Universitat Politècnica de Catalunya

Study for the numerical resolution of conservation equations of mass, momentum and energy to be applied in different engineering problems: case 5.

Anexo A: Códigos

Grado en Ingeniería en Vehículos Aeroespaciales

Julio 2015

Autor: Miguel Menéndez Melgoso

Director: Carlos David Pérez Segarra

Co-Director: Asensio Oliva Llena

Contenido

1. Two-Dimensional Transient Conduction Problem

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <vector>
#include <fstream>
#include <direct.h>
#include <time.h>
#include <funciones_DC.h>

using namespace std;

//parametros del problema
        // Geométricos
double const Lx = 1.1;
double const Ly = 0.8;
double const P1x = 0.5;
double const P1y = 0.4;
double const P2x = 0.5;
double const P2y = 0.7;
double const P3x = 0.65;
double const P3y = 0.56;
double const P4x = 0.74;
double const P4y = 0.72;
int const Nx = 110;
int const Ny =80;
        // Temporales
```

```
int const t_limite = 5000;
double Time;
int const CONT=100000;
double time_step = 0.0625;
        // Materiales
double const rho1 = 1500.;
double const Cp1 = 750.;
double const K1 = 170.;
double const rho2 = 1600.;
double const Cp2 = 770.;
double const K2 = 140.;
double const rho3 = 1900.;
double const Cp3 = 810.;
double const K3 = 200.;
double const rho4 = 2500.;
double const Cp4 = 930.;
double const K4 = 140.;

//Condiciones
        // Isoterma inferior
double const Tb = 23+273.15;
        // Flujo superior
double const q_flow = -60.;
        // Conveccion
double const Tg = 33+273.15;
double const alfa = 9.;
```

```
class material{
    public:
    double rho, cp, K;
    material(double a, double b, double c){
            rho=a;
            cp=b;
            K=c;
    }
};

class Volumen_Control{
    int nx,ny;
    public:

    double dx,dy;
    vector<double> X,Y;
    vector<double> Kp,Rho,Cp;
    vector<double> ke,kw,kn,ks;
    vector<double> ae,aw,an,as,b;
    vector<double> T,T0;

    void Coordenadas(){
        int i,j,k;

          for(j=0; j<ny; j++){
              for(i=0; i<nx; i++){
                  k=nx*j+i;
```

```
                X.at(k)=dx*(i-0.5);
                Y.at(k)=dy*(j-0.5);

                if(i==0){
                        X.at(k)=0;
                }
                if(i==nx-1){
                        X.at(k)=Lx;
                }
                if(j==0){
                        Y.at(k)=0;
                }
                if(j==ny-1){
                        Y.at(k)=Ly;
                }
            }
        }
}

void Propiedades(material m1, material m2, material m3, material m4){
    int i,j,k;
    double x,y;

    for(j=0; j<ny; j++){
        for(i=0; i<nx; i++){
            k=nx*j+i;
```

```
x=X.at(k);
y=Y.at(k);

if(x>=0 && x<P1x && y>=0 && y<P1y){
        Kp.at(k)=m1.K;
        Cp.at(k)=m1.cp;
        Rho.at(k)=m1.rho;
}
if(x>P1x && x<=Lx && y>=0 && y<P2y){
         Kp.at(k)=m2.K;
         Cp.at(k)=m2.cp;
         Rho.at(k)=m2.rho;
}
if(x>=0 && x<P2x && y>P1y && y<=Ly){
        Kp.at(k)=m3.K;
        Cp.at(k)=m3.cp;
        Rho.at(k)=m3.rho;
}
if(x>P2x && x<=Lx && y>P2y && y<=Ly){
         Kp.at(k)=m4.K;
         Cp.at(k)=m4.cp;
         Rho.at(k)=m4.rho;
}
if(x==P1x && y>=0 && y<P1y){
          Kp.at(k)=(m1.K+m2.K)*0.5;
          Cp.at(k)=(m1.cp+m2.cp)*0.5;
          Rho.at(k)=(m1.rho+m2.rho)*0.5;
}
```

