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

UNIVERSITAT POLITÈCNICA DE CATALUNYA

BACHELOR'S DEGREE IN AEROSPACE TECHONOLOGIES ENGINEERING

BACHELOR'S THESIS - ANNEX I : POTENTIAL FLOW

Computational studies of non-viscous and viscous fluid flows

Author:
PÉREZ RICARDO, Carlos

Director:
OLIVA LLENA, Asensio

Co-director:
PÉREZ SEGARRA, Carles-David

June 10^{th}, 2019

Contents

List of Figures

1 Lineal Parallel flow in a duct

1.1 Results

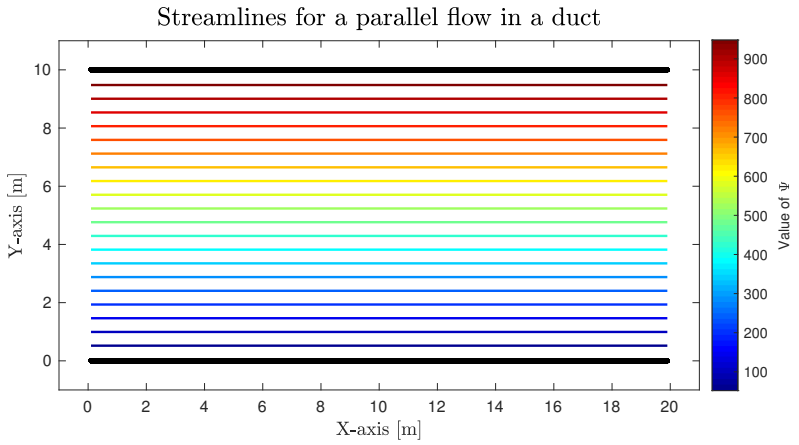

Figure 1.1: Linear Parallel flow in a duct - Streamlines

1.2 Code in C++: ParallelFlow.cpp

```
#include <iostream>
#include <stdlib.h>
#include <fstream>
#include <math.h>
#include <iomanip>
#include <cstdlib>

using namespace std;

// Constants

const float pi = 3.14159265;

// PROBLEM DATA

const int Nx = 100; // number of divisions along dir. X
const int Ny = 100; // number of divisions along dir. Y
const int dim = 2; // number of dimensions
float L = 20; // length of the canal in [m]
float H = 10; // height of the canal in [m]
const float u_inf = 100; // entrance's velocity  [m]


// VARIABLES

    const int nnode = Nx*Ny-1;
    float COOR[Nx+1][Ny+1][dim];
    float COORlim[Nx+2][4][dim];
    bool CosFluidlim[Nx*4];
    float X2[nnode+1][dim];
    float Xlim[nnode+1][dim];
    const float deltaX = L/Nx;
    const float deltaY = H/Ny;
    const float rho0 = 10;
    float rho [nnode+1];
    float pressio [nnode+1];
    float valor [nnode];
    bool valorB [nnode];
    float vel [nnode+1][dim];
    float ModVel [nnode+1];
    float Temp [nnode+1];
    float phiS [nnode+1];
```

```
    float ae [nnode+1];
    float aw [nnode+1];
    float an [nnode+1];
    float as [nnode+1];
    float ap [nnode+1];
    float bp [nnode+1];
    // TDMA line by line
    float bp_ast [nnode+1];
    float P [nnode+1];
    float R [nnode+1];

    float vect_N [nnode+1][dim];
    float velEntrance [2] = { 0 , u_inf };
    float velocity =  sqrt ( pow(velEntrance[0],2)
    +pow(velEntrance[1],2) );
    bool CosFluid[nnode+1];
    bool CondContorn[nnode+1];
    float criteriConv = pow(10,-8);

    int north = 0; int south = 0;
    int west = 0; int east = 0;
    int point = 0;

// FUNCTIONS' MENTION

void Geometria2D (float L, float H, int Nx, int Ny);
void EquationDiscretization (int nx, int ny,
float deltaX, float deltaY, float rho [nnode],
float rho0, float X2[nnode][dim]);
void EquationDiscretization2 (int nx, int ny,
float deltaX, float deltaY, float rho [nnode],
float rho0, float X2[nnode][dim]);
void IdentifyNodes(int point,int nx,int ny,int nnode);
bool IniciLinea (int num, int nx, int ny);
bool FinalLinea (int num, int nx, int ny);
void Centroide(float COOR[Nx+1][Ny+1][dim],int Nx, int Ny);
float DegtoRad (float angle);
int ParImpar (int N);
void Limits (float l, float h, int nx, int ny);
float Circulation (float vye, float vxn, float vyw,
float vxs, float deltaY,float deltaX);
float Harmonicmean (float dPE, float dPe, float dEe,
float rho0, float rhoP, float rhoE);
void Velocities (int nx, int ny, float deltaX,
```

```
float deltaY, float rho [nnode], float rho0);
void CondInicials (int Nx, int Ny, float deltaX,
float deltaY, float rho0, float X2[nnode][dim],
float L, float H) ;
void pasartablaafichero (float valor [nnode],
int Nx, int tipo);
void pasartablaaficheroMalla (bool valorB [nnode],
int Nx, int tipo );

// FUNCTIONS

void Geometria2D (float l, float h, int nx, int ny) {
    int i = 0; int j = 0;
    float dx= l/nx;
    float dy = h/ny;

   for (j=0; j<=ny; j++) {
         for (i=0; i<=nx; i++) {
            COOR [i][j][0] = (i)*dx;
            COOR [i][j][1] = (j)*dy;
            //rho  [(nx+1)*j+i] = rho0;
         }
    }

    /*
        cout << "---------- Coordenadas globales ----------" << endl;
        for (j=0; j<=ny; j++) {
         for (i=0; i<=nx; i++) {
            cout << "Punto ( " << i << " " << j << " )";
            cout << " = Comp X: " << COOR[i][j][0];
            cout << " // Comp Y: " << COOR[i][j][1] << endl;
         }
        }
        cout << "------ End of Coordenadas globales ------" << endl;
    */

    Centroide (COOR, nx, ny);
    cout << "1. Mesh calculated with sucess" << endl;
}

void Centroide (float  COOR[Nx+1][Ny+1][dim], int nx, int ny) {

    int i=0; int j=0;
```

```
   for (j=0; j<ny; j++) {
         for (i=0; i<nx; i++) {
            X2 [nx*j+i][0] = (COOR[i][j][0]+COOR[i+1][j][0])/2;
            X2 [nx*j+i][1]= (COOR[i][j][1]+COOR[i][j+1][1])/2;
         }
    }
    /*
    int k=0;

    cout << "---------- Coordenadas centroides 2 ----------" << endl;
    for (k=0; k<ny*nx; k++) {
            cout << "Punto " << k;
            cout << " = Comp X: " << X2[k][0];
            cout << " // Comp Y: " << X2[k][1] << endl;
    }
    cout << "------ End of Coordenadas centroides 2 ------" << endl;
    */

}

void IdentifyNodes (int point, int nx, int ny, int nnode) {

    north = point+(nx);
    south = point-(nx);
    west = point-1;
    east = point+1;

    if ( north >= nnode+1 ) {
        north = -1;
    }
    if ( south <= -1 ) {
        south = -1;
    }
    if ( IniciLinea(point, nx, ny) == 1) {
        west = -1;
    }
    if ( FinalLinea(point, nx, ny) == 1) {
        east = -1;
    }
    //cout << "P: "<< point << " / N: " << north << " / S: "
    // << south << " / W: " << west << " / E: " << east << endl;
}

// Function that returns (1) if the node is the last in the row
```

```
// or (0) if it is not
bool FinalLinea (int num, int nx, int ny) {

    int j = 0;
    bool ent = 0;

    while (ent == 0 && j<=Ny) {
        if(num == nx*(j+1)-1) {
            ent = 1;
        } else {
            ent = 0;
        }
        j++;
    }
    return ent;
}


// Function that returns (1) if the node is the first in the row
// or (0) if it is not
bool IniciLinea (int num, int nx, int ny) {

    int j = 0;
    bool ent = 0;

    while (ent == 0 && j<=Ny) {
        if(num == j*nx) {
            ent = 1;
        } else {
            ent = 0;
        }
        j++;
    }
    return ent;
}

// Representation of Blocking Off Method
void plotBOM (bool CosFluid [nnode], int Nx, int Ny) {

    int i = 0;

    for (i=0; i<=nnode; i++) {
        if (FinalLinea (i, Nx, Ny) == 0) {
            if (CosFluid[i] == 1) {
```

```
                cout << "x" << "";
            } else {
                cout << "-" << "";
            }
        } else {
            if (CosFluid[i] == 1) {
                cout << "x" << endl;
            } else {
                cout << "-" << endl;
            }
        }
    }

}

float DegtoRad (float angle){
    float angleRAD = angle * pi / 180;
    return angleRAD;
}

void Limits (float l, float h, int nx, int ny) {
