

                // SOLVER
                Gauss_Seidel (ap, aw, ae, as, an, bp, fr, delta, N1+N2, M1+M2+M3, T);

                // Assignation of the instant of time
                for(int i = 0; i<N1+N2; i++)
                {
                        for(int j = 0; j<M1+M2+M3; j++)
                        {
                                Tant[j][i] = T[j][i];
```

```
                    }
                }

                // Temperature at the given  points
                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 ])
                    ;
                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 ])
                    ;
        }

        cout<<endl<<endl<<"Final temperature:"<<endl;

        // Output of the matrix temperature at the  final  instant  of time
        print_matrix (T, N1+N2, M1+M2+M3);

    // Output  file
    cout<<"Creating file ... "<<endl;
    output_file  (Tpoint1,  Tpoint2,  Time, dt);
//        resultaats  (x,  y,  T,  N1+N2,  M1+M2+M3);

    cout<<"End of program"<<endl;

    ofstream  results ;
    results .open("Ressultats5000.dat");
    int  N = N1+N2;
    int  M = M1+M2+M3;
    for(int  i  = -1; i<N+1; i++)
    {
        for(int  j  = -1; j<M+1; j++)
        {
                if(i==-1 && j==-1)
                {
                        results <<0.000<<"     "<<0.800<<"   "<<(200*T[0][0]/0.005+alpha*Tgleft)/(
                            alpha+200/0.005)<<endl;
                }
                else  if(i==-1 && j==M)
                {
                        results <<0.000<<"     "<<0.000<<"   "<<23.000<<endl;
                }
                else  if(i==-1 && j!=-1 && j!=M)
                {
                        results <<0.000<<"     "<<y[j]<<"     "<<(lambda[j][0]*T[j
                            ][0]/0.005+alpha*Tgleft)/(alpha+lambda[j][0]/0.005)<<endl;
                }
                else  if(i==N && j==-1)
                {
                        results <<1.100<<"     "<<0.800<<"   "<<8+0.005*tfinal<<endl;
                }
```

```
                    else if(i==N && j==M)
                    {
                        results <<1.100<<"    "<<0.000<<"  "<<8+0.005*tfinal<<endl;
                    }
                    else if(i==N && j!=-1 && j!=M)
                    {
                        results <<1.100<<"    "<<y[j]<<"    "<<8+0.005*tfinal<<endl;
                    }
                    else if(j==-1 && i!=-1 && i!=N)
                    {
                        results <<x[i]<<"      "<<0.800<<"  "<<T[0][i]+Qtop*0.005/(1.10*
                            lambda[0][i]*0.005)<<endl;
                    }
                    else if(j==M && i!=-1 && i!=N)
                    {
                        results <<x[i]<<"      "<<0.000<<"  "<<23.000<<endl;
                    }
                else
                {
                    results <<x[i]<<"      "<<y[j]<<"    "<<T[j][i]<<endl;
                    }
                }
                results <<endl;
        }
        results.close();

    return 0;

}



void horizontal_coordinates (double dx1, double dx2, double xvc[], double x[])
{
        for (int i = 1; i<N1+N2+1; i++)
        {
                if(i<=N1)
                {
                        xvc[i] = xvc[i-1]+dx1;
                        x[i-1] = (xvc[i-1]+xvc[i])/2;
                }
                else
                {
                        xvc[i] = xvc[i-1]+dx2;
                        x[i-1] = (xvc[i-1]+xvc[i])/2;
                }
        }
}


```

```
void vertical_coordinates (double dy1, double dy2, double dy3, double yvc[], double y[])
{
        for (int j = 1; j<M1+M2+M3+1; j++)
        {
                if(j<=M3)
                {
                        yvc[j] = yvc[j-1]-dy3;
                        y[j-1] = (yvc[j-1]+yvc[j])/2;
                }
                else if(j>M3 && j<=M2+M3)
                {
                        yvc[j] = yvc[j-1]-dy2;
                        y[j-1] = (yvc[j-1]+yvc[j])/2;
                }
                else
                {
                        yvc[j] = yvc[j-1]-dy1;
                        y[j-1] = (yvc[j-1]+yvc[j])/2;
                }
        }
}


void volume (double *xvc, double *yvc, int N, int M, matrix& V)
{
        for(int i = 0; i<N; i++)
        {
                for(int j = 0; j<M; j++)
                {
                        V[j][i] = fabs(xvc[i+1]-xvc[i])*fabs(yvc[j]-yvc[j+1]); // Volume
                }
        }
}


void surface (double *yvc, int M, double Sx[])
{
        for(int j = 0; j<M; j++)
        {
                Sx[j] = fabs(yvc[j]-yvc[j+1]);
        }
}


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)
{
        for(int i = 0; i<N1+N2; i++)
        {
                for(int j = 0; j<M1+M2+M3; j++)
                {
                        if(x[i]<=p[0][0] && y[j]<=p[0][1])
                        {
                                rho[j][i] = rhod[0];
                                cp[j][i] = cpd[0];
                                lambda[j][i] = lamd[0];
                        }
                        else if(x[i]<=p[0][0] && y[j]>p[0][1])
                        {
                                rho[j][i] = rhod[2];
                                cp[j][i] = cpd[2];
                                lambda[j][i] = lamd[2];
                        }
                        else if(x[i]>p[0][0] && y[j]<=p[1][1])
                        {
                                rho[j][i] = rhod[1];
                                cp[j][i] = cpd[1];
                                lambda[j][i] = lamd[1];
                        }
                        else
                        {
                                rho[j][i] = rhod[3];
                                cp[j][i] = cpd[3];
                                lambda[j][i] = lamd[3];
                        }
                }
        }
}


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)
{
        for(int i = 0; i<N; i++)
        {
                for(int j = 0; j<M; j++)
                {
                        if(i==0)
                        {
                                lambdaw[j][i] = lambda[j][i];
                                lambdan[j][i] = (y[j-1]-y[j])/((y[j-1]-yvc[j])/lambda[j-1][i]+(yvc[j]-
                                     y[j])/lambda[j][i]);
                                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]);
                                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]);
                        }
                        else if(i==N-1)
                        {
                                lambdaw[j][i] = (x[i]-x[i-1])/((x[i]-xvc[i])/lambda[j][i-1]+(x[i]-xvc[i
                    ])/lambda[j][i]);
                lambdan[j][i] = (y[j-1]-y[j])/((y[j-1]-yvc[j])/lambda[j-1][i]+(yvc[j]-
                    y[j])/lambda[j][i]);
                lambdae[j][i] = lambda[j][i];
                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]);
            }
            else if(j==0)
            {
                lambdaw[j][i] = (x[i]-x[i-1])/((x[i]-xvc[i])/lambda[j][i-1]+(x[i]-xvc[i
                    ])/lambda[j][i]);
                lambdan[j][i] = lambda[j][i];
                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]);
                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]);
            }
            else if(j==M-1)
            {
                lambdaw[j][i] = (x[i]-x[i-1])/((x[i]-xvc[i])/lambda[j][i-1]+(x[i]-xvc[i
                    ])/lambda[j][i]);
                lambdan[j][i] = (y[j-1]-y[j])/((y[j-1]-yvc[j])/lambda[j-1][i]+(yvc[j]-
                    y[j])/lambda[j][i]);
                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]);
                lambdas[j][i] = lambda[j][i];
            }
            else
            {
                lambdaw[j][i] = (x[i]-x[i-1])/((x[i]-xvc[i])/lambda[j][i-1]+(x[i]-xvc[i
                    ])/lambda[j][i]);
                lambdan[j][i] = (y[j-1]-y[j])/((y[j-1]-yvc[j])/lambda[j-1][i]+(yvc[j]-
                    y[j])/lambda[j][i]);
                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]);
                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]);
            }
        }
    }
}


// Searching the index of the node closest to a given point (and the second closest)
void search_index (float point, double *x, int Number, int &ipoint, int& ip)
{
    for(int i = 0; i<Number-1; i++)
  {
    if(x[i+1]-x[i]>0)
    {
            if(x[i]<=point && x[i+1]>point)
            {
                if(point-x[i]<x[i+1]-point)
                {
                    ipoint = i; //ipoint is the index of the node closest to the
                        point we want
                    ip = i+1; //ip is the second node closest to it (used in
                        interpolation )
                }
                else
                {
                    ipoint = i+1;
                    ip = i;
                }
            }
        }
        else
        {
            if(x[i]>point && x[i+1]<=point)
        {
            if(point-x[i+1]<x[i]-point)
            {
                ipoint = i;
                ip = i+1;
                }
                else
                {
                    ipoint = i+1;
                    ip = i;
                }
            }
        }
    }

}


// Calculation of the constant coefficients
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)
{
    for(int i =0; i<N1+N2; i++)
    {
        for(int j = 0; j<M1+M2+M3; j++)
        {
            if(i==0 && j==0)
            {
                ae[j][i] = beta*lambdae[j][i]*Sx[j]/(x[i+1]-x[i]);
                aw[j][i]  = 0;
                as[j][i]  = beta*lambdas[j][i]*Sy[i]/(y[j]-y[j+1]);
                an[j][i]  = 0;
                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]);
            }
            else if(i==0 && j!=0 && j!=M1+M2+M3-1)
            {
                ae[j][i]  = beta*lambdae[j][i]*Sx[j]/(x[i+1]-x[i]);
                aw[j][i]  = 0;
                as[j][i]  = beta*lambdas[j][i]*Sy[i]/(y[j]-y[j+1]);
                an[j][i]  = beta*lambdan[j][i]*Sy[i]/(y[j-1]-y[j]);
                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]);
            }
            else if(i==0 && j==M1+M2+M3-1)
            {
                ae[j][i]  = beta*lambdae[j][i]*Sx[j]/(x[i+1]-x[i]);
                aw[j][i]  = 0;
                as[j][i]  = 0;
                an[j][i]  = beta*lambdan[j][i]*Sy[i]/(y[j-1]-y[j]);
                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];
            }
            else if(i==N1+N2-1 && j==0)
            {
                ae[j][i]  = 0;
                aw[j][i]  = beta*lambdaw[j][i]*Sx[j]/(x[i]-x[i-1]);
                as[j][i]  = beta*lambdas[j][i]*Sy[i]/(y[j]-y[j+1]);
                an[j][i]  = 0;
                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]);
            }
            else if(i==N1+N2-1 && j==M1+M2+M3-1)
            {
                ae[j][i]  = 0;
                aw[j][i]  = beta*lambdaw[j][i]*Sx[j]/(x[i]-x[i-1]);
                as[j][i]  = 0;
                an[j][i]  = beta*lambdan[j][i]*Sy[i]/(y[j-1]-y[j]);
                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];
            }
            else if(i==N1+N2-1 && j!=0 && j!=M1+M2+M3-1)
            {
                ae[j][i]  = 0;
                aw[j][i]  = beta*lambdaw[j][i]*Sx[j]/(x[i]-x[i-1]);
                as[j][i]  = beta*lambdas[j][i]*Sy[i]/(y[j]-y[j+1]);
                an[j][i]  = beta*lambdan[j][i]*Sy[i]/(y[j-1]-y[j]);
```

```
                    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]);
                }
                else if(i!=0 && i!=N1+N2-1 && j==0)
                {
                    ae[j][i] = beta*lambdae[j][i]*Sx[j]/(x[i+1]-x[i]);
                    aw[j][i] = beta*lambdaw[j][i]*Sx[j]/(x[i]-x[i-1]);
                    as[j][i] = beta*lambdas[j][i]*Sy[i]/(y[j]-y[j+1]);
                    an[j][i] = 0;
                    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;
                }
                else if(i!=0 && i!=N1+N2-1 && j==M1+M2+M3-1)
                {
                    ae[j][i] = beta*lambdae[j][i]*Sx[j]/(x[i+1]-x[i]);
                    aw[j][i] = beta*lambdaw[j][i]*Sx[j]/(x[i]-x[i-1]);
                    as[j][i] = 0;
                    an[j][i] = beta*lambdan[j][i]*Sy[i]/(y[j-1]-y[j]);
                    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]);
                }
                else
                {
                    ae[j][i] = beta*lambdae[j][i]*Sx[j]/(x[i+1]-x[i]);
                    aw[j][i] = beta*lambdaw[j][i]*Sx[j]/(x[i]-x[i-1]);
                    as[j][i] = beta*lambdas[j][i]*Sy[i]/(y[j]-y[j+1]);
                    an[j][i] = beta*lambdan[j][i]*Sy[i]/(y[j-1]-y[j]);
                    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;
                }
            }
        }
}


// Calculation of non-constant coefficients
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)
{
        for(int i =0; i<N1+N2; i++)
            {
                for(int j = 0; j<M1+M2+M3; j++)
                {
                    if(i==0 && j==0)
                    {
                        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];
        }
        else if(i==0 && j!=0 && j!=M1+M2+M3−1)
        {
              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];
        }
        else if(i==0 && j==M1+M2+M3−1)
        {
              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];
        }
        else if(i==N1+N2−1 && j==0)
        {
              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];
        }
        else if(i==N1+N2−1 && j==M1+M2+M3−1)
        {
              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];
        }
        else if(i==N1+N2−1 && j!=0 && j!=M1+M2+M3−1)
        {
              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];
                        }
                        else if(i!=0 && i!=N1+N2−1 && j==0)
                        {
                        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];
                        }
                        else if(i!=0 && i!=N1+N2−1 && j==M1+M2+M3−1)
                        {
                                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];
                        }
                        else
                        {
                                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];
                        }
                }
        }
}


// Solver (using Gauss−Seidel)
void Gauss_Seidel (matrix ap, matrix aw, matrix ae, matrix as, matrix an, matrix bp, float fr, float
    delta, int N, int M, matrix& T)
{
    double Tcalc[M][N]; // Temperature calculated in the previous iteration
    for(int i = 0; i<N; i++)
    {
        for(int j = 0; j<M; j++)
        {
            Tcalc[j][i] = T[j][i];
        }
    }

    double MAX = 1; // Maximum value of the difference between T and Tcalc
    double resta = 1; // Difference between T and Tcalc

    while(MAX>delta)
    {

        // SOLVER: Gauss−Seidel
        for(int i = 0; i<N; i++)
        {
            for(int j = 0; j<M; j++)
            {
                if(i==0 && j==0)
                {
                    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]);
                }
                else if(i==0 && j==M−1)
                {
                    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]);
                }
                else if(i==0 && j!=0 && j!=M−1)
                {
                    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]);
                }
                else if(i==N−1 && j==0)
                {
                    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]);
                }
                else if(i==N−1 && j==M−1)
                {
                    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]);
                }
                else if(i==N && j!=0 && j!=M−1)
                {
                    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])
                        ;
                }
                else if(i!=0 && i!=N−1 && j==0)
                {
                    T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i−1]+ae[j][i]*Tcalc[j
                        ][i+1]+as[j][i]*Tcalc[j+1][i]+bp[j][i])/ap[j][i]−Tcalc[j
                        ][i]);
                }
                else if(i!=0 && i!=N−1 && j==M−1)
```