```
if(x>=0 && x<P1x && y==P1y){
        Kp.at(k)=(m1.K+m3.K)*0.5;
        Cp.at(k)=(m1.cp+m3.cp)/2;
        Rho.at(k)=(m1.rho+m3.rho)/2;
}
if(x==P2x && y>P1y && y<P2y){
         Kp.at(k)=(m2.K+m3.K)/2;
         Cp.at(k)=(m2.cp+m3.cp)/2;
         Rho.at(k)=(m2.rho+m3.rho)/2;
}
if(x==P2x && y>P2y && y<=Ly){
         Kp.at(k)=(m3.K+m4.K)/2;
         Cp.at(k)=(m3.cp+m4.cp)/2;
         Rho.at(k)=(m3.rho+m4.rho)/2;
}
if(x>P2x && x<=Lx && y==P2y){
        Kp.at(k)=(m2.K+m4.K)/2;
        Cp.at(k)=(m2.cp+m4.cp)/2;
        Rho.at(k)=(m2.rho+m4.rho)/2;
}
if(x==P1x && y==P1y){
         Kp.at(k)=(m1.K+m2.K+m3.K)/3;
         Cp.at(k)=(m1.cp+m2.cp+m3.cp)/3;
         Rho.at(k)=(m1.rho+m2.rho+m3.rho)/3;
}
if(x==P2x && y==P2y){
         Kp.at(k)=(m2.K+m3.K+m4.K)/3;
         Cp.at(k)=(m2.cp+m3.cp+m4.cp)/3;
```

```
                    Rho.at(k)=(m2.rho+m3.rho+m4.rho)/3;
                }
            }
        }
    }

void MediaArmonica(){
    int i,j,k;
    double KP,KW,KE,KN,KS;

    for(j=1; j<ny-1; j++){
        for(i=1; i<nx-1; i++){
            k=nx*j+i;

            KP=Kp.at(k);
            KW=Kp.at(k-1);
            KE=Kp.at(k+1);
            KN=Kp.at(k+nx);
            KS=Kp.at(k-nx);

            ke.at(k)=dx/((0.5*dx)/KP+(0.5*dx)/KE);
            kw.at(k)=dx/((0.5*dx)/KP+(0.5*dx)/KW);
            kn.at(k)=dy/((0.5*dy)/KP+(0.5*dy)/KN);
            ks.at(k)=dy/((0.5*dy)/KP+(0.5*dy)/KS);

            if(i==1){
                kw.at(k)=KW;
            }
```

```
                if(i==nx-2){
                        ke.at(k)=KE;
                }
                if(j==1){
                        ks.at(k)=KS;
                }
                if(j==ny-2){
                        kn.at(k)=KN;
                }

        }
    }
}

void Coeficientes(){
    int i,j,k;
    double AP,AE,AW,AN,AS;
    double RHO,CP,KP,KE,KW,KN,KS;
    double dpe,dpw,dpn,dps;

    for(j=1; j<ny-1; j++){
        for(i=1; i<nx-1; i++){
                k=nx*j+i;

                RHO=Rho.at(k);
                CP=Cp.at(k);
                KP=Kp.at(k);
```

```
KE=ke.at(k);
KW=kw.at(k);
KN=kn.at(k);
KS=ks.at(k);

dpe=dx;
dpw=dx;
dpn=dy;
dps=dy;

if(i==1){
        dpw=dx/2;
}
if(i==nx-2){
        dpe=dx/2;
}
if(j==1){
        dps=dy/2;
}
if(j==ny-2){
          dpn=dy/2;
}

AP=RHO*CP*dx*dy/time_step;

AE=KE*dy/dpe;
AW=KW*dy/dpw;
AN=KN*dx/dpn;
```

```
                AS=KS*dx/dps;

                ae.at(k)=AE/AP;
                aw.at(k)=AW/AP;
                an.at(k)=AN/AP;
                as.at(k)=AS/AP;

            }
        }
    }

    void Solver(){
        int i,j,k;
        double t0,TE,TW,TN,TS;
        double AE,AW,AN,AS,B;
        double KP,dps,dpe;

        for(j=1; j<ny-1; j++){
            for(i=1; i<nx-1; i++){
                k=nx*j+i;

                t0=T0.at(k);
                TN=T0.at(k+nx);
                TS=T0.at(k-nx);
                TE=T0.at(k+1);
                TW=T0.at(k-1);

                AE=ae.at(k);
```

```
            AW=aw.at(k);
            AN=an.at(k);
            AS=as.at(k);

            B=t0*(1-AE-AW-AN-AS);