    float deltaX = l/nx;
    float deltaY = h/ny;
    int i = 0; int j = 0;
    int k = 0;

   for (k=0; k<4*Nx; k++) {
            if (j<2) {
                Xlim[nx*j+i][0] = ((i)*deltaX+(i+1)*deltaX)/2;
                Xlim[nx*j+i][1] = -j*deltaY/2;
                CosFluidlim[nx*j+i] = 1;
                i++;
                if (i == Nx) {
                    i=0;
                    j++;
                }
            } else if (j == 2) {
                Xlim[nx*j+i][0] = ((i)*deltaX+(i+1)*deltaX)/2;
                Xlim[nx*j+i][1] = h;
                CosFluidlim[nx*j+i] = 1;
                i++;
                if (i == Nx) {
                    i=0;
                    j++;
```

```
                }
            } else {
                Xlim[nx*j+i][0] = ((i)*deltaX+(i+1)*deltaX)/2;
                Xlim[nx*j+i][1] = h+deltaY;
                CosFluidlim[nx*j+i] = 1;
                i++;
                if (i == Nx) {
                    i=0;
                    j++;
                }
            }
    }
            //cout << "X: " << COORlim [i][j][0] << " Y: "
            //<< COORlim [i][j][1];
            //cout << " Cos: " << CosFluidlim[nx*j+i] << endl;
}

// Funcio que retorna 1 si el num es par o 0 si el num es impar
int ParImpar (int N) {
    if (N % 2 == 0){
        return 1;
    } else{
      return 0;
     }
}

void CondInicials (int Nx, int Ny, float deltaX, float deltaY,
float rho0, float X2[nnode][dim], float L, float H) {

    int i = 0;
    int j = 0;
    int k = 0;

    for (i=0; i<=nnode; i++) {
        CondContorn [i] = false;
        point = i;
        IdentifyNodes (point, Nx, Ny, nnode);


        //cout << "P: "<< point << " / N: " << north << " /
        // S: " << south << " / W: " << west << " /
        // E: " << east << endl;

        if (south == -1 && CondContorn [i] == false){
```

```
        CondContorn [i] = true;
        // Linea de corriente conocida
        vel [i][0] = u_inf;
        vel [i][1] = 0;
        rho[i] = rho0;
        as[i]=0;
        ModVel [i] = sqrt ( pow(vel[i][0],2) + pow(vel[i][1],2) );
        phiS[i] = (rho[i] / rho0) *
        ( + vel[i][0] * X2 [i][1] - vel [i][1] * X2 [i][0] )
    }

    if (west == -1 && CondContorn [i] == false) {
        CondContorn [i] = true;
        // Las lineas de corriente en la entrada
        vel [i][0] = u_inf;
        vel [i][1] = 0;
        rho[i]=rho0;
        aw[i]=0;
        ModVel [i] = sqrt ( pow(vel[i][0],2) + pow(vel[i][1],2) );
        phiS[i] = (rho[i]/rho0)*( +vel[i][0]*X2[i][1]-vel[i][1]*X2[i][0]);
    }

    if (north == -1 && CondContorn [i] == false) {
        CondContorn [i] = true;

        // Linea de corriente conocida
        vel [i][0] = u_inf;
        vel [i][1] = 0;

        rho[i]=rho0;
        an[i]=0;

        ModVel [i] = sqrt ( pow(vel[i][0],2) + pow(vel[i][1],2) );
        phiS[i] = (rho[i]/rho0)*(+vel[i][0]*X2[i][1]-vel[i][1]*X2[i][0]);
    }

    // SUPOSEM UNES VELOCITATS I PHIS INICIALS

    if (CondContorn [i] == false) {
        // ja sabem phi
        rho[i]=rho0;
        vel [i][0] = u_inf * 1;
        vel [i][1] = u_inf * 0;
        phiS[i] = (rho[i]/rho0)*(+vel[i][0]*X2[i][1]-vel[i][1]*X2[i][0]);
```

```
        }
    }
    cout << "2. Condiciones iniciales establecidas" << endl;

    /*cout << "---------- Coordenadas centroides 2 ----------" << endl;
        for (k=0; k<Ny*Nx; k++) {
            cout << "Punto " << k;
            cout << " = Comp X: " << X2[k][0];
            cout << " // Comp Y: " << X2[k][1] << endl;
        }
    cout << "------ End of Coordenadas centroides 2 ------" << endl;*/

}

float Circulation (float vye, float vxn, float vyw, float vxs,
float deltaY, float deltaX) {
    float circ = vye*deltaY - vxn*deltaX - vyw*deltaY + vxs*deltaX;
    return circ;
}


void EquationDiscretization (int Nx, int Ny, float deltaX,
float deltaY, float rho [nnode], float rho0, float X2[nnode][dim]) {

    //float ae = 0; float aw = 0;
    //float an = 0; float as = 0;
    //float ap = 0; //float bp = 0;

    int i = 0; float M = 0;
    float circ;

    int bucles = 0;

    float newphiS = 0;

    while (bucles < 10000) {

        for (i=0; i<=nnode; i++) {
            if ( CondContorn[i] == false) {
                point = i;
                IdentifyNodes (point, Nx, Ny, nnode);

                circ = abs(Circulation(vel[east][1], vel[north][0],
                vel[west][1], vel[south][0],deltaY, deltaX));
```

```
                //cout << circ << endl;

                an [i] = rho [north]/rho0 * deltaX / deltaY;
                as [i]  = rho [south]/rho0 * deltaX / deltaY;
                aw [i] = rho [west]/rho0 * deltaY / deltaX;

                if (east == -1){
                ae [i]  = 0;
                } else {
                ae [i] = rho [east]/rho0 * deltaY / deltaX;
                }

                ap [i] = an [i]  + as [i]  + aw [i]  + ae [i] ;

                // NEW PHIS

                newphiS = ( an[i] *phiS[north] + as[i] *phiS[south] +
                ae[i] *phiS[east] + aw[i] *phiS[west] ) / ap[i] ;

                if ( abs(newphiS - phiS[i]) > criteriConv) {
                    phiS[i] = newphiS;
                }

            }

        }
        bucles++;
    }
    cout << "3. Calculation of the stream function" << endl;
}


void EquationDiscretization2 (int Nx, int Ny, float deltaX,
float deltaY, float rho [nnode], float rho0, float X2[nnode][dim]) {

    int i = 0; float M = 0;
    float circ;

    int bucles = 0;

    float newphiS = 0;

        for (i=0; i<=nnode; i++) {
                point = i;
```

```
            IdentifyNodes (point, Nx, Ny, nnode);

            circ = abs(Circulation(vel[east][1], vel[north][0],
             vel[west][1], vel[south][0], deltaY, deltaX));
            //cout << circ << endl;

            if (north == -1){
            an [i]  = 0;
            } else {
            an [i]  = rho [north]/rho0 * deltaX / deltaY;
            }

            if (south == -1){
            as [i]  = 0;
            } else {
            as [i]  = rho [south]/rho0 * deltaX / deltaY;
            }

            if (west == -1){
            ae [i]  = 0;
            } else {
            aw [i]  = rho [west]/rho0 * deltaY / deltaX;
            }

            if (east == -1){
            ae [i]  = 0;
            } else {
            ae [i]  = rho [east]/rho0 * deltaY / deltaX;
            }

            bp [i] = 0;
            ap [i]  = an[i]  + as[i]  + aw[i]  + ae[i] ;
    }

    for (i=0; i<=nnode; i++) {
            point = i;
            IdentifyNodes (point, Nx, Ny, nnode);

        bp_ast [i] = an[i]*phiS[north] + as[i]*phiS[south] + bp[i];