```
                    {
                        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])
                            ;
                    }
                    else
                    {
                        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]);
                    }
                }
            }

            // Comprovation
            MAX = 0;
            for(int i = 0; i<N; i++)
            {
                for(int j = 0; j<M; j++)
                {
                    resta = fabs(Tcalc[j][i]-T[j][i]);

                    if(resta>MAX)
                    {
                        MAX = resta;
                    }
                }
            }

            // New assignation
            for(int i = 0; i<N; i++)
            {
                for(int j = 0; j<M; j++)
                {
                    Tcalc[j][i] = T[j][i];
                }
            }
        }
}


// Double interpolation
double double_interpolation (float x, float y, double T11, double T12, double T21, double T22,
    double x1, double x2, double y1, double y2)
{
    double result1, result2, finalresult;
    result1 = T11+(T21-T11)*(x-x1)/(x2-x1);
    result2 = T12+(T22-T12)*(x-x1)/(x2-x1);
    finalresult = result1+(result2-result1)*(y-y1)/(y2-y1);
    return finalresult;
```

```
}


// Print matrix
void print_matrix (matrix T, int N, int M)
{
        for(int j = 0; j<M; j++)
    {
        for(int i = 0; i<N; i++)
        {
            cout<<T[j][i]<<"   "; // display the current element out of the array
        }
                cout<<endl; // go to a new line
    }
}

// Create an output file with the results
void output_file (double* Tpoint1, double* Tpoint2, int Time, float dt)
{
        ofstream puntss;
    puntss.open("Punts.dat");
    float t = 0;
    for(int k = 0; k<Time; k++)
    {
        puntss<<t<<" "<<Tpoint1[k]<<"      "<<Tpoint2[k]<<"\n";
        t = t+dt;
        }
    puntss.close();
}
```

2 | Smith-Hutton problem

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

using namespace std;

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

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


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

```


int main(){

        cout<<"Program started"<<endl<<endl;

        // DATA
        float alpha = 10; // Angle [degrees]
        float rho = 1; // Density
        float Sc = 0; // Source term = Sc+Sp*phi
        float Sp = 0;
        string method = "EDS";

        float delta = 0.000000001; // Precision of the simulation
        float fr = 1.1; // Relaxation factor


        // PREVIOUS CALCULATIONS

        // Increments
        float dx, dy, dt;
        dx = 2.0/N;
        dy = 1.0/M;
        dt = 1;

        // Coordinates
        float xvc[N+1], yvc[M+1]; // Coordinates of the faces
        float x[N+2], y[M+2]; // Coordinates of the nodes


        coordinates(-1, 1, dx, N+1, xvc, x);
        coordinates(1, 0, -dy, M+1, yvc, y);


        // Surfaces and volumes
        float Sh[N+2], Sv[M+2];
        matrix V;
        surface(yvc, M+2, Sv);
        surface(xvc, N+2, Sh);
        volume(xvc, yvc, N+2, M+2, V);


        // Mass flow on the faces
        mface mflowx, mflowy;
        mass_flow(rho, N+1, M+1, Sv, Sh, xvc, yvc, mflowx, mflowy);


        // Boundary conditions
        double phi_boundary, phis[N+1];
        phi_inlet_outlet(x, alpha, N+2, phis);
        phi_boundary = 1-tanh(alpha);


        // Output coordinates
        float  xfinal [11];
        int  index [2][11];
        xfinal [0]  = 0;
        for(int  i  = 0;  i<11;  i++)
        {
                if(i==0)
                {
                        xfinal [i]  = 0;
                }
                else
                {
                        xfinal [i]  = xfinal[i-1]+0.1;
                }
                search_index ( xfinal [i],  x,  N+2, index[0][i],  index [1][ i]) ;
        }
        index [0][10]  = N+2;


        // Resolution
        float  gamma;
        double phi1 [11],  phi2 [11],  phi3 [11];

        cout<<"Solving rho/gamma = 10..."<<endl;
        gamma = rho/10;
        solver (method, rho, gamma, dt, fr,  delta ,  Sp,  Sc,  phi_boundary, phis ,  x,  y,  Sh, Sv, V, mflowx,
              mflowy,  xfinal ,  index ,  phi1) ;


        cout<<"Solving rho/gamma = 1000..."<<endl;
        gamma = rho/1000;
        solver (method, rho, gamma, dt, fr,  delta ,  Sp,  Sc,  phi_boundary, phis ,  x,  y,  Sh, Sv, V, mflowx,
              mflowy,  xfinal ,  index ,  phi2) ;

        cout<<"Solving rho/gamma = 1000000..."<<endl;
        gamma = rho/1000000;
        solver (method, rho, gamma, dt, fr,  delta ,  Sp,  Sc,  phi_boundary, phis ,  x,  y,  Sh, Sv, V, mflowx,
              mflowy,  xfinal ,  index ,  phi3) ;


        cout<<endl<<"Creating an output file..."<<endl;
        ofstream  results ;
    results .open("Resultats.dat") ;
    for(int  k = 0; k<11; k++)
    {
         results <<xfinal[k]<<"  "<<phi1[k]<<" "<<phi2[k]<<" "<<phi3[k]<<"\n";
        }
    results . close ();

   return 0;
}



void coordinates(float x0, float xN, float dx, int N, float xvc [], float x [])
{
        xvc [0] = x0;
        x [0] = xvc [0];
        for(int i = 0; i<N; i++)
        {
                xvc[ i+1] = xvc[i]+dx;
                x[ i+1] = (xvc[i+1]+xvc[i])/2;
        }
        x[N] = xN;
        xvc [0] = x0+dx/2;
        xvc[N-1] = xN-dx/2;
}


void surface(float *yvc, int M, float Sv [])
{
        for(int j = 0; j<M; j++)
        {
                Sv[j] = fabs(yvc[j]-yvc[j+1]);
                if(j==M-1)
                {
                        Sv[j] = Sv[j-1];
                }
        }
}


void volume(float *xvc, float *yvc, int N, int M, matrix& V)
{
        for(int i = 0; i<N; i++)
        {
                for(int j = 0; j<M; j++)
                {
                        V[j][i] = fabs(xvc[i]-xvc[i-1])*fabs(yvc[j-1]-yvc[j]);
                }
        }
}


void velocity (float *x, float *y, int N, int M, mface& u, mface& v)
{
        for(int i = 0; i<N; i++)
```

```
		{
			for(int j = 0; j<M; j++)
			{
				u[j][i] = 2*y[j]*(1-pow(x[i],2));
				v[j][i] = -2*x[i]*(1-pow(y[j],2));
			}
		}
}


void mass_flow(float rho, int N, int M, float *Sv, float *Sh, float *xvc, float *yvc, mface& mflowx,
     mface& mflowy)
{
		for(int i = 0; i<N; i++)
		{
			for(int j = 0; j<M; j++)
			{
				mflowx[j][i] = rho*Sv[j]*2*yvc[j]*(1-pow(xvc[i],2));
				mflowy[j][i] = -rho*Sh[i]*2*xvc[i]*(1-pow(yvc[j],2));
			}
		}
}


void phi_inlet_outlet(float *x, float alpha, int N, double phis[])
{
		for(int i = 0; i<N; i++)
		{
			if(x[i]<=0)
			{
				phis[i] = 1+tanh(alpha*(2*x[i]+1));
			}
			else
			{
				phis[i] = 0;
			}
		}
}


// Searching the index of the node closest to a given point (and the second closest)
void search_index(float point, float *x, int Number, int& ipoint, int& ip)
{
		for(int i = 0; i<Number-1; i++)
	{
		if(x[i+1]-x[i]>0)
		{
			if(x[i]<=point && x[i+1]>point)
			{
				if(point-x[i]<x[i+1]-point)
```

```
                    {
                        ipoint = i; //ipoint is the index of the node closest to the
                           point we want
                        ip = i+1; //ip is the second node closest to it (used in
                           interpolation )
                    }
                    else
                    {
                        ipoint = i+1;
                        ip = i;
                    }
                }
            }
            else
            {
                if(x[i]>point && x[i+1]<=point)
            {
                if(point-x[i+1]<x[i]-point)
                {
                    ipoint = i;
                    ip = i+1;
                    }
                    else
                    {
                        ipoint = i+1;
                        ip = i;
                    }
                }
            }
        }

}


double max(double a, double b)
{
        if(a>b)
        {
            return a;
        }
        else
        {
            return b;
        }
}


double Aperator(string method, double P)
{
        double A;
        if(method=="CDS") // Central Differencing Scheme
        {
                A = 1-0.5*fabs(P);
        }
        else if(method=="UDS") // Upwind Differencing Scheme
        {
                A = 1;
        }
        else if(method=="HDS") // Hybrid Differencing Scheme
        {
                A = max(0,1-0.5*fabs(P));
        }
        else if(method=="PLDS") // Power Law Differencing Scheme
        {
                A = max(0,pow(1-0.1*fabs(P),5));
        }
        else if(method=="EDS") // Exponential Differencing Scheme
        {
                A = fabs(P)/(exp(fabs(P))-1);
        }
        return A;
}


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)
{
        double De, Dw, Dn, Ds;
        double Pe, Pw, Pn, Ps;
        double Fe, Fw, Fn, Fs;

        for(int i = 0; i<N; i++)
        {
                for(int j = 0; j<M; j++)
                {
                        if(j==M-1 && x[i]>=0)
                        {
                                ae[j][i] = 0;
                                aw[j][i] = 0;
                                an[j][i] = 1;
                                as[j][i] = 0;
                                ap[j][i] = 1;
                        }
                        else if(j==M-1 && x[i]<0)
                        {
                                ae[j][i] = 0;
                                aw[j][i] = 0;
                                an[j][i] = 0;
                                as[j][i] = 0;
                ap[j][i] = 1;
            }
            else if(i==0)
            {
                ae[j][i] = 0;
                aw[j][i] = 0;
                an[j][i] = 0;
                as[j][i] = 0;
                ap[j][i] = 1;
            }
            else if(j==0)
            {
                ae[j][i] = 0;
                aw[j][i] = 0;
                an[j][i] = 0;
                as[j][i] = 0;
                ap[j][i] = 1;
            }
            else if(i==N-1)
            {
                ae[j][i] = 0;
                aw[j][i] = 0;
                an[j][i] = 0;
                as[j][i] = 0;
                ap[j][i] = 1;
            }
            else
            {
                De = gamma*Sh[i]/fabs(x[i+1]-x[i]);
                Dw = gamma*Sh[i-1]/fabs(x[i]-x[i-1]);
                Dn = gamma*Sv[j-1]/fabs(y[j-1]-y[j]);
                Ds = gamma*Sv[j]/fabs(y[j]-y[j+1]);
                Fe = mflowx[j-1][i];
                Fw = mflowx[j-1][i-1];
                Fn = mflowy[j-1][i-1];
                Fs = mflowy[j][i-1];
                Pe = Fe/De;
                Pw = Fw/Dw;
                Pn = Fn/Dn;
                Ps = Fs/Ds;
                ae[j][i] = De*Aperator(method,Pe)+max(-Fe,0);
                aw[j][i] = Dw*Aperator(method,Pw)+max(Fw,0);
                an[j][i] = Dn*Aperator(method,Pn)+max(-Fn,0);
                as[j][i] = Ds*Aperator(method,Ps)+max(Fs,0);
                ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i]+rho0*V[j][i]/dt-Sp*V[j
                    ][i];
            }
        }
    }
}
```

```


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)
{
        for(int i = 0; i<N; i++)
        {
                for(int j = 0; j<M; j++)
                {
                        if(j==M-1 && x[i]>=0)
                        {
                                bp[j][i] = 0;
                        }
                        else if(j==M-1 && x[i]<0)
                        {
                                bp[j][i] = phis[i];
                        }
                        else if(i==0)
                        {
                                bp[j][i] = phi_boundary;
                        }
                        else if(j==0)
                        {
                                bp[j][i] = phi_boundary;
                        }
                        else if(i==N-1)
                        {
                                bp[j][i] = phi_boundary;
                        }
                        else
                        {
                                bp[j][i] = rho0*V[j][i]*phi0[j][i]/dt+Sc*V[j][i];
                        }
                }
        }
}


// Solver (using Gauss-Seidel)
void Gauss_Seidel (matrix ap, matrix aw, matrix ae, matrix as, matrix an, matrix bp, float fr, float
        delta, int N, int M, matrix& T)
{
        double Tcalc[M][N]; // Temperature calculated in the previous iteration
        for(int i = 0; i<N; i++)
        {
                for(int j = 0; j<M; j++)
                {
                        Tcalc[j][i] = T[j][i];
                }
        }
```

```

    double MAX = 1; // Maximum value of the difference between T and Tcalc
    double resta = 1; // Difference between T and Tcalc

    while(MAX>delta)
    {
        // SOLVER: Gauss-Seidel
        for(int i = 0; i<N; i++)
        {
            for(int j = 0; j<M; j++)
            {
                if(i==0 && j==0)
                {
                    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]);
                }
                else if(i==0 && j==M-1)
                {
                    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]);
                }
                else if(i==0 && j!=0 && j!=M-1)
                {
                    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]);
                }
                else if(i==N-1 && j==0)
                {
                    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]);
                }
                else if(i==N-1 && j==M-1)
                {
                    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]);
                }
                else if(i==N && j!=0 && j!=M-1)
                {
                    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]);
                }
                else if(i!=0 && i!=N-1 && j==0)
                {
                    T[j][i] = Tcalc[j][i]+fr*((aw[j][i]*T[j][i-1]+ae[j][i]*Tcalc[j][i+1]+as[j][i]*Tcalc[j+1][i]+bp[j][i])/ap[j][i]-Tcalc[j][i]);
                }
                else if(i!=0 && i!=N-1 && j==M-1)
```