            T.at(k)=TE*AE+TW*AW+TN*AN+TS*AS+B;

        }
}

for(i=1; i<nx-1; i++){
        j=ny-1;
        k=nx*j+i;

        t0=T.at(k);
        TS=T.at(k-nx);

        dps=dy*0.5;
        KP=Kp.at(k);

        T.at(k)=TS-(q_flow*dps/KP);
}

for(i=1; i<nx-1; i++){
        j=0;
        k=nx*j+i;
```

```
        T.at(k)=Tb;
}

for(j=1; j<ny-1; j++){
        i=0;
        k=nx*j+i;

        t0=T.at(k);
        TE=T.at(k+1);

        dpe=dx*0.2;
        KP=Kp.at(k);

        T.at(k)=((alfa*dpe/KP)*Tg+TE)/(1+(alfa*dpe/KP));
}

for(j=1; j<ny-1; j++){
        i=nx-1;
        k=nx*j+i;

        T.at(k)=(8+0.005*Time)+273.15;
}

i=0;
j=0;
k=nx*j+i;
T.at(k)=T.at(k+1);
```

```
    i=nx-1;
    j=0;
    k=nx*j+i;
    T.at(k)=T.at(k-1);

    i=0;
    j=ny-1;
    k=nx*j+i;
    T.at(k)=T.at(k-nx);

    i=nx-1;
    j=ny-1;
    k=nx*j+i;
    T.at(k)=T.at(k-nx);

}

Volumen_Control(int a2, int b2){
            nx=a2+2;
            ny=b2+2;

            dx=Lx/a2;
            dy=Ly/b2;

            for(int i=0; i<nx*ny; i++){

                X.push_back(0.);
                Y.push_back(0.);
```

```
                    Kp.push_back(0.);
                    Rho.push_back(0.);
                    Cp.push_back(0.);
                    ke.push_back(0.);
                    kw.push_back(0.);
                    kn.push_back(0.);
                    ks.push_back(0.);
                    ae.push_back(0.);
                    aw.push_back(0.);
                    an.push_back(0.);
                    as.push_back(0.);
                    b.push_back(0.);
                    T0.push_back(8+273.15);
                    T.push_back(0.);
                }
    }
};

int main (void){
ofstream solucion;
int iter, cont, n, k1,k2,aux, conta;
vector<double> z;
solucion.open("MMENENDEZ.txt",std::ofstream::out);
iter=0;
cont=0;
conta=0;
Time=0;
```

```
Volumen_Control VC(Nx,Ny);

material M1(rho1,Cp1,K1);
material M2(rho2,Cp2,K2);
material M3(rho3,Cp3,K3);
material M4(rho4,Cp4,K4);

VC.Coordenadas();
VC.Propiedades(M1,M2,M3,M4);
VC.MediaArmonica();
VC.Coeficientes();

cout<<time_step<<endl;
system("pause");

mkdir("C:/Users/Miguel/Documents/ProyectoFinal/ProblemaConduccion/FILE");



z = multi( 0. , VC.T);

n-1;
aux=0;
while(aux==0){
        n++;
        if(VC.X.at(n)>P3x && VC.Y.at(n)>P3y){
                aux=1;
        }
}
```

```
k1=n;


n=-1;
aux=0;
while(aux==0){
        n++;
        if(VC.X.at(n)>P4x && VC.Y.at(n)>P4y){
                aux=1;
        }
}
k2=n;


while (Time<t_limite){
    iter++;
    cont++;
    Time=Time+time_step;


    VC.Solver();


    if(cont==CONT){
            cout<<"Max cont"<<" "<<Time<<endl;
            write_vtu_2D (int(time) , Nx+2 , Ny+2 , VC.X , VC.Y , z , z , z , z , VC.T);
    }


    conta++;
    if(conta==16){
            solucion<<Time<<"    "<<VC.T.at(k1)<<"    "<<VC.T.at(k2)<<endl;
            conta=0;
```