            P[i] = ae[i] / (ap[i] - aw[i]*P[west]) ;
            R[i] = (bp_ast[i] + aw[i]*R[west]) / (ap[i] - aw[i]*P[west]);

    //bp_ast[i] = ae[i]*phiS[east] + aw[i]*phiS[west] + bp[i];
    //P[i] = an[i] / (ap[i] - as[i]*P[south]);
```

```
        //R[i] = (bp_ast[i] + as[i]*R[south]) / (ap[i] - as[i]*P[south]);


                if ( CondContorn[i] == false) {
                    phiS[i] = P[i]*phiS[east] + R[i] ;
                    //phiS[i] = P[i]*phiS[north] + R[i] ;
                }

        }

    cout << "3. Calculation of the stream function" << endl;
}


float Harmonicmean (float dPE, float dPe, float dEe,
float rho0, float rhoP, float rhoE) {
    float rho0_rhoe = dPE / (dPe/(rho0/rhoP) + dEe/(rho0/rhoE));
    return rho0_rhoe ;
}


void Velocities (int Nx, int Ny, float deltaX,
float deltaY, float rho [nnode], float rho0) {
    float vye = 0;
    float vyw = 0;
    float vxn = 0;
    float vxs = 0;

    float dpx; float dpX;
    float dpy; float dpY;

    int i = 0;

    for (i=0; i<=nnode; i++) {
        point = i;
        IdentifyNodes (point, Nx, Ny, nnode);
        if (CondContorn[i] == 0){

                if (east == -1){
                    dpX =  0; dpx = deltaX/2;
                    dpY = deltaY; dpy = deltaY/2;
                } else {
                    dpX = deltaX; dpx = deltaX/2;
                    dpY = deltaY; dpy = deltaY/2;
```

```
                }

                vye = - Harmonicmean(dpX, dpx, dpx, rho0, rho[i],
                rho[east]) * ( phiS[east] - phiS[i] ) / deltaX;
                vyw = + Harmonicmean(dpX, dpx, dpx, rho0, rho[i],
                rho[west]) * ( phiS[west] - phiS[i] ) / deltaX;
                vxs = - Harmonicmean(dpY, dpy, dpy, rho0, rho[i],
                rho[south]) * ( phiS[south] - phiS[i] ) / deltaY;
                vxn = + Harmonicmean(dpY, dpy, dpy, rho0, rho[i],
                rho[north]) * ( phiS[north] - phiS[i] ) / deltaY;

                vel[i][0] = ( vxn + vxs )/2;
                vel[i][1] = ( vye + vyw )/2;

                ModVel [i] = sqrt ( pow(vel[i][0],2) + pow(vel[i][1],2) );

        }
    }
    cout << "4. Velocity field defined" << endl;
}


// MAIN CODE

int main() {

    int i=0; int j=0;

    Geometria2D (L, H, Nx, Ny);
    Limits (L, H, Nx, Ny);
        pasartablaaficheroMalla (CosFluid, Nx, 1);

    CondInicials (Nx, Ny, deltaX, deltaY, rho0, X2, L, H);
        //pasartablaafichero (phiS, Nx, 1);
        //pasartablaafichero (ModVel, Nx, 2);

    EquationDiscretization2 (Nx, Ny, deltaX, deltaY, rho, rho0, X2);
        pasartablaafichero (phiS, Nx, 2);

    Velocities(Nx, Ny, deltaX, deltaY,rho,rho0);
        pasartablaafichero (ModVel, Nx, 3);

    return 0;
}
```

```
void pasartablaafichero (float valor [nnode], int Nx, int tipo ) {

     ofstream prod;
     if (tipo == 1) {
         prod.open("MallaCanal.dat");
     } else if (tipo == 2) {
         prod.open("FlujoCanal.dat");
     } else if (tipo == 3) {
         prod.open("VelCanal.dat");
     } else {
         cout << "6. Archivo no válido" <<endl;
     }

        if (prod.is_open()) {

        //prod << "# X     Y     Z" << endl;

            int i = 0; int j=0;
        if (tipo==1) {
            for (j=0; j<2; j++) {
                for (i=0; i<=Nx; i++) {
                    prod << COORlim [i][j][0]<<"    "<<COORlim [i][j][1]
                    <<CosFluidlim[Nx*j+i];
                }
            }
        }

            int k = 0;
            int buc = 0;
          while (k <= nnode) {
             //prod << "Coordenadas: ";
             //prod << X2[i][0] <<" " << X2[i][1] << " ";
             //prod << " Cos (1) Fluid (0): " << CosFluid[i] <<endl;
             //prod << X2[i][0] << " " << X2[i][1] << " " << CosFluid[i] <<endl;

             prod << fixed << setprecision(5) << X2[k][0] << "    ";
             prod << fixed << setprecision(5) << X2[k][1] << "    ";
             prod << fixed << setprecision(5) << valor[k] << endl;

            buc++;
            if (buc==Nx){
                prod << " " << endl;
                buc=0;
```

```
            }

             k++;
            }
                 if (tipo == 1) {
                    cout << "  3.1 Stream function results saved in .dat"<<endl;
                } else if (tipo == 2) {
                    cout << "  4.1 Velocity results saved in .dat"<<endl;
                }

                if (tipo==1) {
                    for (j=2; j<4; j++) {
                        for (i=0; i<=Nx; i++) {
                            prod << COORlim [i][j][0]<<"   "<<COORlim [i][j][1]
                            <<CosFluidlim[Nx*j+i];
                    }
                }
        }

        } else {

            cout << "6. Cannot access to file .dat"<<endl;

        }
     prod.close();
}

void pasartablaaficheroMalla (bool valorB [nnode], int Nx, int tipo ) {

     ofstream prod;
     if (tipo == 1) {
         prod.open("MallaCanal.dat");
     } else if (tipo == 2) {
         prod.open("FlujoCanal.dat");
     } else if (tipo == 3) {
         prod.open("VelCanal.dat");
     } else {
         cout << "6. File not valid" <<endl;
     }

        if (prod.is_open()) {

        //prod << "# X     Y     Z" << endl;
```

```
    int i = 0; int j=0; int buc1=0;
if (tipo==1) {
        for (i=0; i<=2*Nx-1; i++) {
            prod << Xlim[i][0]<<"   "<<Xlim [i][1]<< "   " <<
            CosFluidlim[Nx*j+i] << endl;
            buc1++;
            if (buc1==Nx){
                prod << " " << endl;
                buc1 = 0;
            }
        }
}

    int k = 0;
    int buc = 0;
  while (k <= nnode) {
     //prod << "Coordenadas: ";
     //prod << X2[i][0] <<" " << X2[i][1] << " ";
     //prod << " Cos (1) Fluid (0): " << CosFluid[i] <<endl;
     //prod << X2[i][0] << " " << X2[i][1] << " " << CosFluid[i] <<endl;

     prod << fixed << setprecision(5) << X2[k][0] << "   ";
     prod << fixed << setprecision(5) << X2[k][1] << "   ";
     prod << fixed << setprecision(5) << valorB[k] << endl;

    buc++;
    if (buc==Nx){
        prod << " " << endl;
        buc=0;
    }

     k++;
    }
         if (tipo == 1) {
            cout << "  1.1 Resultados de función de corriente
            recogidos en el archivo .dat"<<endl;
        }

    if (tipo==1) {
        for (i=2*Nx; i<4*Nx; i++) {
            prod << Xlim [i][0]<<"   "<<Xlim [i][1]<< "   " <<
            CosFluidlim[Nx*j+i] << endl;
            buc1++;
            if (buc1==Nx){
```

```
                        prod << " " << endl;
                        buc1 = 0;
                    }
                }
        }

        } else {

            cout << "6. Cannot acces to file .dat"<<endl;

        }
    prod.close();
}
```