```
                                {
                                        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])
                                            ;
                                }
                                else
                                {
                                        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]);
                                }
                        }
                }
                // Comprovation
                MAX = 0;
                for(int i = 0; i<N; i++)
                {
                        for(int j = 0; j<M; j++)
                        {
                                resta = fabs(Tcalc[j][i]-T[j][i]);

                                if(resta>MAX)
                                {
                                        MAX = resta;
                                }
                        }
                }
                // New assignation
                for(int i = 0; i<N; i++)
                {
                        for(int j = 0; j<M; j++)
                        {
                                Tcalc[j][i] = T[j][i];
                        }
                }
        }
}


double interpolation(float x, double T1, double T2, double x1, double x2)
{
        double result;
         result = T1+(T2-T1)*(x-x1)/(x2-x1);
        return result;
}


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])
```

```
{
        matrix phi,phi0;

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


        matrix ae, aw, an, as, ap, bp;
        constant_coefficients (N+2, M+2, method, rho, gamma, dt, Sp, x, y, Sh, Sv, V, mflowx, mflowy,
            ae, aw, an, as, ap);


        float resta = 1;

        while(resta>delta)
        {
                //New increment of time
                for(int i = 0; i<N+2; i++)
                {
                        for(int j = 0; j<M+2; j++)
                        {
                                phi0[j][i] = phi[j][i];
                        }
                }

                bp_coefficient (N+2, M+2, rho, dt, Sc, x, phi_boundary, phis, phi0, V, bp);
                Gauss_Seidel (ap, aw, ae, as, an, bp, fr, delta, N+2, M+2, phi);

                resta = 0;
                for(int i = 0; i<N+2; i++)
                {
                        for(int j = 0; j<M+2; j++)
                        {
                                resta = max(resta, fabs(phi[j][i]-phi0[j][i]));
                        }
                }
        }

        for(int i = 0; i<11; i++)
        {
                phi1[i] = interpolation(xfinal[i], phi[M+1][index[0][i]], phi[M+1][index[1][i]], x[
                    index[0][i]], x[index[1][i]]);
        }
}
```

3 | Driven cavity problem

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

using namespace std;

// Numerical parameters
const int N = 112;
const int M = 112;

typedef double matrix[M+2][N+2];
typedef double staggx[M+2][N+1];
typedef double staggy[M+1][N+2];

void coordinates(float L, int N, double xvc [], double x[]) ;
void surface(double *yvc, int M, double Sv[]);
void volume(double *xvc, double *yvc, int N, int M, matrix& V);
void initial_conditions (int N, int M, float uref, staggx& u0, staggx& Ru0, staggy& v0, staggy& Rv0);
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);
double convective_term (double xf, double x2, double x3, double u2, double u3);
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);
void bp_coefficient (int N, int M, float rho, double dt, double* Sh, double* Sv, staggx up, staggy vp,
      matrix bp);
void Gauss_Seidel (matrix ap, matrix aw, matrix ae, matrix as, matrix an, matrix bp, float fr, float
     delta, int N, int M, matrix& T);
void 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);
double min(double a, double b);
double max(double a, double b);
double time_step (double dtd, double* x, double* y, staggx u, staggy v);
double error (int N, int M, staggx u, staggy v, staggx u0, staggy v0);
void search_index (float point, double *x, int Number, int& ipoint, int& ip);
double interpolation (float x, double T1, double T2, double x1, double x2);
void output_files (int N, int M, float L, double* x, double* y, double* xvc, double* yvc, staggx u,
     staggy v);
```

```


int main()
{
        int Re = 10000; // Reynolds number
        float L = 1; // Length of the cavity
        float rho = 1; // Density
        float uref = 1; // Reference velocity
        double mu = rho*uref*L/Re; // Viscosity

        float delta = 2e-4; // Precision of the simulation (as the Re increases it is recommended to
            use 5e-5, 1e-4, 2e-4...)
        float fr = 1.2; // Relaxation factor

        cout<<"Program started"<<endl;
        cout<<"Re="<<Re<<endl<<endl;

        // Coordinates
        double xvc[N+1], yvc[M+1], x[N+2], y[M+2];
        coordinates(L, N, xvc, x);
        coordinates(L, M, yvc, y);

        // Surfaces
        double Sh[N+2], Sv[M+2];
        matrix V;
        surface(xvc, N+2, Sh); // Horizontal surface
        surface(yvc, M+2, Sv); // Vertical surface
        volume(xvc, yvc, N+2, M+2, V); // Volume


        // Properties that are going to be calculated
        matrix p; // Values in the nodes (pressure)
        staggx u, u0, Ru0; // Values in the points given by the staggered meshes ( velocities )
        staggy v, v0, Rv0;

        //  Inicialization
         initial_conditions (N, M, uref, u0, Ru0, v0, Rv0);

        // Calculation of the constant coefficients that are used to determine the pressure
        matrix ae, aw, an, as, ap, bp;
         constant_coefficients (N+2, M+2, x, y, Sv, Sh, ae, aw, an, as, ap);

        // Time step (CFL condition)
        double resta = 1;
        double dtd = 0.2*rho*pow(x[2]-xvc[1],2)/mu;
        double dtc = 0.35*fabs(x[2]-xvc[1])/uref;
        double dt = min(dtd, dtc);

        staggx up, Ru; // Intermediate velocities
        staggy vp, Rv;
```

```

    cout<<"Solving..."<<endl;
    // Fractional Step Method
    while(resta>delta)
    {
        // STEP 1: INTERMEDIATE VELOCITY
        intermediate_velocities (N, M, rho, mu, delta, dt, x, y, xvc, yvc, Sh, Sv, V, u0, v0,
            Ru0, Rv0, Ru, Rv, up, vp);


        // STEP 2: PRESSURE
        bp_coefficient (N+2, M+2, rho, dt, Sh, Sv, up, vp, bp);
        Gauss_Seidel (ap, aw, ae, as, an, bp, fr, delta, N+2, M+2, p);


        // STEP 3: VELOCITY
        velocities (N, M, rho, dt, uref, x, y, p, up, vp, u, v);


        // STEP 4: TIME STEP
        dt = time_step (dtd, x, y, u, v);


        // Comprovation
        resta = error (N, M, u, v, u0, v0);

        // New time step
        for(int i = 0; i<N+1; i++)
        {
            for(int j = 0; j<M+2; j++)
            {
                u0[j][i] = u[j][i];
                Ru0[j][i] = Ru[j][i];
            }
        }
        for(int i = 0; i<N+2; i++)
        {
            for(int j = 0; j<M+1; j++)
            {
                v0[j][i] = v[j][i];
                Rv0[j][i] = Rv[j][i];
            }
        }
    }

    // Results
cout<<endl<<"Creating some output files..."<<endl;
output_files (N, M, L, x, y, xvc, yvc, u, v);

    return 0;
```

```
}


// Coordinates of the control volumes (x -> nodes, xvc -> faces)
void coordinates(float L, int N, double xvc[], double x[])
{
        double dx = L/N;
        xvc[0] = 0;
        x[0] = 0;
        for(int i = 0; i<N; i++)
        {
                xvc[i+1] = xvc[i]+dx;
                x[i+1] = (xvc[i+1]+xvc[i])/2;
        }
        x[N+1] = L;
}


// Surfaces of the control volumes
void surface(double *yvc, int M, double Sv[])
{
        for(int j = 0; j<M-1; j++)
        {
                Sv[j+1] = fabs(yvc[j]-yvc[j+1]);
        }
        Sv[0] = 0;
        Sv[M-1] = 0;
}


// Volume of each control volume
void volume(double *xvc, double *yvc, int N, int M, matrix& V)
{
        for(int i = 0; i<N; i++)
        {
                for(int j = 0; j<M; j++)
                {
                        if(i==N-1 || j==M-1 || i==0 || j==0)
                        {
                                V[j][i] = 0;
                        }
                        else
                        {
                                V[j][i] = fabs(xvc[i]-xvc[i-1])*fabs(yvc[j]-yvc[j-1]);
                        }
                }
        }
}
```

```
// Initial conditions of the problem
void initial_conditions (int N, int M, float uref, staggx& u0, staggx& Ru0, staggy& v0, staggy& Rv0)
{
        for(int j = 0; j<M+2; j++)
        {
                for(int i = 0; i<N+1; i++)
                {
                        if(j==M+1 && i!=0 && i!=N)
                        {
                                u0[j][i] = uref; // Horizontal velocity at n
                        }
                        else
                        {
                                u0[j][i] = 0; // Horizontal velocity at n
                        }
                        Ru0[j][i] = 0; // R (horizontal) at n-1
                }
        }
        for(int j = 0; j<M+1; j++)
        {
                for(int i = 0; i<N+2; i++)
                {
                        v0[j][i] = 0; // Vertical velocity at n
                        Rv0[j][i] = 0; // R (vertical) at n-1
                }
        }
}


// Calculation of the constant coefficients (ae, aw, an, as, ap) of the Poisson equation (pressure)
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)
{
        for(int i = 0; i<N; i++)
        {
                for(int j = 0; j<M; j++)
                {
                        if(j==M-1 && i!=0 && i!=N-1)
                        {
                                ae[j][i] = 0;
                                aw[j][i] = 0;
                                an[j][i] = 0;
                                as[j][i] = 1;
                                ap[j][i] = 1;
                        }
                        else if(i==0 && j==0)
                        {
                                ae[j][i] = 1;
                                aw[j][i] = 0;
                                an[j][i] = 1;
        as[j][i] = 0;
        ap[j][i] = 1;
    }
    else if(i==0 && j==M-1)
    {
        ae[j][i] = 1;
        aw[j][i] = 0;
        an[j][i] = 0;
        as[j][i] = 1;
        ap[j][i] = 1;
    }
    else if(i==0 && j!=0 && j!=M-1)
    {
        ae[j][i] = 1;
        aw[j][i] = 0;
        an[j][i] = 0;
        as[j][i] = 0;
        ap[j][i] = 1;
    }
    else if(i==N-1 && j==0)
    {
        ae[j][i] = 0;
        aw[j][i] = 1;
        an[j][i] = 1;
        as[j][i] = 0;
        ap[j][i] = 1;
    }
    else if(i==N-1 && j==M-1)
    {
        ae[j][i] = 0;
        aw[j][i] = 1;
        an[j][i] = 0;
        as[j][i] = 1;
        ap[j][i] = 1;
    }
    else if(i==N-1 && j!=0 && j!=M-1)
    {
        ae[j][i] = 0;
        aw[j][i] = 1;
        an[j][i] = 0;
        as[j][i] = 0;
        ap[j][i] = 1;
    }
    else if(j==0 && i!=0 && i!=N-1)
    {
        ae[j][i] = 0;
        aw[j][i] = 0;
        an[j][i] = 1;
        as[j][i] = 0;
        ap[j][i] = 1;
```

```
                        }
                        else
                        {
                                ae[j][i] = Sv[j]/fabs(x[i+1]-x[i]);
                                aw[j][i] = Sv[j]/fabs(x[i]-x[i-1]);
                                an[j][i] = Sh[i]/fabs(y[j+1]-y[j]);
                                as[j][i] = Sh[i]/fabs(y[j]-y[j-1]);
                                ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i];
                        }
                }
        }
}


// Computation of the velocity in the convective term using CDS
double convective_term (double xf, double x2, double x3, double u2, double u3)
{
        // 2 refers to node P, 3 to node E
        double u;
        u = u2+fabs(x2-xf)*(u3-u2)/fabs(x3-x2);

        return u;
}


// Calculation of the intermediate velocities
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)
{
        double mflowe, mfloww, mflown, mflows;
        double ue, uw, un, us;
        double De, Dw, Dn, Ds;

        for(int i = 0; i<N+1; i++)
        {
                for(int j = 0; j<M+2; j++)
                {
                        // Mass flow terms (rho*v*S)
                        mflowe = (rho*u0[j][i+1]+rho*u0[j][i])*Sv[j]/2;
                        mfloww = (rho*u0[j][i-1]+rho*u0[j][i])*Sv[j]/2;
                        mflown = (rho*v0[j][i]+rho*v0[j][i+1])*Sh[i]/2;
                        mflows = (rho*v0[j-1][i]+rho*v0[j-1][i+1])*Sh[i]/2;


                        // HORIZONTAL
                        ue = convective_term (x[i+1], xvc[i], xvc[i+1], u0[j][i], u0[j][i+1]);
                        uw = convective_term (x[i], xvc[i], xvc[i-1], u0[j][i], u0[j][i-1]);
                        un = convective_term (yvc[j], y[j], y[j+1], u0[j][i], u0[j+1][i]);
                        us = convective_term (yvc[j-1], y[j], y[j-1], u0[j][i], u0[j-1][i]);
```

```

                    De = mu*Sv[j]/fabs(xvc[i+1]-xvc[i]);
                    Dw = mu*Sv[j]/fabs(xvc[i]-xvc[i-1]);
                    Dn = mu*Sh[i]/fabs(y[j+1]-y[j]);
                    Ds = mu*Sh[i]/fabs(y[j]-y[j-1]);

                    // R (horizontal)
                    if(i==0 || i==N || j==0 || j==M+1)
                    {
                            Ru[j][i] = 0;
                    }
                    else
                    {
                            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];
                    }

                    // Intermediate velocity (horizontal)
                    up[j][i] = u0[j][i]+dt*(1.5*Ru[j][i]-0.5*Ru0[j][i])/rho;
            }
    }

    double ve, vw, vn, vs;

    for(int i = 0; i<N+2; i++)
    {
            for(int j = 0; j<M+1; j++)
            {
                    // Mass flow terms (rho*v*S)
                    mflowe = (rho*u0[j+1][i]+rho*u0[j][i])*Sv[j]/2;
                    mfloww = (rho*u0[j+1][i-1]+rho*u0[j][i-1])*Sv[j]/2;
                    mflown = (rho*v0[j][i]+rho*v0[j+1][i])*Sh[i]/2;
                    mflows = (rho*v0[j][i]+rho*v0[j-1][i])*Sh[i]/2;


                    // VERTICAL
                    ve = convective_term (xvc[i], x[i], x[i+1], v0[j][i], v0[j][i+1]);
                    vw = convective_term (xvc[i-1], x[i], x[i-1], v0[j][i], v0[j][i-1]);
                    vn = convective_term (y[j+1], yvc[j], yvc[j+1], v0[j][i], v0[j+1][i]);
                    vs = convective_term (y[j], yvc[j], yvc[j-1], v0[j][i], v0[j-1][i]);

                    De = mu*Sv[j]/fabs(x[i+1]-x[i]);
                    Dw = mu*Sv[j]/fabs(x[i]-x[i-1]);
                    Dn = mu*Sh[i]/fabs(yvc[j+1]-yvc[j]);
                    Ds = mu*Sh[i]/fabs(yvc[j]-yvc[j-1]);

                    // R (vertical)
                    if(i==0 || i==N+1 || j==0 || j==M)
                    {
```