```
    }

    VC.T0=VC.T;

    cout<<Time<<endl;
}

write_vtu_2D (int(time) , Nx+2 , Ny+2 , VC.X , VC.Y , z , z , z , z , VC.T);

solucion.close();
}
```

2. Unsteady convection and diffusion

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <vector>
#include <fstream>
#include <direct.h>
#include <conio.h>
#include <windows.h>
#include <funciones_C.h>

using namespace std;

int const Nx = 400;
int const Ny = 200;
double const Lx = 2;
double const Ly = 1;

double const rho = 10;
double const gamma = 1;

double timestep = 1;
double counter = 100;
double MaxError = 1e-5;

int const esquema = 1;
```

```
class ContVol{

    public:
    double nx, ny;
    double ax, ay;

    vector<double> x, y;
    vector<double> u, v;
    vector<double> Fe, Fw, Fn, Fs;
    vector<double> De, Dw, Dn, Ds;
    vector<double> ap, ae, aw, an, as, b;
    vector<double> t, t0;
    vector<double> MAT, MatComp;
    vector<double> SOL;

    void Coordenadas(){
        int i, j, k;
        double H;

        H = Lx/2;

        for(j=0; j<ny; j++){
            for(i=0; i<nx; i++){
                k = nx * j + i;

                x[k] = (Lx/(nx-1)) * i - H;
                y[k] = (Ly/(ny-1)) * j;
```

```
            }
        }

}

void Velocidad(){
    int i, j, k;
    double X, Y;

    for(j=0; j<ny; j++){
            for(i=0; i<nx; i++){
                    k = nx * j + i;

                    X = x[k];
                    Y = y[k];

                    u[k] = 2 * Y * (1 - X * X);
                    v[k] = -2 * X * (1 - Y * Y);

            }
    }

}

void CalcularFD(){
    int i, j, k;
    double UE, UW, VN, VS;
    double dpe, dpw, dpn, dps;
```

```
double XE, XW, XN, XS;
double YE, YW, YN, YS;

dpe = ax;
dpw = ax;
dpn = ay;
dps = ay;

for(j=1; j<ny-1; j++){
        for(i=1; i<nx-1; i++){
                k = nx * j + i;

                XE = x[k] + 0.5 * ax;
                XW = x[k] - 0.5 * ax;
                XN = x[k];
                XS = x[k];

                YE = y[k];
                YW = y[k];
                YN = y[k] + 0.5 * ay;
                YS = y[k] - 0.5 * ay;

                UE = 2 * YE * (1 - XE * XE);
                UW = 2 * YW * (1 - XW * XW);

                VN = -2 * XN * (1 - YN * YN);
                VS = -2 * XS * (1 - YS * YS);
```

```
                Fe[k] = rho * UE * ay;
                Fw[k] = rho * UW * ay;
                Fn[k] = rho * VN * ax;
                Fs[k] = rho * VS * ax;

                De[k] = gamma * ay / dpe;
                Dw[k] = gamma * ay / dpw;
                Dn[k] = gamma * ax / dpn;
                Ds[k] = gamma * ax / dps;

            }
        }

    }

    void Coeficientes(){
        int i, j, k;
        double Ae, Aw, An, As;
        double PE, PW, PN, PS;
        double AP, AE, AW, AN, AS, A0;
        double FE, FW, FN, FS;
        double DE, DW, DN, DS;
        double ple, plw, pln, pls;
        double X;

        A0 = rho * ax * ay / timestep;

        for(j=1; j<ny-1; j++){
```

```
for(i=1; i<nx-1; i++){
        k = nx * j + i;

        FE = Fe[k];
        FW = Fw[k];
        FN = Fn[k];
        FS = Fs[k];

        DE = De[k];
        DW = Dw[k];
        DN = Dn[k];
        DS = Ds[k];

        PE = fabs( FE / DE );
        PW = fabs( FW / DW );
        PN = fabs( FN / DN );
        PS = fabs( FS / DS );

        if(esquema == 1){
                Ae = 1;
                Aw = 1;
                An = 1;
                As = 1;
        }

        if(esquema == 2){
                Ae = 1 - 0.5 * PE;
                Aw = 1 - 0.5 * PW;
```

```
        An = 1 - 0.5 * PN;
        As = 1 - 0.5 * PS;
}

if(esquema == 3){
        Ae = max(0. , 1 - 0.5 * PE);
        Aw = max(0. , 1 - 0.5 * PW);
        An = max(0. , 1 - 0.5 * PN);
        As = max(0. , 1 - 0.5 * PS);
}

if(esquema == 4){
        if(PE != 0){
            Ae = PE / (exp(PE) - 1);
        }
        if(PE == 0){
            Ae = 1;
        }

        if(PW != 0){
            Aw = PW / (exp(PW) - 1);
        }
        if(PW == 0){
            Aw = 1;
        }

        if(PN != 0){
            An = PN / (exp(PN) - 1);
```

```
        }
        if(PN == 0){
            An = 1;
        }

        if(PS != 0){
            As = PS / (exp(PS) - 1);
        }
        if(PS == 0){
            As = 1;
        }
}

if(esquema == 5){