2 Flow around a cylinder

2.1 Results

*BOM : Blocking Off Method

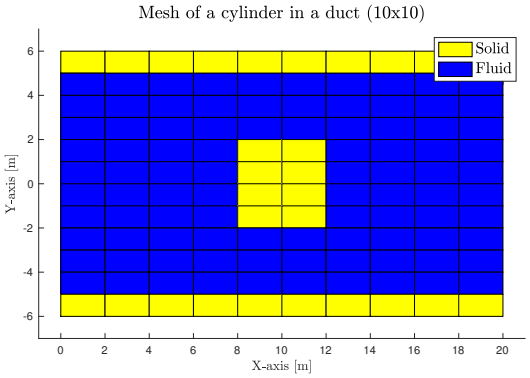

Figure 2.1: BOM - Cylinder with a mesh of dimensions 10x10

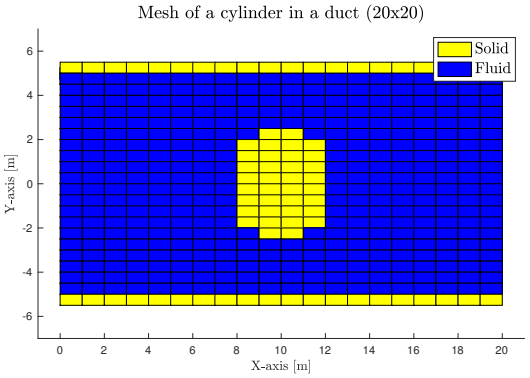

Figure 2.2: BOM - Cylinder with a mesh of dimensions 20x20

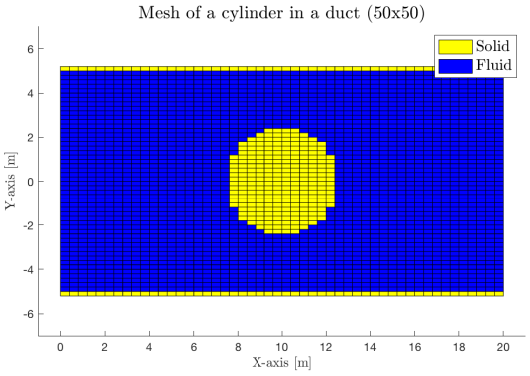

Figure 2.3: BOM - Cylinder with a mesh of dimensions 50x50

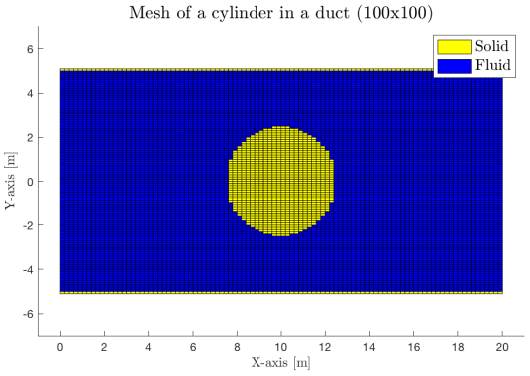

Figure 2.4: BOM - Cylinder with a mesh of dimensions 100x100

2.2 Code in C++: CylinderFlow.cpp

```
#include <iostream>
#include <stdlib.h>
#include <fstream>
#include <math.h>
#include <iomanip>
#include <cstdlib>

using namespace std;

// Constants

const float pi = 3.14159265;

// Problem data

const int Nx = 100; // number of divisions along dir. X
const int Ny = 100; // number of divisions along dir. Y
const int dim = 2; // number of dimensions
const float L = 20; // length of the canal in [m]
const float H = 10; // height of the canal in [m]
const float Rad = 2.5;
const float u_inf = 100; // entrance's velocity  [m]


// VARIABLES

    const int nnode = Nx*Ny-1;
    float COOR[Nx+1][Ny+1][dim];
    float COORlim[Nx+2][4][dim];
    bool CosFluidlim[Nx*4];
    float X2[nnode+1][dim];
    const float deltaX = L/Nx;
    const float deltaY = H/Ny;
    float rho0 = 10;
    float rho [nnode+1];
    float pressio [nnode+1];
    float Temp [nnode+1];
    float phi [nnode+1][dim];
    float phiS [nnode+1];
    float ae [nnode+1];
    float aw [nnode+1];
    float an [nnode+1];
```

```
    float as [nnode+1];
    float ap [nnode+1];
    float bp [nnode+1];
    // TDMA line by line
    float bp_ast [nnode+1];
    float P [nnode+1];
    float R [nnode+1];


    float valor [nnode+1];
    float vel [nnode+1][dim];
    float Xlim[nnode+1][dim];
    float ModVel [nnode+1];
    float vect_N [nnode][dim];
    float velEntrance [2] = { 0 , u_inf };
    float velocity =  sqrt ( pow(velEntrance[0],2)
    +pow(velEntrance[1],2) );
    bool CosFluid[nnode+1];
    bool CondContorn[nnode+1];
    bool CondContornCil[nnode+1];
    float criteriConv = pow(10,-7);

    int north = 0; int south = 0;
    int west = 0; int east = 0;
    int point = 0;

// FUNCTIONS' MENTION

void Geometria2D (float L, float H, int Nx, int Ny);
void EquationDiscretization (int nx, int ny, float deltaX,
float deltaY, float rho [nnode], float rho0, float X2[nnode][dim]);
void EquationDiscretization2 (int Nx, int Ny, float deltaX,
float deltaY, float rho [nnode], float rho0,
float X2[nnode][dim],bool CosFluid[nnode+1]);
void IdentifyNodes (int point, int nx, int ny, int nnode);
bool IniciLinea (int num, int nx, int ny);
bool FinalLinea (int num, int nx, int ny);
//void PerfilNACA (float COOR[Nx+1][Ny+1][dim]);
void BlockingOffMethodCilindre (float X2[nnode][dim], float L,
float H, float R);
void plotBOM (bool CosFluid [nnode], int Nx, int Ny);
void Centroide (float COOR[Nx+1][Ny+1][dim], int Nx, int Ny);
float DegtoRad (float angle);
int ParImpar (int N);
```

```
void Limits (float l, float h, int nx, int ny);
float TangCilindro (float coorX, float R);
float Circulation (float vye, float vxn, float vyw, float vxs,
float deltaY, float deltaX);
float Harmonicmean (float dPE, float dPe, float dEe, float rho0,
float rhoP, float rhoE);
void Velocities (int nx, int ny, float deltaX, float deltaY,
float rho [nnode], float rho0);
//void pasartablaafichero (float valor [nnode], int Nx, int tipo);
void pasartablaafichero (float valor [nnode], int Nx, int tipo,
bool CondContorn[nnode+1]);
void pasartablaaficheroMalla (bool valorB [nnode], int Nx, int tipo) ;
float NACA_yt (float X, float t, float c);

// FUNCTIONS

void Geometria2D (float l, float h, int nx, int ny) {
    int i = 0; int j = 0;

   for (j=0; j<=ny; j++) {
         for (i=0; i<=nx; i++) {
            COOR [i][j][0] = (i)*deltaX;
            COOR [i][j][1] = h/2 - (j)*deltaY ;
        }
    }
    Centroide (COOR, nx, ny);
}


void Centroide (float  COOR[Nx+1][Ny+1][dim], int nx, int ny) {

    int i=0; int j=0;

   for (j=0; j<ny; j++) {
         for (i=0; i<nx; i++) {
            X2 [nx*j+i][0] = (COOR[i][j][0]+COOR[i+1][j][0])/2;
            X2 [nx*j+i][1]= (COOR[i][j][1]+COOR[i][j+1][1])/2;
         }
    }
    /*
    int k=0;
    cout << "---------- Coordenadas centroides 2 ----------" << endl;
    for (k=0; k<ny*nx; k++) {
            cout << "Punto " << k;
```

```
            cout << " = Comp X: " << X2[k][0];
            cout << " // Comp Y: " << X2[k][1] << endl;
    }
    cout << "------ End of Coordenadas centroides 2 ------" << endl;
    */
}

void IdentifyNodes (int point, int nx, int ny, int nnode) {

    north = point+(nx);
    south = point-(nx);
    west = point-1;
    east = point+1;

    if ( north >= nnode+1 ) {
        north = -1;
    }
    if ( south <= 0 ) {
        south = -1;
    }
    if ( IniciLinea(point, nx, ny) == 1) {
        west = -1;
    }
    if ( FinalLinea(point, nx, ny) == 1) {
        east = -1;
    }
    //cout << "P: "<< point << " / N: " << north << " / S: " << south
    << " / W: " << west << " / E: " << east << endl;
}

bool FinalLinea (int num, int nx, int ny) {

    int j = 0;
    bool ent = 0;

    while (ent == 0 && j<=Ny) {