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

                // Intermediate velocity (vertical)
                vp[j][i] = v0[j][i]+dt*(1.5*Rv[j][i]-0.5*Rv0[j][i])/rho;
            }
    }
}


// Calculation of the bp coefficient of the Poisson equation (pressure)
void bp_coefficient (int N, int M, float rho, double dt, double* Sh, double* Sv, staggx up, staggy vp,
        matrix bp)
{
    for(int i = 0; i<N; i++)
    {
        for(int j = 0; j<M; j++)
        {
            if(i==0 || j==0 || i==N-1 || j==M-1)
            {
                bp[j][i] = 0;
            }
            else
            {
                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;
            }
        }
    }
}


// Solver (using Gauss-Seidel)
void Gauss_Seidel (matrix ap, matrix aw, matrix ae, matrix as, matrix an, matrix bp, float fr, float
        delta, int N, int M, matrix& T)
{
    double Tcalc[M][N]; // Temperature calculated in the previous iteration
    for(int i = 0; i<N; i++)
    {
        for(int j = 0; j<M; j++)
        {
            Tcalc[j][i] = T[j][i];
        }
    }
```

```

        double MAX = 1; // Maximum value of the difference between T and Tcalc
        double resta = 1; // Difference between T and Tcalc

        while(MAX>delta)
        {

                // SOLVER: Gauss-Seidel
                for(int i = 0; i<N; i++)
                {
                        for(int j = 0; j<M; j++)
                        {
                                if(i==0 && j==M-1)
                                {
                                        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]);
                                }
                                else if(i==0 && j==0)
                                {
                                        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]);
                                }
                                else if(i==0 && j!=0 && j!=M-1)
                                {
                                        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]);
                                }
                                else if(i==N-1 && j==M-1)
                                {
                                        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]);
                                }
                                else if(i==N-1 && j==0)
                                {
                                        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]);
                                }
                                else if(i==N-1 && j!=0 && j!=M-1)
                                {
                                        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])
                                            ;
                                }
                                else if(i!=0 && i!=N-1 && j==M-1)
                                {
                                        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])
                                            ;
                                }
                                else if(i!=0 && i!=N-1 && j==0)
                                {
                                        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]);
                                }
                                else
                                {
                                        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]);
                                }
                        }
                }

                // Comprovation
                MAX = 0;
                for(int i = 0; i<N; i++)
                {
                        for(int j = 0; j<M; j++)
                        {
                                resta = fabs(Tcalc[j][i]-T[j][i]);

                                if(resta>MAX)
                                {
                                        MAX = resta;
                                }
                        }
                }

                // New assignation
                for(int i = 0; i<N; i++)
                {
                        for(int j = 0; j<M; j++)
                        {
                                Tcalc[j][i] = T[j][i];
                        }
                }
        }
}


// Calculation of the velocity with the correction of pressure
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)
{
        // Horizontal velocity at n+1
        for(int i = 0; i<N+1; i++)
        {
                for(int j = 0; j<M+2; j++)
                {
                        if(i==0 || i==N || j==0)
                        {
                                u[j][i] = 0;
                        }
                        else if(j==M+1)
                        {
                                u[j][i] = uref;
                        }
                        else
                        {
                                u[j][i] = up[j][i]-dt*(p[j][i+1]-p[j][i])/(rho*fabs(x[i+1]-x[i]));
                        }
                }
        }

        // Vertical velocity at n+1
        for(int i = 0; i<N+2; i++)
        {
                for(int j = 0; j<M+1; j++)
                {
                        if(j==0 || j==M || i==0 || i==N+1)
                        {
                                v[j][i] = 0;
                        }
                        else
                        {
                                v[j][i] = vp[j][i]-dt*(p[j+1][i]-p[j][i])/(rho*fabs(y[j+1]-y[j]));
                        }
                }
        }
}


// Returns the minimum value
double min(double a, double b)
{
        if(a>b)
        {
                return b;
        }
        else
        {
                return a;
        }
}


// Returns the maximum value
double max(double a, double b)
```

```
{
	if(a>b)
	{
		return a;
	}
	else
	{
		return b;
	}
}


// Calculation of the proper time step (CFL condition)
double time_step (double dtd, double* x, double* y, staggx u, staggy v)
{
	double dt;
	double dtc = 100;

	for(int i = 1; i<N; i++)
	{
		for(int j = 1; j<M+1; j++)
		{
			dtc = min(dtc, 0.35*fabs(x[i+1]-x[i])/fabs(u[j][i]));
		}
	}
	for(int i = 1; i<N+1; i++)
	{
		for(int j = 1; j<M; j++)
		{
			dtc = min(dtc, 0.35*fabs(y[j+1]-y[j])/fabs(v[j][i]));
		}
	}
	dt = min(dtc, dtd);
	return dt;
}


// Difference between the previous and the actual time step
double error (int N, int M, staggx u, staggy v, staggx u0, staggy v0)
{
	double resta = 0;
	for(int i = 0; i<N+1; i++)
	{
		for(int j = 0; j<M+2; j++)
		{
			resta = max(resta, fabs(u[j][i]-u0[j][i]));
		}
	}
	for(int i = 0; i<N+2; i++)
	{
```

```
            for(int j = 0; j<M+1; j++)
            {
                resta = max(resta, fabs(v[j][i]-v0[j][i]));
            }
        }
        return resta;
}


// Searching the index of the node closest to a given point (and the second closest)
void search_index (float point, double *x, int Number, int& ipoint, int& ip)
{
        for(int i = 0; i<Number-1; i++)
    {
        if(x[i+1]-x[i]>0)
        {
            if(x[i]<=point && x[i+1]>point)
            {
                if(point-x[i]<x[i+1]-point)
                    {
                        ipoint = i; //ipoint is the index of the node closest to the
                            point we want
                        ip = i+1; //ip is the second node closest to it (used in
                            interpolation)
                    }
                    else
                    {
                        ipoint = i+1;
                        ip = i;
                    }
                }
            }
            else
            {
                if(x[i]>point && x[i+1]<=point)
            {
                if(point-x[i+1]<x[i]-point)
                {
                    ipoint = i;
                    ip = i+1;
                    }
                    else
                    {
                        ipoint = i+1;
                        ip = i;
                    }
                }
            }
        }

```

```
}


// Linear  interpolation
double interpolation(float x, double T1, double T2, double x1, double x2)
{
        double result;
        result = T1+(T2-T1)*(x-x1)/(x2-x1);
        return result;
}


// Output of the  results
void output_files (int N, int M, float L, double* x, double* y, double* xvc, double* yvc, staggx u,
    staggy v)
{
        // Horizontal  coordinates
    ofstream xx;
        xx.open("x.dat");
    for(int i = 0; i<N+2; i++)
    {
        xx<<x[i]<<endl;
        }
    xx.close();

    // Vertical  coordinates
        ofstream yy;
    yy.open("y.dat");
    for(int j = M+1; j>=0; j--)
    {
        yy<<y[j]<<endl;
        }
    yy.close();

        // Horizontal  velocities
        ofstream resultats;
     resultats.open("Resultats.dat");
        for(int i = 0; i<N+1; i++)
        {
                for(int j = 0; j<M+2; j++)
                {
                        resultats<<xvc[i]<<"   "<<y[j]<<"      "<<u[j][i]<<endl;
                }
                resultats<<endl;
        }
        resultats.close();

        // Vertical  velocities
        ofstream resvltats;
     resvltats.open("Resvltats.dat");
```

```
	for(int i = 0; i<N+2; i++)
	{
		for(int j = 0; j<M+1; j++)
		{
			resvltats <<x[i]<<"	"<<yvc[j]<<"	"<<v[j][i]<<endl;
		}
		resvltats <<endl;
	}
	resvlt . close ();

	// Matrix of horizontal velocities
	ofstream result ;
    result .open("Matrixu.dat");
	for(int j = M+1; j>=0; j--)
	{
		for(int i = 0; i<N+2; i++)
		{
			if(i==0 || i==N+1)
			{
				result <<u[j][i]<<"	";
			}
			else
			{
				result <<convective_term (x[i], xvc[i-1], xvc[i], u[j][i-1], u[j][i])
				<<" ";
			}
		}
		result <<endl;
	}
	result . close ();

	// Matrix of vertical velocities
	ofstream resvlt ;
    resvlt .open("Matrixv.dat");
	for(int j = M+1; j>=0; j--)
	{
		for(int i = 0; i<N+2; i++)
		{
			if(j==0 && j==M+1)
			{
				resvlt <<v[j][i]<<"	";
			}
			else
			{
				resvlt <<convective_term (y[j], yvc[j-1], yvc[j], v[j-1][i], v[j][i])
				<<" ";
			}
		}
		resvlt <<endl;
	}
```

```
        resvlt . close();


        // Searching the indexes to interpolate
        int ipoint , ip , jpoint , jp;
        search_index (L/2, xvc, N+1, ipoint, ip);
    search_index (L/2, yvc, M+1, jpoint, jp);

        // Horizontal velocity in the central vertical line
        ofstream resultsu ;
    resultsu .open("u.dat");
    for(int i = M+1; i>=0; i--)
    {
        resultsu <<y[i]<<"        "<<interpolation(L/2, u[i][ipoint], u[i][ip], xvc[ipoint], xvc[ip])<<
            endl;
        }
    resultsu . close();

        // Vertical velocity in the central horizontal line
        ofstream resultsv ;
    resultsv .open("v.dat");
    for(int i = N+1; i>=0; i--)
    {
        resultsv <<x[i]<<"        "<<interpolation(L/2, v[jpoint][i], u[jp][i], yvc[jpoint], yvc[jp])<<
            endl;
        }
    resultsv . close();
}
```

4 | Differentially heated cavity

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

using namespace std;

// Numerical parameters
const int N = 50;
const int M = 50;

typedef double matrix[M+2][N+2];
typedef double staggx[M+2][N+1];
typedef double staggy[M+1][N+2];

void coordinates(float L, int N, double xvc[], double x[]) ;
void surface(double *yvc, int M, double Sv[]);
void volume(double *xvc, double *yvc, int N, int M, matrix& V);
void initial_conditions (int N, int M, staggx& u0, staggx& Ru0, staggy& v0, staggy& Rv0, matrix& T0);
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);
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);
double convective_term (double xf, double x2, double x3, double u2, double u3);
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);
void bp_coefficient (int N, int M, double dt, double* Sh, double* Sv, staggx up, staggy vp, matrix bp)
        ;
void Gauss_Seidel (matrix ap, matrix aw, matrix ae, matrix as, matrix an, matrix bp, float fr, float
        delta, int N, int M, matrix& T);
void velocities (int N, int M, double dt, double* x, double* y, matrix p, staggx up, staggy vp, staggx
         &u, staggy &v);
double min(double a, double b);
double max(double a, double b);
double time_step (double dtd, double* x, double* y, staggx u, staggy v);
double error (int N, int M, staggx u, staggy v, staggx u0, staggy v0, matrix T, matrix T0);
void heat_flux(int N, int M, double* x, staggx u, matrix T, matrix Q);
void Nusselt(int N, int M, double* x, double* yvc, matrix Q, double Nu[]);
void maximum_planes (int N, int M, double* x, double* y, staggx u, staggy v);
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);


int main()
{
        float Pr = 0.71; // Prandtl number
        int Ra = 1e6; // Rayleigh number
        float L = 1; // Length of the cavity

        float delta = 1e-4; // Precision of the simulation
        float fr = 1.2; // Relaxation factor

        cout<<"Program started"<<endl;
        cout<<"Pr="<<Pr<<endl<<endl;
        cout<<"Ra="<<Ra<<endl<<endl;

        // Coordinates
        double xvc[N+1], yvc[M+1], x[N+2], y[M+2];
        coordinates(L, N, xvc, x);
        coordinates(L, M, yvc, y);

        // Surfaces
        double Sh[N+2], Sv[M+2];
        matrix V;
        surface(xvc, N+2, Sh); // Horizontal surface
        surface(yvc, M+2, Sv); // Vertical surface
        volume(xvc, yvc, N+2, M+2, V); // Volume


        // Properties that are going to be calculated
        matrix p, T, T0, Q; // Values in the nodes (pressure)
        double Nu[N+2]; // Nusselt number
        staggx u, u0, Ru0; // Values in the points given by the staggered meshes ( velocities )
        staggy v, v0, Rv0;

        //  Inicialization
         initial_conditions (N, M, u0, Ru0, v0, Rv0, T0);

        matrix aTe, aTw, aTn, aTs, aTp, bTp;

        // Calculation of the constant coefficients that are used to determine the pressure
        matrix ae, aw, an, as, ap, bp;
         constant_coefficients (N+2, M+2, x, y, Sv, Sh, ae, aw, an, as, ap);

        // Time step (CFL condition)
        double resta = 1;
        double dtd = 0.2*pow(x[2]-xvc[1],2)/Pr;
        double dtc = 0.35*fabs(x[2]-xvc[1]);
        double dt = min(dtd, dtc);

        staggx up, Ru; // Intermediate velocities
        staggy vp, Rv;

        cout<<"Solving..."<<endl;
        // Fractional Step Method
        while(resta>delta)
        {
            // STEP 1: INTERMEDIATE VELOCITY
            intermediate_velocities (N, M, Pr, Ra, dt, x, y, xvc, yvc, Sh, Sv, V, T0, u0, v0, Ru0,
                Rv0, Ru, Rv, up, vp);


            // STEP 2: PRESSURE
            bp_coefficient (N+2, M+2, dt, Sh, Sv, up, vp, bp);
            Gauss_Seidel (ap, aw, ae, as, an, bp, fr, delta, N+2, M+2, p);


            // STEP 3: VELOCITY
            velocities (N, M, dt, x, y, p, up, vp, u, v);


            // STEP 4: TEMPERATURE
            temperature_coefficients (N, M, dt, x, y, Sv, Sh, V, u, v, T0, aTe, aTw, aTn, aTs, aTp,
                bTp);
            Gauss_Seidel (aTp, aTw, aTe, aTs, aTn, bTp, fr, delta, N+2, M+2, T);


            // STEP 5: TIME STEP
            dt = time_step (dtd, x, y, u, v);


            // Comprovation
            resta = error (N, M, u, v, u0, v0, T, T0);

            // New time step
            for(int i = 0; i<N+1; i++)
            {
                for(int j = 0; j<M+2; j++)
                {
                    u0[j][i] = u[j][i];
                    Ru0[j][i] = Ru[j][i];
                }
            }
            for(int i = 0; i<N+2; i++)
            {
                for(int j = 0; j<M+1; j++)
                {
```

```
                                        v0[j][i] = v[j][i];
                                        Rv0[j][i] = Rv[j][i];
                                }
                        }
                        for(int j = 0; j<M+2; j++)
                        {
                                for(int i = 0; i<N+2; i++)
                                {
                                        T0[j][i] = T[j][i];
                                }
                        }
                }

                // Results
                heat_flux(N, M, x, u, T, Q);
                Nusselt(N, M, x, y, Q, Nu);
            cout<<endl<<"Creating some output files..."<<endl;
            output_files (N, M, L, x, y, xvc, yvc, u, v, T, Nu);
                maximum_planes (N, M, x, y, u, v);

                return 0;
}


// Coordinates of the control volumes (x -> nodes, xvc -> faces)
void coordinates(float L, int N, double xvc[], double x[])
{
        double dx = L/N;
        xvc[0] = 0;
        x[0] = 0;
        for(int i = 0; i<N; i++)
        {
                xvc[i+1] = xvc[i]+dx;
                x[i+1] = (xvc[i+1]+xvc[i])/2;
        }
        x[N+1] = L;
}


// Surfaces of the control volumes
void surface(double *yvc, int M, double Sv[])
{
        for(int j = 0; j<M-1; j++)
        {
                Sv[j+1] = fabs(yvc[j]-yvc[j+1]);
        }
        Sv[0] = 0;
        Sv[M-1] = 0;
}