        ple = 1 - 0.1 * PE;
        plw = 1 - 0.1 * PW;
        pln = 1 - 0.1 * PN;
        pls = 1 - 0.1 * PS;

        Ae = max(0. , ple * ple * ple * ple * ple);
        Aw = max(0. , plw * plw * plw * plw * plw);
        An = max(0. , pln * pln * pln * pln * pln);
        As = max(0. , pls * pls * pls * pls * pls);
}

AE = DE * Ae + max(-FE , 0.);
AW = DW * Aw + max(FW , 0.);
AN = DN * An + max(-FN , 0.);
```

```
            AS = DS * As + max(FS , 0.);


            AP = A0 + AE + AW + AN + AS;


            ap[k] = AP;
            ae[k] = -AE;
            aw[k] = -AW;
            an[k] = -AN;
            as[k] = -AS;
        }
}

for(i=0; i<nx; i++){
        j = 0;
        k = nx * j + i;


        X = x[k];


        if(X > 0){
            an[k] = -1;
        }
}
/*printv(nx, ny, ap);
cout<<endl;
printv(nx, ny, ae);
cout<<endl;
printv(nx, ny, aw);
cout<<endl;
```

```
    printv(nx, ny, an);
    cout<<endl;
    printv(nx, ny, as);
    system("pause");*/
}

void RecalcularB(){
    int i, j, k;
    double A0;
    double X;

    A0 = rho * ax * ay / timestep;

    for(j=1; j<ny-1; j++){
        for(i=1; i<nx-1; i++){
            k = nx * j + i;

            b[k] = t0[k] * A0;
        }
    }

    for(i=0; i<nx; i++){
        j = ny - 1;
        k = nx * j + i;

        b[k] = 1 - tanh(10);

        j = 0;
```

```
            k = nx * j + i;

            X = x[k];

            if(X <= 0){
                b[k] = 1 + tanh(10 * (2 * X + 1));
            }
        }

        for(j=1; j<ny-1; j++){
            i = 0;
            k = nx * j + i;

            b[k] = 1 - tanh(10);

            i = nx - 1;
            k = nx * j + i;

            b[k] = 1 - tanh(10);
        }
        /*printv(nx, ny, b);
        system("pause");*/
    }

    void Montar(){

        cambiarc(as.size() , 5 , MatComp , 0 , 0 , as.size() - 1 , as);
        cambiarc(aw.size() , 5 , MatComp , 1 , 0 , aw.size() - 1 , aw);
```

```
    cambiarc(ap.size() , 5 , MatComp , 2 , 0 , ap.size() - 1 , ap);
    cambiarc(ae.size() , 5 , MatComp , 3 , 0 , ae.size() - 1 , ae);
    cambiarc(an.size() , 5 , MatComp , 4 , 0 , an.size() - 1 , an);

    //printv(5, nx*ny , MatComp);
   // system("pause");
}

void Solver(){
    t = GaussSeidel(MatComp , b , nx);
}

void Datos(){
    int i, j, k;
    vector<double> datax, datay, dataz, datasol;
    ofstream data;

    for(i=0; i<nx; i++){
            j = 0;
            k = nx * j + i;

            datax.push_back(x[k]);
            datay.push_back(t[k]);
    }

    data.open("Covection_Diffusion.dat",std::ofstream::out);

    data<<"X"<<"    "<<"T"<<endl;
```

```
    for(i=0; i<datax.size(); i++){
            data<<datax[i]<<"    "<<datay[i]<<endl;
    }

    data.close();
    datax.clear();
    datay.clear();

}

ContVol(int aa, int bb){
        double vsize;
        double matsize, MatCompSize;

        nx = aa + 2;
        ny = bb + 2;

        ax = Lx / (aa + 1);
        ay = Ly / (bb + 1);

        vsize = (nx * ny);
        matsize = (2 * nx + 1) * (vsize);
        MatCompSize = vsize * 5;

        x.resize(vsize, 0);
        y.resize(vsize, 0);

        u.resize(vsize, 0);
```

```
v.resize(vsize, 0);

Fe.resize(vsize, 0);
Fw.resize(vsize, 0);
Fn.resize(vsize, 0);
Fs.resize(vsize, 0);

De.resize(vsize, 0);
Dw.resize(vsize, 0);
Dn.resize(vsize, 0);
Ds.resize(vsize, 0);

ap.resize(vsize, 1);
ae.resize(vsize, 0);
aw.resize(vsize, 0);
an.resize(vsize, 0);
as.resize(vsize, 0);

b.resize(vsize, 0);

t.resize(vsize, 0);
t0.resize(vsize, 100);

MAT.resize(matsize, 0);
SOL.resize(matsize, 0);

MatComp.resize(MatCompSize, 0);
```

```
    }
};

int main (void){

    vector<double> z;
    double error = 100;

    ContVol cv(Nx, Ny);

    z = multi (0 , cv.t);

    cv.Coordenadas();
    cv.Velocidad();
    cv.CalcularFD();
    cv.Coeficientes();
    cv.Montar();

    cv.RecalcularB();
    cv.Solver();

    while(error >= MaxError){
            cv.t0 = cv.t;

            cv.RecalcularB();
            cv.Solver();

            error = maxabs( restarv(cv.t , cv.t0));
```

```
            cout<<"Error = "<<error<<endl;
    }
    cout<<"ok"<<endl;

    //mkdir("C:/Users/Miguel/Documents/ProyectoFinal/Convection_Diffusion/FILE");

    cv.Datos();
    cout<<"Datos ok"<<endl;

    //write_vtu_2D (0 , cv.nx , cv.ny , cv.x , cv.y , z , cv.u , cv.v , z , cv.t);
}
```