        if(num == nx*(j+1)-1) {
            ent = 1;
        } else {
            ent = 0;
        }
        j++;
    }
    return ent;
```

```
}

bool IniciLinea (int num, int nx, int ny) {

    int j = 0;
    bool ent = 0;

    while (ent == 0 && j<=Ny) {
        if(num == j*nx) {
            ent = 1;
        } else {
            ent = 0;
        }
        j++;
    }
    return ent;

    /*bool ent = 0;
        if(I == 0) {
            ent = 1;
        } else {
            ent = 0;
        }
    return ent; */
}

void plotBOM (bool CosFluid [nnode], int Nx, int Ny) {

    int i = 0;

    for (i=0; i<=nnode; i++) {
        if (FinalLinea (i, Nx, Ny) == 0) {
            if (CosFluid[i] == 1) {
                cout << "x" << "";
            } else {
                cout << "-" << "";
            }
        } else {
            if (CosFluid[i] == 1) {
                cout << "x" << endl;
            } else {
                cout << "-" << endl;
            }
        }
```

```
    }

}

float DegtoRad (float angle){
    float angleRAD = angle * pi / 180;
    return angleRAD;
}

void Limits (float l, float h, int nx, int ny) {
    float deltaX = l/nx;
    float deltaY = h/ny;
    int i = 0; int j = 0;
    int k = 0;

   for (k=0; k<4*Nx; k++) {
              if (j<2) {
                 Xlim[nx*j+i][0] = ((i)*deltaX+(i+1)*deltaX)/2;
                 Xlim[nx*j+i][1] = h/2+j*deltaY/2;
                 CosFluidlim[nx*j+i] = 1;
                 i++;
                 if (i == Nx) {
                     i=0;
                     j++;
                 }
              } else if (j == 2) {
                 Xlim[nx*j+i][0] = ((i)*deltaX+(i+1)*deltaX)/2;
                 Xlim[nx*j+i][1] = -h/2;
                 CosFluidlim[nx*j+i] = 1;
                 i++;
                 if (i == Nx) {
                     i=0;
                     j++;
                 }
              } else {
                 Xlim[nx*j+i][0] = ((i)*deltaX+(i+1)*deltaX)/2;
                 Xlim[nx*j+i][1] = -h/2-deltaY;
                 CosFluidlim[nx*j+i] = 1;
                 i++;
                 if (i == Nx) {
                     i=0;
                     j++;
                 }
              }
```

```
    }
            //cout << "X: " << COORlim [i][j][0] << " Y: " << COORlim [i][j][1];
            //cout << " Cos: " << CosFluidlim[nx*j+i] << endl;
}

// Function that returns (1) if the node is inside of the body
// or (0) if it is in the fluid = CYLINDER
void BlockingOffMethodCilindre (float X2[nnode][dim], float L, float H,
float Rad) {
    int i=0;

    float xc = L / 2;
    float yc = 0;
    float dist2 = 0;

    for (i=0; i<=nnode ; i++){
        dist2 = pow(X2[i][0]-xc,2) + pow(X2[i][1]-yc,2);
        if ( dist2 <= pow(Rad,2) ) {
            CosFluid[i]= 1 ;
        } else {
            CosFluid[i]= 0 ;
        }
    }
    //plotBOM(CosFluid, Nx, Ny);
    cout << "1. Mesh done" << endl;

}

int ParImpar (int N) {
    if (N % 2 == 0){
        return 1;
    } else{
      return 0;
     }
}

float Harmonicmean (float dPE, float dPe, float dEe, float rho0, float rhoP,
float rhoE) {
    float rho0_rhoe = dPE / (dPe/(rho0/rhoP) + dEe/(rho0/rhoE));
    return rho0_rhoe ;
}

void CondInicials (int nx, int ny, float deltaX, float deltaY, float rho [nnode],
float rho0, float X2[nnode][dim]) {
```

```
float xc = L / 2;
float yc = 0;

int i = 0; float M = 0;
float modul = 0;
float x = 0;
int bool_NS = 0;
int bool_WE = 0;

for (i=0; i<=nnode; i++) {
    CondContorn [i] = false;
    CondContornCil [i] = false;
    point = i;
    IdentifyNodes (point, nx, ny, nnode);

    if (CosFluid[i] == 0 && ( CosFluid[north] == 1 || CosFluid[south] == 1 ||
    CosFluid[west] == 1 || CosFluid[east] == 1 ) ){
        phiS[i] = 0; // Conocemos PHI pero no sus velocidades
        rho[i] = rho0;
        CondContorn [i] = true;
        //CondContorn[i] = true;
    }

    if (CosFluid[i] == 1) {
        phiS[i] = 0;
        CondContornCil[i] = true;
        CondContorn[i] = true;
        rho[i]=0;
        vel [i][0] = 0;
        vel [i][1] = 0;
        ModVel [i] = sqrt ( pow(vel[i][0],2) + pow(vel[i][1],2) );
    }

    if (north == -1 && CondContorn [i] == false) {
        CondContorn [i] = true;
        CondContornCil [i] = true;
        // Linea de corriente conocida
        vel [i][0] = u_inf;
        vel [i][1] = 0;
        rho[i]=rho0;
        ModVel [i] = sqrt ( pow(vel[i][0],2) + pow(vel[i][1],2) );
        phiS[i] = (rho[i] / rho0) * ( + vel[i][0] * X2 [i][1] - vel [i][1]
        * X2 [i][0] );
```

```
        }

        if (south == -1 && CondContorn [i] == false){
            CondContorn [i] = true;
            CondContornCil [i] = true;
            // Linea de corriente conocida
            vel [i][0] = u_inf;
            vel [i][1] = 0;
            rho[i] = rho0;
            ModVel [i] = sqrt ( pow(vel[i][0],2) + pow(vel[i][1],2) );
            phiS[i] = (rho[i] / rho0) * ( + vel[i][0] * X2 [i][1] - vel [i][1]
            * X2 [i][0] );
        }

        if (west == -1 && CondContorn [i] == false) {
            CondContorn [i] = true;
            CondContornCil [i] = true;
            // Las lineas de corriente en la entrada
            vel [i][0] = u_inf;