```

```

// Volume of each control volume
void volume(double *xvc, double *yvc, int N, int M, matrix& V)
{
	for(int i = 0; i<N; i++)
	{
		for(int j = 0; j<M; j++)
		{
			if(i==N-1 || j==M-1)
			{
				V[j][i] = 0;
			}
			else
			{
				V[j][i] = fabs(xvc[i]-xvc[i-1])*fabs(yvc[j]-yvc[j-1]);
			}
		}
	}
}


// Initial conditions
void initial_conditions (int N, int M, staggx& u0, staggx& Ru0, staggy& v0, staggy& Rv0, matrix& T0)
{
	for(int j = 0; j<M+2; j++)
	{
		for(int i = 0; i<N+1; i++)
		{
			u0[j][i] = 0; // Horizontal velocity at n
			Ru0[j][i] = 0; // R (horizontal) at n-1
		}
	}
	for(int j = 0; j<M+1; j++)
	{
		for(int i = 0; i<N+2; i++)
		{
			v0[j][i] = 0; // Vertical velocity at n
			Rv0[j][i] = 0; // R (vertical) at n-1
		}
	}
	for(int j = 0; j<M+2; j++)
	{
		for(int i = 0; i<N+2; i++)
		{
			if(i==0)
			{
				T0[j][i] = 1;
			}
			else
			{
```

```
                                        T0[j][i] = 0;
                              }
                    }
          }
}


// Calculation of the constant coefficients (ae, aw, an, as, ap) of the Poisson equation (pressure)
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)
{
          for(int i = 0; i<N; i++)
          {
                    for(int j = 0; j<M; j++)
                    {
                              if(j==M-1 && i!=0 && i!=N-1)
                              {
                                        ae[j][i] = 0;
                                        aw[j][i] = 0;
                                        an[j][i] = 0;
                                        as[j][i] = 1;
                                        ap[j][i] = 1;
                              }
                              else if(i==0 && j==0)
                              {
                                        ae[j][i] = 1;
                                        aw[j][i] = 0;
                                        an[j][i] = 1;
                                        as[j][i] = 0;
                                        ap[j][i] = 1;
                              }
                              else if(i==0 && j==M-1)
                              {
                                        ae[j][i] = 1;
                                        aw[j][i] = 0;
                                        an[j][i] = 0;
                                        as[j][i] = 1;
                                        ap[j][i] = 1;
                              }
                              else if(i==0 && j!=0 && j!=M-1)
                              {
                                        ae[j][i] = 1;
                                        aw[j][i] = 0;
                                        an[j][i] = 0;
                                        as[j][i] = 0;
                                        ap[j][i] = 1;
                              }
                              else if(i==N-1 && j==0)
                              {
                                        ae[j][i] = 0;
```

```
                        aw[j][i] = 1;
                        an[j][i] = 1;
                        as[j][i] = 0;
                        ap[j][i] = 1;
                    }
                    else if(i==N-1 && j==M-1)
                    {
                        ae[j][i] = 0;
                        aw[j][i] = 1;
                        an[j][i] = 0;
                        as[j][i] = 1;
                        ap[j][i] = 1;
                    }
                    else if(i==N-1 && j!=0 && j!=M-1)
                    {
                        ae[j][i] = 0;
                        aw[j][i] = 1;
                        an[j][i] = 0;
                        as[j][i] = 0;
                        ap[j][i] = 1;
                    }
                    else if(j==0 && i!=0 && i!=N-1)
                    {
                        ae[j][i] = 0;
                        aw[j][i] = 0;
                        an[j][i] = 1;
                        as[j][i] = 0;
                        ap[j][i] = 1;
                    }
                    else
                    {
                        ae[j][i] = Sv[j]/fabs(x[i+1]-x[i]);
                        aw[j][i] = Sv[j]/fabs(x[i]-x[i-1]);
                        an[j][i] = Sh[i]/fabs(y[j+1]-y[j]);
                        as[j][i] = Sh[i]/fabs(y[j]-y[j-1]);
                        ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i];
                    }
            }
        }
}


// Coefficients used to calculate the temperature
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)
{
        double Fe, Fw, Fn, Fs;
        double De, Dw, Dn, Ds;

        for(int j = 0; j<M+2; j++)
        {
            for(int i = 0; i<N+2; i++)
            {
                    // Mass flow terms (v*S)
                    Fe = u[j][i]*Sv[j];
                    Fw = u[j][i-1]*Sv[j];
                    Fn = v[j][i]*Sh[i];
                    Fs = v[j-1][i]*Sh[i];

                    // Areas and distances
                    De = Sv[j]/fabs(x[i+1]-x[i]);
                    Dw = Sv[j]/fabs(x[i]-x[i-1]);
                    Dn = Sh[i]/fabs(y[j+1]-y[j]);
                    Ds = Sh[i]/fabs(y[j+1]-y[j]);

                    if(i==0)
                    {
                        aTe[j][i] = 0;
                        aTw[j][i] = 0;
                        aTn[j][i] = 0;
                        aTs[j][i] = 0;
                        aTp[j][i] = 1;
                        bTp[j][i] = 1;
                    }
                    else if(i==N+1)
                    {
                        aTe[j][i] = 0;
                        aTw[j][i] = 0;
                        aTn[j][i] = 0;
                        aTs[j][i] = 0;
                        aTp[j][i] = 1;
                        bTp[j][i] = 0;
                    }
                    else if(j==0 && i!=0 && i!=N+1)
                    {
                        aTe[j][i] = 0;
                        aTw[j][i] = 0;
                        aTn[j][i] = 1;
                        aTs[j][i] = 0;
                        aTp[j][i] = 1;
                        bTp[j][i] = 0;
                    }
                    else if(j==M+1 && i!=0 && i!=N+1)
                    {
                        aTe[j][i] = 0;
                        aTw[j][i] = 0;
                        aTn[j][i] = 0;
                        aTs[j][i] = 1;
                        aTp[j][i] = 1;
                        bTp[j][i] = 0;
                    }
                    else
                    {
                        aTe[j][i] = De-0.5*Fe;
                        aTw[j][i] = Dw+0.5*Fw;
                        aTn[j][i] = Dn-0.5*Fn;
                        aTs[j][i] = Ds+0.5*Fs;
                        aTp[j][i] = aTe[j][i]+aTw[j][i]+aTn[j][i]+aTs[j][i]+V[j][i]/dt;
                        bTp[j][i] = T0[j][i]*V[j][i]/dt;
                    }
            }
    }
}


// Computation of the velocity in the convective term using CDS
double convective_term (double xf, double x2, double x3, double u2, double u3)
{
    // 2 refers to node P, 3 to node E
    double u;
    u = u2+fabs(x2-xf)*(u3-u2)/fabs(x3-x2);

    return u;
}


// Calculation of the intermediate velocities
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)
{
    double mflowe, mfloww, mflown, mflows;
    double ue, uw, un, us;

    for(int i = 0; i<N+1; i++)
    {
        for(int j = 0; j<M+2; j++)
        {
            // Mass flow terms (v*S)
            mflowe = (u0[j][i+1]+u0[j][i])*Sv[j]/2;
            mfloww = (u0[j][i-1]+u0[j][i])*Sv[j]/2;
            mflown = (v0[j][i]+v0[j][i+1])*Sh[i]/2;
            mflows = (v0[j-1][i]+v0[j-1][i+1])*Sh[i]/2;


            // HORIZONTAL
            ue = convective_term (x[i+1], xvc[i], xvc[i+1], u0[j][i], u0[j][i+1]);
            uw = convective_term (x[i], xvc[i], xvc[i-1], u0[j][i], u0[j][i-1]);
            un = convective_term (yvc[j], y[j], y[j+1], u0[j][i], u0[j+1][i]);
```

```
                    us = convective_term (yvc[j-1], y[j], y[j-1], u0[j][i], u0[j-1][i]);


                    // R (horizontal)
                    if(i==0 || i==N || j==0 || j==M+1)
                    {
                            Ru[j][i] = 0;
                    }
                    else
                    {
                            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];
                    }

                    // Intermediate velocity (horizontal)
                    up[j][i] = u0[j][i]+dt*(1.5*Ru[j][i]-0.5*Ru0[j][i]);
            }
    }

    double ve, vw, vn, vs;

    for(int i = 0; i<N+2; i++)
    {
            for(int j = 0; j<M+1; j++)
            {
                    // Mass flow terms (v*S)
                    mflowe = (u0[j+1][i]+u0[j][i])*Sv[j]/2;
                    mfloww = (u0[j+1][i-1]+u0[j][i-1])*Sv[j]/2;
                    mflown = (v0[j][i]+v0[j+1][i])*Sh[i]/2;
                    mflows = (v0[j][i]+v0[j-1][i])*Sh[i]/2;


                    // VERTICAL
                    ve = convective_term (xvc[i], x[i], x[i+1], v0[j][i], v0[j][i+1]);
                    vw = convective_term (xvc[i-1], x[i], x[i-1], v0[j][i], v0[j][i-1]);
                    vn = convective_term (y[j+1], yvc[j], yvc[j+1], v0[j][i], v0[j+1][i]);
                    vs = convective_term (y[j], yvc[j], yvc[j-1], v0[j][i], v0[j-1][i]);

                    // R (vertical)
                    if(i==0 || i==N+1 || j==0 || j==M)
                    {
                            Rv[j][i] = 0;
                    }
                    else
                    {
                            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(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;
                        }

                        // Intermediate velocity (vertical)
                        vp[j][i] = v0[j][i]+dt*(1.5*Rv[j][i]-0.5*Rv0[j][i]);
                }
        }
}


// Calculation of the bp coefficient of the Poisson equation (pressure)
void bp_coefficient (int N, int M, double dt, double* Sh, double* Sv, staggx up, staggy vp, matrix bp)
{
        for(int i = 0; i<N; i++)
        {
                for(int j = 0; j<M; j++)
                {
                        if(i==0 || j==0 || i==N-1 || j==M-1)
                        {
                                bp[j][i] = 0;
                        }
                        else
                        {
                                bp[j][i] = -(up[j][i]*Sv[j]+vp[j][i]*Sh[i]-up[j][i-1]*Sv[j]-vp[j-1][i
                                    ]*Sh[i])/dt;
                        }
                }
        }
}


// Solver (using Gauss-Seidel)
void Gauss_Seidel (matrix ap, matrix aw, matrix ae, matrix as, matrix an, matrix bp, float fr, float
    delta, int N, int M, matrix& T)
{
        double Tcalc[M][N]; // Temperature calculated in the previous iteration
        for(int i = 0; i<N; i++)
        {
                for(int j = 0; j<M; j++)
                {
                        Tcalc[j][i] = T[j][i];
                }
        }

        double MAX = 1; // Maximum value of the difference between T and Tcalc
        double resta = 1; // Difference between T and Tcalc

        while(MAX>delta)
        {

            // SOLVER: Gauss-Seidel
            for(int i = 0; i<N; i++)
            {
                for(int j = 0; j<M; j++)
                {
                    if(i==0 && j==M-1)
                    {
                        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]);
                    }
                    else if(i==0 && j==0)
                    {
                        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]);
                    }
                    else if(i==0 && j!=0 && j!=M-1)
                    {
                        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]);
                    }
                    else if(i==N-1 && j==M-1)
                    {
                        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]);
                    }
                    else if(i==N-1 && j==0)
                    {
                        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]);
                    }
                    else if(i==N-1 && j!=0 && j!=M-1)
                    {
                        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])
                            ;
                    }
                    else if(i!=0 && i!=N-1 && j==M-1)
                    {
                        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])
                            ;
                    }
                    else if(i!=0 && i!=N-1 && j==0)
                    {
                        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]);
                    }
                    else
                    {
                        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]);
                    }
                }
            }

            // Comprovation
            MAX = 0;
            for(int i = 0; i<N; i++)
            {
                for(int j = 0; j<M; j++)
                {
                    resta = fabs(Tcalc[j][i]-T[j][i]);

                    if(resta>MAX)
                    {
                        MAX = resta;
                    }
                }
            }

            // New assignation
            for(int i = 0; i<N; i++)
            {
                for(int j = 0; j<M; j++)
                {
                    Tcalc[j][i] = T[j][i];
                }
            }
        }
}


// Calculation of the velocity with the pressure correction
void velocities (int N, int M, double dt, double* x, double* y, matrix p, staggx up, staggy vp, staggx
    &u, staggy &v)
{
        // Horizontal velocity at n+1
        for(int i = 0; i<N+1; i++)
        {
            for(int j = 0; j<M+2; j++)
            {
                if(i==0 || i==N || j==0 || j==M+1)
                {
                    u[j][i] = 0;
                }
                else
                {
                        u[j][i] = up[j][i]-dt*(p[j][i+1]-p[j][i])/(fabs(x[i+1]-x[i]));
                }
            }
        }

        // Vertical velocity at n+1
        for(int i = 0; i<N+2; i++)
        {
            for(int j = 0; j<M+1; j++)
            {
                if(j==0 || j==M || i==0 || i==N+1)
                {
                        v[j][i] = 0;
                }
                else
                {
                        v[j][i] = vp[j][i]-dt*(p[j+1][i]-p[j][i])/(fabs(y[j+1]-y[j]));
                }
            }
        }
}


// Returns the minimum value
double min(double a, double b)
{
        if(a>b)
        {
                return b;
        }
        else
        {
                return a;
        }
}


// Returns the maximum value
double max(double a, double b)
{
        if(a>b)
        {
                return a;
        }
        else
        {
                return b;
        }
```

```
}


// Calculation of the proper time step (CFL condition)
double time_step (double dtd, double* x, double* y, staggx u, staggy v)
{
        double dt;
        double dtc = 100;

        for(int i = 1; i<N; i++)
        {
                for(int j = 1; j<M+1; j++)
                {
                        dtc = min(dtc, 0.35*fabs(x[i+1]-x[i])/fabs(u[j][i]));
                }
        }
        for(int i = 1; i<N+1; i++)
        {
                for(int j = 1; j<M; j++)
                {
                        dtc = min(dtc, 0.35*fabs(y[j+1]-y[j])/fabs(v[j][i]));
                }
        }
        dt = min(dtc, dtd);
        return dt;
}


// Difference between the previous and the actual time step
double error (int N, int M, staggx u, staggy v, staggx u0, staggy v0, matrix T, matrix T0)
{
        double resta = 0;
        for(int i = 0; i<N+1; i++)
        {
                for(int j = 0; j<M+2; j++)
                {
                        resta = max(resta, fabs(u[j][i]-u0[j][i]));
                }
        }
        for(int i = 0; i<N+2; i++)
        {
                for(int j = 0; j<M+1; j++)
                {
                        resta = max(resta, fabs(v[j][i]-v0[j][i]));
                }
        }
        for(int j = 0; j<M+2; j++)
        {
                for (int i = 0; i<N+2; i++)
                {
                        resta  = max(resta, fabs(T[j][i]-T0[j][i]));
                }
        }

        cout<<resta<<endl;
        return resta;
}


// Heat flux in the horizontal direction at any point in the cavity
void heat_flux(int N, int M, double* x, staggx u, matrix T, matrix Q)
{
        for(int j = 0; j<M+2; j++)
        {
                for(int i = 0; i<N+1; i++)
                {
                        if(i==0)
                        {
                                Q[j][i] = u[j][i]*T[j][i]-(T[j][i+1]-T[j][i])/fabs(x[i+1]-x[i]);
                        }
                        else if(i==N)
                        {
                                Q[j][i] = Q[j][0];
                        }
                        else
                        {
                                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
                                    +1]-x[i]);
                        }
                }
        }
}


// Computation of the Nusselt numbers
void Nusselt(int N, int M, double* x, double* yvc, matrix Q, double Nu[])
{
        for(int i = 0; i<N+1; i++)
        {
                Nu[i] = 0;
                for(int j = 0; j<M+1; j++)
                {
                        Nu[i] = Nu[i]+(yvc[j+1]-yvc[j])*Q[j][i];
                }

        }

        double Numax = -100;
        double Numin = 100;
        double Nuavg = 0;
        double Nu0 = Nu[0];
        double Nu12 = (Nu[N/2+1]+Nu[N/2+2])/2;
        int jmax, jmin;

        for(int i = 0; i<N+2; i++)
        {
                Nuavg = Nuavg+(x[i+1]-x[i])*Nu[i];
        }
        for(int j = 0; j<M+2; j++)
        {
                if(Q[j][0]>Numax)
                {
                        Numax = Q[j][0];
                        jmax = j;
                }
                if(Q[j][0]<Numin)
                {
                        Numin = Q[j][0];
                        jmin = j;
                }
        }
        cout<<endl<<endl;
        cout<<"Nu average = "<<Nuavg<<endl;
        cout<<"Nu0 = "<<Nu0<<endl;
        cout<<"Nu1/2 = "<<Nu12<<endl;
        cout<<"Nu max = "<<Numax<<" at y = "<<yvc[jmax]<<endl;
        cout<<"Nu min = "<<Numin<<" at y = "<<yvc[jmin]<<endl;
}


// Maximum velocity at the central horizontal and vertical planes
void maximum_planes (int N, int M, double* x, double* y, staggx u, staggy v)
{
        double umax = 0, vmax = 0;
        int imax, jmax;
        double uavg, vavg;
        for(int j = 0; j<M+2; j++)
        {
                uavg = (u[j][N/2+1]+u[j][N/2])/2;
                if(uavg>umax)
                {
                        umax = uavg;
                        jmax = j;
                }
        }
        for(int i = 0; i<N+2; i++)
        {
                vavg = (v[M/2+1][i]+v[M/2][i])/2;
                if(vavg>vmax)
                {
                        vmax = vavg;
                        imax = i;
                }
        }

        cout<<"u max = "<<umax<<" at y = "<<y[jmax]<<endl;
        cout<<"v max = "<<vmax<<" at x = "<<x[imax]<<endl;
}


// Output of the  results
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)
{
        // Horizontal coordinates
    ofstream xx;
        xx.open("x.dat");
    for(int i = 0; i<N+2; i++)
    {
        xx<<x[i]<<endl;
        }
    xx.close();

    // Vertical coordinates
        ofstream yy;
    yy.open("y.dat");
    for(int j = M+1; j>=0; j--)
    {
        yy<<y[j]<<endl;
        }
    yy.close();

        // Horizontal  velocities
        ofstream  resultats ;
     resultats .open("Resultats.dat");
        for(int i = 0; i<N+1; i++)
        {
                for(int j = 0; j<M+2; j++)
                {
                        resultats <<xvc[i]<<"   "<<y[j]<<"     "<<u[j][i]<<endl;
                }
                 resultats <<endl;
        }
         resultats . close();

        // Vertical  velocities
        ofstream  resvltats ;
     resvltats .open("Resvltats.dat");
        for(int i = 0; i<N+2; i++)
        {
```

```
                        for(int  j  = 0;  j<M+1; j++)
                        {
                                  resvltats <<x[i]<<"       "<<yvc[j]<<"    "<<v[j][i]<<endl;
                        }
                        resvltats <<endl;
              }
              resvltats . close();

    // Matrix of  horizontal   velocities
             ofstream   result ;
     result .open("Matrixu.dat");
             for(int  j  = M+1; j>=0; j−−)
             {
                        for(int  i  = 0;  i<N+2; i++)
                        {
                                  if(i==0 || i==N+1)
                                  {
                                            result <<u[j][i]<<"        ";
                                  }
                                  else
                                  {
                                            result <<convective_term (x[i], xvc[i−1], xvc[i],  u[j][i−1], u[j][i])
                                                 <<" ";
                                  }
                        }
                        result <<endl;
             }
             result . close();

             // Matrix of  vertical   velocities
             ofstream   resvlt ;
     resvlt .open("Matrixv.dat");
             for(int  j  = M+1; j>=0; j−−)
             {
                        for(int  i  = 0;  i<N+2; i++)
                        {
                                  if(j==0 && j==M+1)
                                  {
                                            resvlt <<v[j][i]<<"        ";
                                  }
                                  else
                                  {
                                            resvlt <<convective_term (y[j], yvc[j−1], yvc[j],  v[j−1][i],  v[j][i])
                                                 <<" ";
                                  }
                        }
                        resvlt <<endl;
             }
             resvlt . close();


    // Temperature
    ofstream temperature;
    temperature.open("Temperatura.dat");
    for(int j = M+1; j>=0; j--)
    {
        for(int i = 0; i<N+2; i++)
        {
                temperature<<T[j][i]<<" ";
                }
                temperature<<endl;
        }
        temperature.close();

        // Nusselt number
        ofstream nuss;
        nuss.open("Nusselt.dat");
        for(int i = 0; i<N+1; i++)
        {
                nuss<<xvc[i]<<" "<<Nu[i]<<endl;
        }
        nuss.close();
}
```