3. Lid Driven Cavity

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <vector>
#include <fstream>
#include <direct.h>
#include <funciones_DC.h>
using namespace std;

// Geometría

int const NX = 35;
int const NY = NX;
double const LX = 1;
double const LY = LX;
int const ESTRUCTURADA=0;

// Fluido

double const U0 = 0;
double const V0 = 0;
double const P0 = 0;
double const Umov = 1;
double const Re = 400;

// Datos numéricos
```

```
double const Cconv = 0.05;

double const Cvisc = 0.2;

double const MaxErrorPerm = 1e-5;

double MaxMasaError;

int PERIODICIDAD = 500;


// Tiempo


double TimeStep;


//Volumen de Control


class vol_cont{

    int k, k1, k2; // cada uno se corresponde respectivamente con el puntero de presión, velocidad v y velocidad u

    public:


        int nx, ny, nx1, ny1, nx2, ny2;

        vector<double> masa;

        vector<double> Xp, Yp, Xv, Yv, Xu, Yu;

        vector<double> p0;

        vector<double> u, u0, v, v0, up, vp;

        vector<double> ap, ae, aw, an, as;

        vector<double> aap, aae, aaw, aan, aas;

        vector<double> MAT, DIV;

        vector<double> U, V, P;

        vector<double> Rx0, Rx00, Ry0, Ry00;
```

```
vector<double> BetaTop, BetaBot, BetaLeft, BetaRight;

vector<double> Pax, Pay, Uax, Uay, Vax, Vay;

vector<double> Pdpe, Pdpw, Pdpn, Pdps, Udpe, Udpw, Udpn, Udps, Vdpe, Vdpw, Vdpn, Vdps;


//Coordenadas


void coordenadas(){
    int i,j;
    vector<double> h;
    vector<double> aux;
    double aux1, denom;


    if(ESTRUCTURADA==0){
                for(i=1; i<=nx-1; i++){
                    if(i<=nx*0.5){
                            h.push_back(double(i));
                    }
                    if(i>nx*0.5){
                            h.push_back(double(nx-i));
                    }
                }
                h.push_back(0);
                Xp[0]=0;
                denom=sum_el(h);


                for(i=0; i<nx-1; i++){
                    aux=selc(h.size(), 1, h, 0, 0, i);
```