            vel [i][1] = 0;
            rho[i] = rho0;
            ModVel [i] = sqrt ( pow(vel[i][0],2) + pow(vel[i][1],2) );
            phiS[i] = (rho[i] / rho0) * ( + vel[i][0] * X2 [i][1] - vel [i][1]
            * X2 [i][0] );
        }

     // SUPOSEM UNES VELOCITATS I PHIS INICIALS
        if (CondContorn [i] == false) {
            vel [i][0] = u_inf * 0.1;
            vel [i][1] = u_inf * 1;
            rho[i] = rho0;
            ModVel [i] = sqrt ( pow(vel[i][0],2) + pow(vel[i][1],2) );
            phiS[i] = (rho[i] / rho0) * ( + vel[i][0] * X2 [i][1] - vel [i][1]
            * X2 [i][0] );
        }
    }
    cout << "2. Initial conditions established" << endl;
}

float Circulation (float vye, float vxn, float vyw, float vxs,
float deltaY, float deltaX) {
    float circ = vye*deltaY - vxn*deltaX - vyw*deltaY + vxs*deltaX;
    return circ;
```

```
}

void EquationDiscretization (int nx, int ny, float deltaX, float deltaY,
float rho [nnode], float rho0, float X2[nnode][dim]) {

    float xc = L / 2;
    float yc = H / 2;
    float dist2 = 0;

    //float ae = 0; float aw = 0;
    //float an = 0; float as = 0;
    //float ap = 0; //float bp = 0;

    int i = 0;
    float x = 0;
    float circ;

    int bucles = 0;

    float newphiS = 0;

    while (bucles < 6000) {
        //cout << bucles << " ";
        for (i=0; i<=nnode; i++) {
            if (CondContorn[i] == false) {
                point = i;
                IdentifyNodes (point, nx, ny, nnode);

                circ = abs(Circulation(vel[east][1], vel[north][0],
                vel[west][1], vel[south][0], deltaY, deltaX));

                //cout << circ << endl;

                an[i] = rho [north]/rho0 * deltaX / deltaY;
                as[i] = rho [south]/rho0 * deltaX / deltaY;
                aw[i] = rho [west]/rho0 * deltaY / deltaX;

                if (east == -1){
                ae[i] = 0;
                } else {
                ae[i] = rho [east]/rho0 * deltaY / deltaX;
                }

                ap[i] = an[i] + as[i] + aw[i] + ae[i];
```

```
                // NEW PHIS
                newphiS = ( an[i]*phiS[north] + as[i]*phiS[south] +
                ae[i]*phiS[east] + aw[i]*phiS[west] ) / ap[i];

                if ( abs(newphiS - phiS[i]) > criteriConv) {
                    phiS[i] = newphiS;
                }

            }

        }
        bucles++;
    }
    cout << "3. Cálculo de la función de corriente" << endl;
}

void Velocities (int nx, int ny, float deltaX, float deltaY,
float rho [nnode], float rho0) {
    float vye = 0;
    float vyw = 0;
    float vxn = 0;
    float vxs = 0;

    float dpx; float dpX;
    float dpy; float dpY;

    int i = 0;

    for (i=0; i<=nnode; i++) {
        point = i;
        IdentifyNodes (point, Nx, Ny, nnode);
        if (CondContornCil[i] == false){
                if (east == -1){
                    dpX =  0; dpx = deltaX/2;
                    dpY = deltaY; dpy = deltaY/2;
                } else {
                    dpX = deltaX; dpx = deltaX/2;
                    dpY = deltaY; dpy = deltaY/2;
                }

                vye = - Harmonicmean(dpX, dpx, dpx, rho0, rho[i],
                rho[east]) * ( phiS[east] - phiS[i] ) / deltaX;
                vyw = + Harmonicmean(dpX, dpx, dpx, rho0, rho[i],
```

```
                rho[west]) * ( phiS[west] - phiS[i] ) / deltaX;
                vxs = - Harmonicmean(dpY, dpy, dpy, rho0, rho[i],
                rho[south]) * ( phiS[south] - phiS[i] ) / deltaY;
                vxn = + Harmonicmean(dpY, dpy, dpy, rho0, rho[i],
                rho[north]) * ( phiS[north] - phiS[i] ) / deltaY;

                vel[i][0] = ( vxn + vxs )/2;
                vel[i][1] = ( vye + vyw )/2;

                ModVel [i] = sqrt ( pow(vel[i][0],2) + pow(vel[i][1],2) );

        }
    }
    cout << "4. Velocity field obtained" << endl;
}

void EquationDiscretization2 (int Nx, int Ny, float deltaX,
float deltaY, float rho [nnode], float rho0,
float X2[nnode][dim],bool CosFluid[nnode+1]) {

    int i = 0; float M = 0;
    float circ;

    int bucles = 0;

    float newphiS = 0;

        for (i=0; i<=nnode; i++) {
                point = i;
                IdentifyNodes (point, Nx, Ny, nnode);

                circ = abs(Circulation(vel[east][1], vel[north][0],
                vel[west][1], vel[south][0], deltaY, deltaX));

                if (north == -1){
                an [i]  = 0;
                } else {
                an [i]  = rho [north]/rho0 * deltaX / deltaY;
                }

                if (south == -1){
                as [i]  = 0;
                } else {
                as [i]  = rho [south]/rho0 * deltaX / deltaY;
```

```
            }

            if (west == -1){
            ae [i]  = 0;
            } else {
            aw [i]  = rho [west]/rho0 * deltaY / deltaX;
            }

            if (east == -1){
            ae [i]  = 0;
            } else {
            ae [i]  = rho [east]/rho0 * deltaY / deltaX;
            }

            bp [i] = 0;
            ap [i]  = an[i]  + as[i]  + aw[i]  + ae[i] ;
    }

    for (i=0; i<=nnode; i++) {
            point = i;
            IdentifyNodes (point, Nx, Ny, nnode);