5 | Burgers' equation

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

using namespace std;


complex<double> diffusive(int k, int N, double Re, vector<complex<double> > u, bool LES, float CK);
complex<double> convective(int k, int N, vector<complex<double> > u);



int main()
{
        const int N = 20;
        const double Re = 40; // Reynolds number
        bool LES = 1; // 1 is LES, 0 is DNS
        double F = 0; // Source term (in Fourier space)

        double delta = 1e-6; // Precision of the simulation
        float CK = 0.05; // Kolgomorov constant
        float C1 = 0.02;
        double dt = C1*Re/pow(N,2); // Increment of time

        vector<complex<double> > u(N);
        vector<complex<double> > u0(N);

        for(double k = 0; k<N; k++)
        {
                u0[k] = 1/(k+1); // u at n
                u[k] = u0[k]; // u at n+1
        }

        complex<double> resta;
        double MAX = 1;


        double t = 0;

        while(MAX>delta)
        {
                t = t+dt;

                for(int k = 1; k<N; k++)
                {
                        u[k] = u0[k]+(diffusive(k, N, Re, u0, LES, CK)-convective(k, N, u0)+F)*dt;
                }

                // Comprovation
                MAX = 0;
                for(int k = 1; k<N; k++)
                {
                        resta = (u[k]-u0[k])/dt;
                        if(abs(resta)>MAX)
                        {
                                MAX = abs(resta);
                        }
                }

                for(int k = 1; k<N; k++)
                {
                        u0[k] = u[k];
                }
        }
        cout<<"Steady state reached at t="<<t;

        vector<double> E(N);
        for(int k = 0; k<N; k++)
        {
                E[k] = abs(u[k]*conj(u[k]));
        }

        ofstream results;
    results.open("Results.dat");
    for(int k = 0; k<N; k++)
    {
        results<<k+1<<" "<<E[k]<<endl;
        }
    results.close();

        return 0;
}



// Calculation of the diffusive term
```

```
complex<double> diffusive(int k, int N, double Re, vector<complex<double> > u, bool LES, float CK)
{
        if(!LES)
        {
                return -(double(k)+1)*(double(k)+1)*u[k]/Re;
        }
        else
        {
                int m = 2; // Slope of the energy spectrum

                double viscosity ;
                double eddy; // Eddy-viscosity
                double vinf;
                double vnon;
                double EkN = abs(u[N-1]*conj(u[N-1])); //Energy at the cutoff frequency

                vinf = 0.31*(5-m)*sqrt(3-m)*pow(CK,-3/2)/(m+1);
                vnon = 1+34.5*exp(-3.03*N/k);
                eddy = vinf*sqrt(EkN/N)*vnon;
                viscosity = 1/Re+eddy;
                return -(double(k)+1)*(double(k)+1)*u[k]*viscosity;
        }
}


// Calculation of the convective term
complex<double> convective(int k, int N, vector<complex<double> > u)
{
        complex<double> conv (0,0);
        complex<double> i(0,1);

        for(int p = -N; p<=N; p++)
        {
                int q = k+1-p;
                if(q>=-N && q<=N)
                {
                        int qu = q;
                        int pu = p;

                        if(qu==0 || pu==0){}
                        else if(qu<0)
                        {
                                qu = -q;
                                conv = conv+u[pu-1]*i*double(q)*conj(u[qu-1]);
                        }
                        else if(pu<0)
                        {
                                pu = -p;
                                conv = conv+conj(u[pu-1])*i*double(q)*u[qu-1];
                        }
```