```
        aux1=sum_el(aux);
        Xp[i+1]=aux1/denom;
}
aux.clear();
h.clear();

aux=selc(Xp.size(), 1, Xp, 0, 0, nx-1);
for(i=1; i<ny; i++){
        cambiarf(ny, nx, Xp, i, 0, nx-1, aux);
}
Xp=multi(LX, Xp);

for(i=1; i<=ny-1;i++){
        if(i<=ny*0.5){
                h.push_back(double(i));
        }
        if(i>ny*0.5){
                h.push_back(double(ny-i));
        }
}
h.push_back(0);
Yp[0]=0;
denom=sum_el(h);
for(i=0; i<ny-1; i++){
        aux=selc(h.size(), 1, h, 0, 0, i);
        aux1=sum_el(aux);

        Yp[i+1]=aux1/denom;
```

```
		}
		aux.clear();
		h.clear();
		aux=selc(Yp.size(), 1, Yp, 0, 0, ny-1);
		for(i=0; i<nx; i++){
			cambiarc(ny, nx, Yp, i, 0, ny-1, aux);
		}
		Yp=multi(LY, Yp);
}

double ax = LX / (nx - 2);
double ay = LY / (ny - 2);

if(ESTRUCTURADA==1){
		for(j=0; j<ny; j++){
			for(i=0; i<nx; i++){
				k = nx * j + i;

				Xp[k] = ax * 0.5 + ax * (i-1);
				Yp[k] = ay * 0.5 + ay * (j-1);

				if(i == 0){
				Xp[k] = 0;
				}
				if(j == 0){
				Yp[k] = 0;
				}
				if(i == nx-1){
```

```
                    Xp[k] = 1;
                    }
                    if(j == ny-1){
                    Yp[k] = 1;
                    }
                }
            }
}

// Malla de V. Coordenadas

double yn, ys;

for(j=0; j<=ny-2; j++){
    for(i=0; i<=nx-1; i++){
        k = nx * j + i;
        k1 = nx1 * j + i;

        Xv[k1] = Xp[k];
    }
}

for(j=1; j<=ny-3; j++){
    for(i=0; i<=nx-1; i++){
        k = nx * j + i;
        k1 = nx1 * j + i;

        yn = Yp[k+nx];
```

```
            ys = Yp[k];

            Yv[k1] = 0.5 * (yn + ys);
        }
}

j = ny - 2;

for(i=0; i<=nx-1; i++){
        k1 = nx1 * j +i;

        Yv[k1] = LY;
}

//Malla de U. coordenadas

double xe, xw;

for(j=0; j<=ny-1; j++){
        for(i=0; i<=nx-2; i++){
            k = nx * j + i;
            k2 = nx2 *j + i;

            Yu[k2] = Yp[k];
        }
}

for(j=0; j<=ny-1; j++){
```

```
            for(i=1; i<=nx-3; i++){
                k = nx *j +i;
                k2 = nx2 * j + i;

                xe = Xp[k+1];
                xw = Xp[k];

                Xu[k2] = 0.5 * (xe + xw);
            }
        }

        i = nx - 2;

        for(j=0; j<=ny-1; j++){
            k2 = nx2 * j + i;

            Xu[k2] = LX;
        }

}

// Valores Iniciales

void Val_Ini(){
    int i, j;
    double aux1, aux2, aux3;

    j = ny - 1;
```

```
for(i=1; i<nx2-1; i++){
        k = nx2 * j + i;

        u0[k] = Umov;
}

//Top

j = ny - 2;

for(i=1; i<=nx-2; i++){
        k = nx * j + i;
        BetaTop[k] = 1;
}

//Bottom

j = 1;

for(i=1; i<=nx-2; i++){
        k = nx* j + i;
        BetaBot[k] = 1;
}

//Right

i = nx - 2;
```

```
for(j=1; j<=ny-2; j++){
        k = nx * j + i;
        BetaRight[k] = 1;
}

//Left

i = 1;

for (j=1; j<=ny-2; j++){
   k = nx * j + i;
   BetaLeft[k] = 1;
}

//Cálculo distancias
//P

double Br, Bl, Bt, Bb;
double XE, XW, YN, YS, XP, YP;

for(j=1; j<=ny-2; j++){
        for(i=1; i<=nx-2; i++){
                k = nx * j + i;

                XP = Xp[k];
                XE = Xp[k+1];
                XW = Xp[k-1];
```

```
            YP = Yp[k];
            YN = Yp[k+nx];
            YS = Yp[k-nx];