                //bp_ast [i] = an[i]*phiS[north] +
                as[i]*phiS[south] + bp[i];
                //P[i] = ae[i] / (ap[i] - aw[i]*P[west]) ;
                //R[i] = (bp_ast[i] + aw[i]*R[west]) /
                (ap[i] - aw[i]*P[west]);

                bp_ast[i] = ae[i]*phiS[east] + aw[i]*phiS[west] + bp[i];
                P[i] = an[i] / (ap[i] - as[i]*P[south]);
                R[i] = (bp_ast[i] + as[i]*R[south]) /
                (ap[i] - as[i]*P[south]);

                if (CosFluid[i] == 1) {
                    P[i]=0;
                    R[i]=0;
                }

            if ( CondContorn[i] == false) {
                //phiS[i] = P[i]*phiS[east] + R[i] ;
                phiS[i] = P[i]*phiS[north] + R[i] ;
            }

    }
```

```
    cout << "3. Stream function obtained" << endl;
}


// MAIN CODE

int main() {

    int i=0; int j=0;

    Geometria2D (L, H, Nx, Ny);
    Limits (L, H, Nx, Ny);
    BlockingOffMethodCilindre (X2, L, H, Rad);
        pasartablaaficheroMalla (CosFluid, Nx, 1);
    CondInicials (Nx, Ny, deltaX, deltaY, rho, rho0, X2);

    EquationDiscretization (Nx, Ny, deltaX, deltaY, rho, rho0, X2);
    //EquationDiscretization2 (Nx, Ny, deltaX, deltaY,
    rho, rho0, X2, CosFluid);
        pasartablaafichero (phiS, Nx, 1, CondContornCil);
    Velocities(Nx, Ny, deltaX, deltaY,rho,rho0);
        pasartablaafichero (ModVel, Nx, 2, CondContornCil);
    return 0;
}

void pasartablaafichero (float valor [nnode], int Nx, int tipo,
bool CondContornCil[nnode]) {

     ofstream prod;
     if (tipo == 1) {
         prod.open("FlujoCilindro.dat");
     } else if (tipo == 2) {
         prod.open("VelCilindro.dat");
     } else if (tipo == 3) {
         prod.open("TempCilindro.dat");
     } else if (tipo == 4){
         prod.open("PresionCilindro.dat");
     } else if (tipo == 5) {
         prod.open("DensidadCilindro.dat");
     } else {
         cout << "6. Archivo no válido" <<endl;
     }
```

```
         prod << "# X    Y    Z" << endl;

        if (prod.is_open()) {
             int k = 0;
             int buc = 0;
           while (k <= nnode) {
              //prod << "Coordenadas: ";
              //prod << X2[i][0] <<" " << X2[i][1] << " ";
              //prod << " Cos (1) Fluid (0): " << CosFluid[i] <<endl;
              //prod << X2[i][0] << " " << X2[i][1] << " " <<
              CosFluid[i] <<endl;

                 prod << fixed << setprecision(5) << X2[k][0] << "    ";
                 prod << fixed << setprecision(5) << X2[k][1] << "    ";
                 prod << fixed << setprecision(5) << valor[k] << endl;


             buc++;
             if (buc==Nx){
                 prod << " " << endl;
                 buc=0;
             }

              k++;
             }
                  if (tipo == 1) {
                     cout << "  3.1 Resultados de función de corriente
                     recogidos en el archivo .dat"<<endl;
                 } else if (tipo == 2) {
                     cout << "  4.1 Resultados de velocidades
                     recogidos en el archivo .dat"<<endl;
                 }

         } else {

             cout << "6. No he accedido al .dat"<<endl;

         }
      prod.close();
}


void pasartablaaficheroMalla (bool valorB [nnode], int Nx, int tipo) {
```

```
ofstream prod;
if (tipo == 1) {
    prod.open("MallaCilindro2.dat");
} else if (tipo == 2) {
    prod.open("FlujoCanal.dat");
} else if (tipo == 3) {
    prod.open("VelCanal.dat");
}

   if (prod.is_open()) {

   prod << "# X    Y    Z" << endl;

       int i = 0; int j=0; int buc1=0;
   if (tipo==1) {
         for (i=0; i<=2*Nx-1; i++) {
             prod << Xlim[i][0]<<"   "<<Xlim [i][1]
             << "   " << CosFluidlim[Nx*j+i] << endl;
             buc1++;
             if (buc1==Nx){
                 prod << " " << endl;
                 buc1 = 0;
             }
         }
   }

       int k = 0;
       int buc = 0;
     while (k <= nnode) {
        //prod << "Coordenadas: ";
        //prod << X2[i][0] <<" " << X2[i][1] << " ";
        //prod << " Cos (1) Fluid (0): " << CosFluid[i] <<endl;
        //prod << X2[i][0] << " " << X2[i][1] << " "
        << CosFluid[i] <<endl;

        prod << fixed << setprecision(5) << X2[k][0] << "   ";
        prod << fixed << setprecision(5) << X2[k][1] << "   ";
        prod << fixed << setprecision(5) << valorB[k] << endl;

       buc++;
       if (buc==Nx){
           prod << " " << endl;
           buc=0;
       }
```

```
            k++;
           }
                if (tipo == 1) {
                   cout << "  1.1 Resultados de función de corriente
                   recogidos en el archivo .dat"<<endl;
               }

           if (tipo==1) {
               for (i=2*Nx; i<4*Nx; i++) {
                   prod << Xlim [i][0]<<"   "<<Xlim [i][1]<< "    " <<
                   CosFluidlim[Nx*j+i] << endl;
                   buc1++;
                   if (buc1==Nx){
                       prod << " " << endl;
                       buc1 = 0;
                   }
               }
       }

       } else {

           cout << "6. No he accedido al .dat"<<endl;

       }
    prod.close();
}
```