```
            else
            {
                conv = conv+u[pu-1]*i*double(q)*u[qu-1];
            }
        }
    }
    return conv;
}
```

6 Square cylinder

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

using namespace std;

// Numerical parameters
const int N1 = 90;
const int N2 = 10;
const int N3 = 300;
const int N = N1+N2+N3;

const int M1 = 30;
const int M2 = 10;
const int M3 = 30;
const int M = M1+M2+M3;

typedef double matrix[M+2][N+2];
typedef double staggx[M+2][N+1];
typedef double staggy[M+1][N+2];
typedef double mtx[M+2][2];
typedef double mty[M+1][2];

void coordinates(float D, float l, float L, int N1, int N2, int N3, double xvc[], double x[]);
void surface(double *yvc, int M, double Sv[]);
void volume(double *xvc, double *yvc, int N, int M, matrix& V);
double parabolic(float umax, float H, double y);
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);
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);
double convective_term (double xf, double x2, double x3, double u2, double u3);
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);
void bp_coefficient (int N, int M, float rho, double dt, double* Sh, double* Sv, staggx up, staggy vp,
    matrix bp);
void Gauss_Seidel (matrix ap, matrix aw, matrix ae, matrix as, matrix an, matrix bp, float fr, float
        delta, int N, int M, matrix& T);
void 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);
double min(double a, double b);
double max(double a, double b);
double time_step (double dtd, double* x, double* y, staggx u, staggy v);
bool error (int N, int M, float delta, staggx u, staggy v, staggx u0, staggy v0);
void search_index (float point, double *x, int Number, int& ipoint, int& ip);
void output_files (int N, int M, double* x, double* y, double* xvc, double* yvc, staggx u, staggy v);

double xvc[N+1], yvc[M+1], x[N+2], y[M+2];
double Sh[N+2], Sv[M+2];
matrix V;
matrix p; // Values in the nodes (pressure)
staggx u, u0, Ru0; // Values in the points given by the staggered meshes ( velocities )
staggy v, v0, Rv0;
mtx u00;
mty v00;
staggx up, Ru; // Intermediate velocities
staggy vp, Rv;


int main()
{
        int Re = 3; // Reynolds number
        float D = 1; // Diameter of the cylinder
        float L = 50*D; // Length of the channel
        float H = 8*D; // Height of the channel
        float l = L/4; // inflow length
        float rho = 1; // Density
        float umax = 1; // Maximum velocity of the inflow parabolic velocity profile
        float mu = rho*umax*D/Re; // Viscosity

        float delta = 1e-4; // Precision of the simulation
        float fr = 1.2; // Relaxation factor

        cout<<"Program started"<<endl;
        cout<<"Re="<<Re<<endl<<endl;

        // Coordinates
        coordinates(D, l, L, N1, N2, N3, xvc, x);
        coordinates(D, H/2, H, M1, M2, M3, yvc, y);

        // Surfaces

        surface(xvc, N+2, Sh); // Horizontal surface
        surface(yvc, M+2, Sv); // Vertical surface
        volume(xvc, yvc, N+2, M+2, V); // Volume


        // Properties that are going to be calculated


        //  Inicialization
         initial_conditions (N, M, umax, H, y, u0, Ru0, v0, Rv0, u00, v00);

        // Calculation of the constant coefficients that are used to determine the pressure
        matrix ae, aw, an, as, ap, bp;
         constant_coefficients (N+2, M+2, x, y, xvc, yvc, Sv, Sh, ae, aw, an, as, ap);

        // Time step (CFL condition)
        double resta = 1;
        double dtd = 0.2*rho*pow(x[2]-xvc[1],2)/mu;
        double dtc = 0.35*fabs(x[2]-xvc[1])/umax;
        double dt = min(dtd, dtc);



        cout<<"Solving..."<<endl;
        // Fractional Step Method
        bool steady = false;
        while(!steady)
        {
                // STEP 1: INTERMEDIATE VELOCITY
                 intermediate_velocities (N, M, rho, mu, delta, dt, x, y, xvc, yvc, Sh, Sv, V, u0, v0,
                      Ru0, Rv0, Ru, Rv, up, vp);


                // STEP 2: PRESSURE
                 bp_coefficient (N, M, rho, dt, Sh, Sv, up, vp, bp);
                Gauss_Seidel (ap, aw, ae, as, an, bp, fr, delta, N+2, M+2, p);


                // STEP 3: VELOCITY
                 velocities (N, M, rho, dt, umax, H, x, y, p, up, vp, u0, v0, u00, v00, u, v);


                // STEP 4: TIME STEP
                dt = time_step (dtd, x, y, u, v);


                // Comprovation
                steady = error (N, M, delta, u, v, u0, v0);

                // New time step
                for(int i = 0; i<N+1; i++)
                {
                        for(int j = 0; j<M+2; j++)
                        {
                                u0[j][i] = u[j][i];
```

```
					Ru0[j][i] = Ru[j][i];
				}
			}
			for(int i = 0; i<N+2; i++)
			{
				for(int j = 0; j<M+1; j++)
				{
					v0[j][i] = v[j][i];
					Rv0[j][i] = Rv[j][i];
				}
			}
			for(int i = 0; i<2; i++)
			{
				for(int j = 0; j<M+2; j++)
				{
					u00[j][i] = u0[j][i+N-1];
				}
			}
			for(int j = 0; j<M+1; j++)
			{
				for(int i = 0; i<2; i++)
				{
					v00[j][i] = v0[j][i+N];
				}
			}
	}

	// Results
    cout<<endl<<"Creating some output files..."<<endl;
    output_files (N, M, x, y, xvc, yvc, u, v);


	return 0;
}


// Coordinates of the control volumes (x -> nodes, xvc -> faces)
void coordinates(float D, float l, float L, int N1, int N2, int N3, double xvc[], double x[])
{
	double dx;
	double dx1 = (l-D/2)/N1;
	double dx2 = D/N2;
	double dx3 = (L-l-D/2)/N3;
	xvc[0] = 0;
	x[0] = 0;
	for(int i = 0; i<N1+N2+N3; i++)
	{
		if(i<N1)
		{
			dx = dx1;
```

```
                }
                else if(i<N1+N2 && i>=N1)
                {
                        dx = dx2;
                }
                else
                {
                        dx = dx3;
                }
                xvc[i+1] = xvc[i]+dx;
                x[i+1] = (xvc[i]+xvc[i+1])/2;
        }
        x[N1+N2+N3+1] = L;
}


// Surfaces of the control volumes
void surface(double *yvc, int M, double Sv[])
{
        for(int j = 0; j<M-1; j++)
        {
                Sv[j+1] = fabs(yvc[j]-yvc[j+1]);
        }
        Sv[0] = 0;
        Sv[M-1] = 0;
}


// Volume of each control volume
void volume(double *xvc, double *yvc, int N, int M, matrix& V)
{
        for(int i = 0; i<N; i++)
        {
                for(int j = 0; j<M; j++)
                {
                        if(i==0 || i==N-1 || j==0 || j==M-1)
                        {
                                V[j][i] = 0;
                        }
                        else
                        {
                                V[j][i] = fabs(xvc[i]-xvc[i-1])*fabs(yvc[j]-yvc[j-1]);
                        }
                }
        }
}


// Parabolic velocity profile in the input
double parabolic(float umax, float H, double y)
```

```
{
        return 4*umax*(y/H-y*y/(H*H));
}


// Initial conditions of the problem
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)
{
        for(int j = 0; j<M+2; j++)
        {
                for(int i = 0; i<N+1; i++)
                {
                        if(i==0)
                        {
                                u0[j][i] = parabolic(umax, H, y[j]); // Horizontal velocity at n
                        }
                        else
                        {
                                u0[j][i] = 0; // Horizontal velocity at n
                        }
                        Ru0[j][i] = 0; // R (horizontal) at n-1
                }
        }
        for(int j = 0; j<M+1; j++)
        {
                for(int i = 0; i<N+2; i++)
                {
                        v0[j][i] = 0; // Vertical velocity at n
                        Rv0[j][i] = 0; // R (vertical) at n-1
                }
        }
        for(int i = 0; i<2; i++)
        {
                for(int j = 0; j<M+2; j++)
                {
                        u00[j][i] = 0; // Horizontal velocity at n-1
                }
        }
        for(int j = 0; j<M+1; j++)
        {
                for(int i = 0; i<2; i++)
                {
                        v00[j][i] = 0; // Vertical velocity at n-1
                }
        }
}


// Calculation of the constant coefficients (ae, aw, an, as, ap) of the Poisson equation (pressure)
```

```
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)
{
        for(int i = 0; i<N; i++)
        {
                for(int j = 0; j<M; j++)
                {
                        // Coefficients in the channel walls, input and output
                        if(j==M-1 && i!=0 && i!=N-1)
                        {
                                ae[j][i] = 0;
                                aw[j][i] = 0;
                                an[j][i] = 0;
                                as[j][i] = 1;
                                ap[j][i] = 1;
                        }
                        else if(i==0 && j==0)
                        {
                                ae[j][i] = 1;
                                aw[j][i] = 0;
                                an[j][i] = 1;
                                as[j][i] = 0;
                                ap[j][i] = 1;
                        }
                        else if(i==0 && j==M-1)
                        {
                                ae[j][i] = 1;
                                aw[j][i] = 0;
                                an[j][i] = 0;
                                as[j][i] = 1;
                                ap[j][i] = 1;
                        }
                        else if(i==0 && j!=0 && j!=M-1)
                        {
                                ae[j][i] = 1;
                                aw[j][i] = 0;
                                an[j][i] = 0;
                                as[j][i] = 0;
                                ap[j][i] = 1;
                        }
                        else if(i==N-1 && j==0)
                        {
                                ae[j][i] = 0;
                                aw[j][i] = 0;
                                an[j][i] = 1;
                                as[j][i] = 0;
                                ap[j][i] = 1;
                        }
                        else if(i==N-1 && j==M-1)
                        {
```

```
            ae[j][i] = 0;
            aw[j][i] = 0;
            an[j][i] = 0;
            as[j][i] = 1;
            ap[j][i] = 1;
        }
        else if(i==N-1 && j!=0 && j!=M-1)
        {
            ae[j][i] = 0;
            aw[j][i] = 0;
            an[j][i] = 0;
            as[j][i] = 0;
            ap[j][i] = 1;
        }
        else if(j==0 && i!=0 && i!=N-1)
        {
            ae[j][i] = 0;
            aw[j][i] = 0;
            an[j][i] = 1;
            as[j][i] = 0;
            ap[j][i] = 1;
        }
        // Coefficients in the cylinder
        else if(i>N1+1 && i<N1+N2 && j>M1+1 && j<M1+M2)
        {
            ae[j][i] = 0;
            aw[j][i] = 0;
            an[j][i] = 0;
            as[j][i] = 0;
            ap[j][i] = 1;
        }
        // Coefficients in the points near the cylinder
        else if(i==N1 && j>M1 && j<M1+M2+1)
        {
            ae[j][i] = Sv[j]/fabs(xvc[i]-x[i]);
            aw[j][i] = Sv[j]/fabs(x[i]-x[i-1]);
            an[j][i] = Sh[i]/fabs(y[j+1]-y[j]);
            as[j][i] = Sh[i]/fabs(y[j]-y[j-1]);
            ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i];
        }
        else if(i==N1+N2+1 && j>M1 && j<M1+M2+1)
        {
            ae[j][i] = Sv[j]/fabs(x[i+1]-x[i]);
            aw[j][i] = Sv[j]/fabs(x[i]-xvc[i-1]);
            an[j][i] = Sh[i]/fabs(y[j+1]-y[j]);
            as[j][i] = Sh[i]/fabs(y[j]-y[j-1]);
            ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i];
        }
        else if(i>N1 && i<N1+N2+1 && j==M1)
        {
```

```
                    ae[j][i] = Sv[j]/fabs(x[i+1]-x[i]);
                    aw[j][i] = Sv[j]/fabs(x[i]-x[i-1]);
                    an[j][i] = Sh[i]/fabs(yvc[j]-y[j]);
                    as[j][i] = Sh[i]/fabs(y[j]-y[j-1]);
                    ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i];
                }
                else if(i>N1 && i<N1+N2+1 && j==M1+M2+1)
                {
                    ae[j][i] = Sv[j]/fabs(x[i+1]-x[i]);
                    aw[j][i] = Sv[j]/fabs(x[i]-x[i-1]);
                    an[j][i] = Sh[i]/fabs(y[j+1]-y[j]);
                    as[j][i] = Sh[i]/fabs(y[j]-yvc[j-1]);
                    ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i];
                }
                // Coefficients in the channel
                else
                {
                    ae[j][i] = Sv[j]/fabs(x[i+1]-x[i]);
                    aw[j][i] = Sv[j]/fabs(x[i]-x[i-1]);
                    an[j][i] = Sh[i]/fabs(y[j+1]-y[j]);
                    as[j][i] = Sh[i]/fabs(y[j]-y[j-1]);
                    ap[j][i] = ae[j][i]+aw[j][i]+an[j][i]+as[j][i];
                }
            }
        }
}


// Computation of the velocity in the convective term using CDS
double convective_term (double xf, double x2, double x3, double u2, double u3)
{
        // 2 refers to node P, 3 to node E
        double u;
        u = u2+fabs(x2-xf)*(u3-u2)/fabs(x3-x2);

        return u;
}


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)
{
        double mflowe, mfloww, mflown, mflows;
        double ue, uw, un, us;

        for(int i = 0; i<N+1; i++)
        {
            for(int j = 0; j<M+2; j++)
            {
```

```
// Mass flow terms (rho*v*S)
mflowe = (rho*u0[j][i+1]+rho*u0[j][i])*Sv[j]/2;
mfloww = (rho*u0[j][i-1]+rho*u0[j][i])*Sv[j]/2;
mflown = (rho*v0[j][i]+rho*v0[j][i+1])*Sh[i]/2;
mflows = (rho*v0[j-1][i]+rho*v0[j-1][i+1])*Sh[i]/2;


// HORIZONTAL
ue = convective_term (x[i+1], xvc[i], xvc[i+1], u0[j][i], u0[j][i+1]);
uw = convective_term (x[i], xvc[i], xvc[i-1], u0[j][i], u0[j][i-1]);
un = convective_term (yvc[j], y[j], y[j+1], u0[j][i], u0[j+1][i]);
us = convective_term (yvc[j-1], y[j], y[j-1], u0[j][i], u0[j-1][i]);


// R (horizontal)
// Channel walls, input and output
if(i==0 || i==N || j==0 || j==M+1)
{
        Ru[j][i] = 0;
}
// In the cylinder
else if(i>=N1 && i<=N1+N2 && j>M1 && j<M1+M2+1)
{
        Ru[j][i] = 0;
}
// Points that surround the cylinder
else if(j==M1 && i>=N1 && i<=N1+N2)
{
        un = 0;
        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];
}
else if(j==M1+M2+1 && i>=N1 && i<=N1+N2)
{
        us = 0;
        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];
}
// Channel
else
{
        Ru[j][i] = (mu*(u0[j][i+1]-u0[j][i])*Sv[j]/fabs(xvc[i+1]-xvc[i])+mu*(
            u0[j+1][i]-u0[j][i])*Sh[i]/fabs(y[j+1]-y[j])-mu*(u0[j][i]-u0[j][i
            -1])*Sv[j]/fabs(xvc[i]-xvc[i-1])-mu*(u0[j][i]-u0[j-1][i])*Sh[i]/
```

```
                                        fabs(y[j]−y[j−1])−(mflowe*ue+mflown*un−mfloww*uw−mflows*us
                                        ))/V[j][i];
                                }

                                // Intermediate velocity (horizontal)
                                up[j][i] = u0[j][i]+dt*(1.5*Ru[j][i]−0.5*Ru0[j][i])/rho;
                        }
                }

                double ve, vw, vn, vs;

                for(int i = 0; i<N+2; i++)
                {
                        for(int j = 0; j<M+1; j++)
                        {
                                // Mass flow terms (rho*v*S)
                                mflowe = (rho*u0[j+1][i]+rho*u0[j][i])*Sv[j]/2;
                                mfloww = (rho*u0[j+1][i−1]+rho*u0[j][i−1])*Sv[j]/2;
                                mflown = (rho*v0[j][i]+rho*v0[j+1][i])*Sh[i]/2;
                                mflows = (rho*v0[j][i]+rho*v0[j−1][i])*Sh[i]/2;


                                // VERTICAL
                                ve = convective_term (xvc[i], x[i], x[i+1], v0[j][i], v0[j][i+1]);
                                vw = convective_term (xvc[i−1], x[i], x[i−1], v0[j][i], v0[j][i−1]);
                                vn = convective_term (y[j+1], yvc[j], yvc[j+1], v0[j][i], v0[j+1][i]);
                                vs = convective_term (y[j], yvc[j], yvc[j−1], v0[j][i], v0[j−1][i]);

                                // R (vertical)
                                // Channel walls, input and output
                                if(i==0 || i==N+1 || j==0 || j==M)
                                {
                                        Rv[j][i] = 0;
                                }
                                // In the cylinder
                                else if(i>N1 && i<N1+N2+1 && j>=M1 && j<=M1+M2)
                                {
                                        Rv[j][i] = 0;
                                }
                                // Points that surround the cylinder
                                else if(j>=M1 && j<=M1+M2 && i==N1)
                                {
                                        ve = 0;
                                        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]/fabs(x[i]−x[i−1])−mu*(v0[j][i]−v0[j−1][i])*Sh[i]/fabs(
                                            yvc[j]−yvc[j−1])−(mflowe*ve+mflown*vn−mfloww*vw−mflows*vs))
                                            /V[j][i];
                                }
                                else if(j>=M1 && j<=M1+M2 && i==N1+N2+1)
                        {
                                vw = 0;
                                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];
                        }
                        // Channel
                        else
                        {
                                Rv[j][i] = (mu*(v0[j][i+1]-v0[j][i])*Sv[j]/fabs(x[i+1]-x[i])+mu*(v0[j
                                      +1][i]-v0[j][i])*Sh[i]/fabs(yvc[j+1]-yvc[j])-mu*(v0[j][i]-v0[j][i
                                      -1])*Sv[j]/fabs(x[i]-x[i-1])-mu*(v0[j][i]-v0[j-1][i])*Sh[i]/fabs(
                                      yvc[j]-yvc[j-1])-(mflowe*ve+mflown*vn-mfloww*vw-mflows*vs))
                                      /V[j][i];
                        }

                        // Intermediate velocity (vertical)
                        vp[j][i] = v0[j][i]+dt*(1.5*Rv[j][i]-0.5*Rv0[j][i])/rho;
                }
        }
}


// Calculation of the bp coefficient of the Poisson equation (pressure)
void bp_coefficient (int N, int M, float rho, double dt, double* Sh, double* Sv, staggx up, staggy vp,
        matrix bp)
{
        for(int i = 0; i<N+2; i++)
        {
                for(int j = 0; j<M+2; j++)
                {
                        // Channel walls, input and output
                        if(i==0 || j==0 || i==N-1 || j==M-1)
                        {
                                bp[j][i] = 0;
                        }
                        // Cylinder
                        else if(i>N1 && i<N1+N2+1 && j>M1 && j<M1+M2+1)
                        {
                                bp[j][i] = 0;
                        }
                        // Channel
                        else
                        {
                                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;
                        }
                }
        }
}


// Solver (using Gauss-Seidel)
void Gauss_Seidel (matrix ap, matrix aw, matrix ae, matrix as, matrix an, matrix bp, float fr, float
    delta, int N, int M, matrix& T)
{
        double Tcalc[M][N]; // Temperature calculated in the previous iteration
        for(int i = 0; i<N; i++)
        {
                for(int j = 0; j<M; j++)
                {
                        Tcalc[j][i] = T[j][i];
                }
        }

        double MAX = 1; // Maximum value of the difference between T and Tcalc
        double resta = 1; // Difference between T and Tcalc

        while(MAX>delta)
        {

                // SOLVER: Gauss-Seidel
                for(int i = 0; i<N; i++)
                {
                        for(int j = 0; j<M; j++)
                        {
                                if(i==0 && j==M-1)
                                {
                                        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]);
                                }
                                else if(i==0 && j==0)
                                {
                                        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]);
                                }
                                else if(i==0 && j!=0 && j!=M-1)
                                {
                                        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]);
                                }
                                else if(i==N-1 && j==M-1)
                                {
                                        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]);
                                }
                                else if(i==N-1 && j==0)
                                {
                                        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]);
                                }
                                else if(i==N-1 && j!=0 && j!=M-1)
                                {
                                        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])
                                            ;
                                }
                                else if(i!=0 && i!=N-1 && j==M-1)
                                {
                                        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])
                                            ;
                                }
                                else if(i!=0 && i!=N-1 && j==0)
                                {
                                        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]);
                                }
                                else
                                {
                                        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]);
                                }
                        }
                }

                // Comprovation
                MAX = 0;
                for(int i = 0; i<N; i++)
                {
                        for(int j = 0; j<M; j++)
                        {
                                resta = fabs(Tcalc[j][i]-T[j][i]);

                                if(resta>MAX)
                                {
                                        MAX = resta;
                                }
                        }
                }

                // New assignation
                for(int i = 0; i<N; i++)
                {
                        for(int j = 0; j<M; j++)
```

```
                        {
                                Tcalc[j][i] = T[j][i];
                        }
                }
        }
}


// Calculation of the velocity with the pressure correction
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)
{
        // Horizontal velocity at n+1
        for(int i = 0; i<N+1; i++)
        {
                for(int j = 0; j<M+2; j++)
                {
                        // Channel walls
                        if(j==0 || j==M+1)
                        {
                                u[j][i] = 0;
                        }
                        // Channel input
                        else if(i==0)
                        {
                                u[j][i] = parabolic(umax, H, y[j]);
                        }
                        // Channel output
                        else if(i==N)
                        {
                                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]));
                        }
                        // Cylinder
                        else if(i>=N1 && i<=N1+N2 && j>M1 && j<M1+M2+1)
                        {
                                u[j][i] = 0;
                        }
                        // Channel
                        else
                        {
                                u[j][i] = up[j][i]-dt*(p[j][i+1]-p[j][i])/(rho*fabs(x[i+1]-x[i]));
                        }
                }
        }

        // Vertical velocity at n+1
        for(int i = 0; i<N+2; i++)
        {
                for(int j = 0; j<M+1; j++)
                {
                        // Channel walls and input
                        if(j==0 || j==M || i==0)
                        {
                                v[j][i] = 0;
                        }
                        // Channel output
                        else if(i==N+1)
                        {
                                v[j][i] = v0[j][i]-dt*umax*(1.5*(v0[j][i]-v0[j][i-1])-0.5*(v00[j][2]-v00[j][1]))/fabs(x[i]-x[i-1]);
                        }
                        // Cylinder
                        else if(i>N1 && i<N1+N2+1 && j>=M1 && j<=M1+M2)
                        {
                                v[j][i] = 0;
                        }
                        else
                        {
                                v[j][i] = vp[j][i]-dt*(p[j+1][i]-p[j][i])/(rho*fabs(y[j+1]-y[j]));
                        }
                }
        }
}


// Returns the minimum value
double min(double a, double b)
{
        if(a>b)
        {
                return b;
        }
        else
        {
                return a;
        }
}


// Returns the maximum value
double max(double a, double b)
{
        if(a>b)
        {
                return a;
        }
        else
        {
                return b;
        }
}


// Calculation of the proper time step (CFL condition)
double time_step (double dtd, double* x, double* y, staggx u, staggy v)
{
        double dt;
        double dtc = 100;

        for(int i = 1; i<N; i++)
        {
                for(int j = 1; j<M+1; j++)
                {
                        if(i>=N1 && i<=N1+N2 && j>M1 && j<M1+M2+1)
                        {}
                        else
                        {
                                dtc = min(dtc, 0.35*fabs(x[i+1]-x[i])/fabs(u[j][i]));
                        }
                }
        }
        for(int i = 1; i<N+1; i++)
        {
                for(int j = 1; j<M; j++)
                {
                        if(i>N1 && i<N1+N2+1 && j>=M1 && j<=M1+M2)
                        {}
                        else
                        {
                                dtc = min(dtc, 0.35*fabs(y[j+1]-y[j])/fabs(v[j][i]));
                        }
                }
        }
        dt = min(dtc, dtd);
        return dt;
}


// Difference between the previous and the actual time step
bool error (int N, int M, float delta, staggx u, staggy v, staggx u0, staggy v0)
{
        double resta = 0;
        for(int i = 0; i<N+1; i++)
        {
                for(int j = 0; j<M+2; j++)
                {
                        resta = max(resta, fabs(u[j][i]-u0[j][i]));
                }
        }
        for(int i = 0; i<N+2; i++)
        {
                for(int j = 0; j<M+1; j++)
                {
                        resta = max(resta, fabs(v[j][i]-v0[j][i]));
                }
        }
        cout<<resta<<endl;
        if(resta>delta)
        {
                return false;
        }
        else
        {
                return true;
        }
}


// Searching the index of the node closest to a given point (and the second closest)
void search_index (float point, double *x, int Number, int& ipoint, int& ip)
{
        for(int i = 0; i<Number-1; i++)
    {
        if(x[i+1]-x[i]>0)
        {
                if(x[i]<=point && x[i+1]>point)
                {
                        if(point-x[i]<x[i+1]-point)
                                {
                                        ipoint = i; //ipoint is the index of the node closest to the
                                             point we want
                                        ip = i+1; //ip is the second node closest to it (used in
                                             interpolation)
                                }
                                else
                                {
                                        ipoint = i+1;
                                        ip = i;
                                }
                        }
                }
                else
                {
                        if(x[i]>point && x[i+1]<=point)
                {
                        if(point-x[i+1]<x[i]-point)
                        {
                                ipoint = i;
                                ip = i+1;
```

```
                                        }
                                        else
                                        {
                                                ipoint  = i+1;
                                                ip  = i;
                                        }
                                }
                        }
                }

}


// Output of the  results
void  output_files  (int  N,  int  M, double* x, double* y,  double* xvc,  double* yvc,  staggx  u ,  staggy  v)
{
        // Horizontal  coordinates
    ofstream  xx;
        xx.open("x.dat");
    for(int  i  = 0;  i<N+2; i++)
    {
        xx<<x[i]<<endl;
        }
    xx. close();

    // Vertical  coordinates
        ofstream  yy;
    yy.open("y.dat");
    for(int  j  = M+1; j>=0; j--)
    {
        yy<<y[j]<<endl;
        }
    yy. close();

        // Horizontal   velocities
        ofstream   resultats ;
     resultats .open("Resultats.dat");
        for(int  j  = M+1; j>=0; j--)
        {
                for(int  i  = 0;  i<N+1; i++)
                {
                        if(i>N1 && i<N1+N2 && j>M1+1 && j<M1+M2)
                        {
                                resultats <<xvc[i]<<"   "<<y[j]<<"      "<<"nan"<<endl;
                        }
                        else  if(i>N1 && i<N1+N2 && j==M1+1)
                        {
                                resultats <<xvc[i]<<"   "<<yvc[j-1]<<" "<<0<<endl;
                        }
                        else  if(i>N1 && i<N1+N2 && j==M1+M2)
                        {
                                resultats <<xvc[i]<<"   "<<yvc[j]<<"   "<<0<<endl;
                        }
                        else
                        {
                                resultats <<xvc[i]<<"   "<<y[j]<<"      "<<u[j][i]<<endl;
                        }
                }
                resultats <<endl;
        }
        resultats . close();

        // Vertical   velocities
        ofstream  resvltats ;
    resvltats .open("Resvltats.dat");
        for(int  j  = M; j>=0; j--)
        {
                for(int  i  = 0;  i<N+2; i++)
                {
                        if(i>N1+1 && i<N1+N2 && j>M1 && j<M1+M2)
                        {
                                resvltats <<x[i]<<"      "<<yvc[j]<<"   "<<"nan"<<endl;
                        }
                        else  if(i==N1+1 && j>M1 && j<M1+M2)
                        {
                                resvltats <<xvc[i-1]<<" "<<yvc[j]<<" "<<0<<endl;
                        }
                        else  if(i==N1+N2 && j>M1 && j<M1+M2)
                        {
                                resvltats <<xvc[i]<<"   "<<yvc[j]<<"   "<<0<<endl;
                        }
                        else
                        {
                                resvltats <<x[i]<<"      "<<yvc[j]<<"   "<<v[j][i]<<endl;
                        }
                }
                resvltats <<endl;
        }
        resvltats . close();

        // Matrix  of  horizontal   velocities
        ofstream  result ;
    result .open("Matrixu.dat");
        for(int  j  = M+1; j>=0; j--)
        {
                for(int  i  = 0;  i<N+2; i++)
                {
                        if(i==0 || i==N+1)
                        {
                                result <<u[j][i]<<"       ";
                        }
                        else
                        {
                                result <<convective_term (x[i], xvc[i-1], xvc[i],  u[j][i-1], u[j][i])
                                    <<" ";
                        }
                }
                result <<endl;
        }
        result . close();

        // Matrix of  vertical   velocities
        ofstream  resvlt ;
    resvlt .open("Matrixv.dat");
        for(int  j  = M+1; j>=0; j--)
        {
                for(int  i  = 0; i<N+2; i++)
                {
                        if(j==0 && j==M+1)
                        {
                                resvlt <<v[j][i]<<"       ";
                        }
                        else
                        {
                                resvlt <<convective_term (y[j], yvc[j-1], yvc[j],  v[j-1][i],  v[j][i])
                                    <<" ";
                        }
                }
                resvlt <<endl;
        }
        resvlt . close();


        // Pressure  matrix
        ofstream  press;
        press.open("Pressure.dat");
        for(int  j  = M+1; j>=0; j--)
        {
                for(int  i  = 0; i<N+2; i++)
                {
                        press<<p[j][i]<<"        ";
                }
                press<<endl;
        }
        press.close();

        // Pressure
        ofstream  presultats ;
        presultats .open("Presultats.dat");
        for(int  j  = M+1; j>=0; j--)
        {
                for(int  i  = 0;  i<N+2; i++)
                {
                        if(i>N1 && i<=N1+N2 && j>M1 && j<=M1+M2)
                        {
                                presultats <<x[i]<<"     "<<y[j]<<"      "<<"nan"<<endl;
                        }
                        else
                        {
                                presultats <<x[i]<<"     "<<y[j]<<"      "<<p[j][i]<<endl;
                        }
                }
                presultats <<endl;
        }

}
```