3 Flow around a NACA airfoil

3.1 Results

*BOM : Blocking Off Method

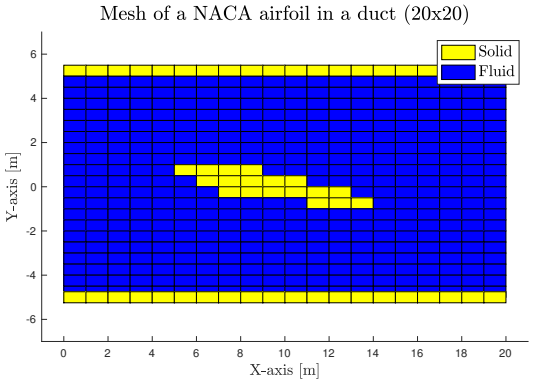

Figure 3.1: BOM - NACA airfoil in a duct with a mesh of dimensions 20x20

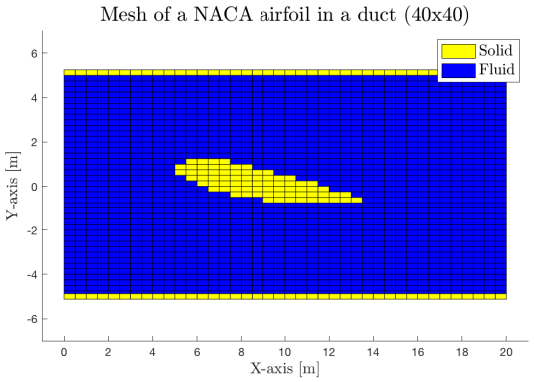

Figure 3.2: BOM - NACA airfoil in a duct with a mesh of dimensions 40x40

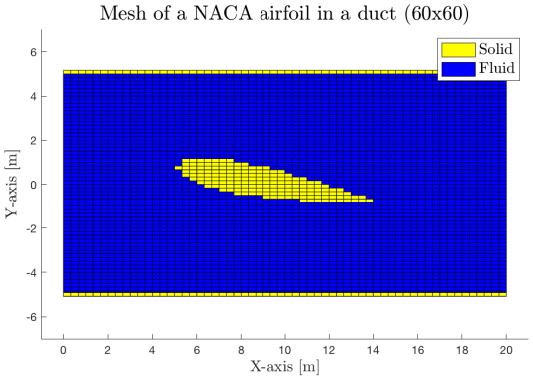

Figure 3.3: BOM - NACA airfoil in a duct with a mesh of dimensions 60x60

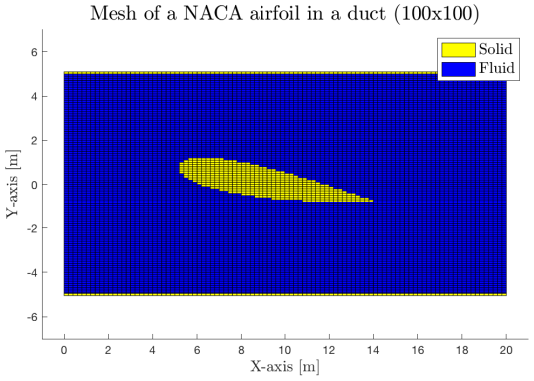

Figure 3.4: BOM - NACA airfoil in a duct with a mesh of dimensions 100x100

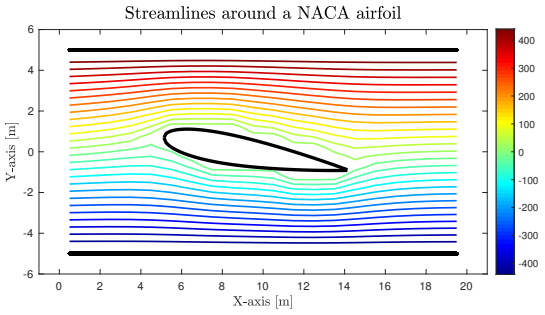

Figure 3.5: Streamlines around a NACA airfoil - Mesh dimension: 20x20

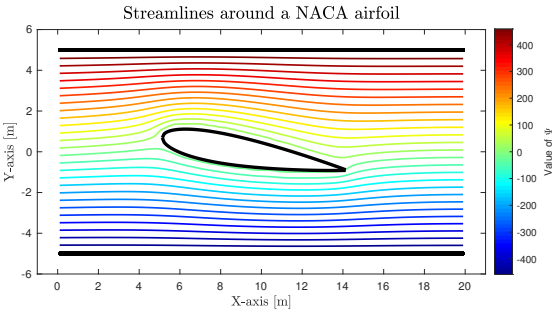

Figure 3.6: Streamlines around a NACA airfoil - Mesh dimension: 100x100

3.2 Code in C++: NACAFlow.cpp