            Br = BetaRight[k];
            Bl = BetaLeft[k];
            Bt = BetaTop[k];
            Bb = BetaBot[k];

            Pax[k] = 0.5 * ( (1+Br) * XE - (1+Bl) * XW +(Bl-Br) * XP);
            Pay[k] = 0.5 * ( (1+Bt) * YN - (1+Bb) * YS +(Bb-Bt) * YP);

            Pdpe[k] = XE - XP;
            Pdpw[k] = XP - XW;
            Pdpn[k] = YN - YP;
            Pdps[k] = YP - YS;

        }
}

double XEp, XPp;

//U

for(j=1; j<=ny-2; j++){
        for(i=1; i<=nx-3; i++){
            k = nx * j + i;
            k2 = nx2 * j + i;
```

```
            XP = Xu[k2];
            XE = Xu[k2+1];
            XW = Xu[k2-1];
            YP = Yu[k2];
            YN = Yu[k2+nx2];
            YS = Yu[k2-nx2];

            Br = BetaRight[k];
            Bl = BetaLeft[k];
            Bt = BetaTop[k];
            Bb = BetaBot[k];

            XEp = Xp[k+1];
            XPp = Xp[k];

            Uax[k2] = XEp - XPp;
            Uay[k2] = 0.5 * ( (1+Bt) * YN - (1+Bb) * YS + (Bb-Bt) * YP);

            Udpe[k2] = XE - XP;
            Udpw[k2] = XP - XW;
            Udpn[k2] = YN - YP;
            Udps[k2] = YP - YS;
        }
}

double YNp, YPp;
```

```
//V


for(j=1; j<=ny-3; j++){
        for(i=1; i<=nx-2; i++){
                k = nx * j + i;
                k1 = nx1 * j + i;


                XP = Xv[k1];
                XE = Xv[k1+1];
                XW = Xv[k1-1];
                YP = Yv[k1];
                YN = Yv[k1+nx1];
                YS = Yv[k1-nx1];


                Br = BetaRight[k];
                Bl = BetaLeft[k];
                Bt = BetaTop[k];
                Bb = BetaBot[k];


                YNp = Yp[k+nx];
                YPp = Yp[k];


                Vax[k1] = 0.5 * ( (1+Br) * XE - (1+Bl) * XW + (Bl-Br) * XP);
                Vay[k1] = YNp - YPp;


                Vdpe[k1] = XE - XP;
                Vdpw[k1] = XP - XW;
                Vdpn[k1] = YN - YP;
```

```
                Vdps[k1] = YP - YS;
            }
        }

        //Time Step

        aux1 = minval0 (Pax);
        aux2 = minval0 (Pay);
        aux3 = fmax( aux1, aux2 );

        TimeStep = Cconv * aux3;

}

//Cálculo Rx y Ry

void CalcularR (int start){
    int i,j;
    double AX, AY;
    double Br, Bl, Bt, Bb;
    double dpe, dpw, dpn, dps;
    double UP, UE, UW, UN, US, UNW;
    double VP, VE, VW, VN, VS, VSE;
    double ue, uw, un, us;
    double ve, vw, vn, vs;
    double De, Dw, Dn, Ds;
    double Fe, Fw, Fn, Fs;
    double dxe, dxw, dyn, dys;
```

```
//Rx

for(j=1; j<=ny-2; j++){
        for(i=1; i<=nx-3; i++){
                k = nx * j + i;
                k1 = nx1 * j + i;
                k2 = nx2 * j + i;

                AX = Uax[k2];
                AY = Uay[k2];

                Bt = BetaTop[k];
                Bb = BetaBot[k];

                dpe = Udpe[k2];
                dpw = Udpw[k2];
                dpn = Udpn[k2];
                dps = Udps[k2];

                UP = u0[k2];
                UE = u0[k2+1];
                UW = u0[k2-1];
                UN = u0[k2+nx2];
                US = u0[k2-nx2];

                VP = v0[k1];
                VE = v0[k1+1];
```

```
VS = v0[k1-nx1];

VSE = v0[k1-nx1+1];


ue = 0.5 * (UP + UE);

uw = 0.5 * (UP + UW);

un = 0.5 * ( (1-Bt) * UP + (1+Bt) * UN );

us = 0.5 * ( (1-Bb) * UP + (1+Bb) * US );

vn = 0.5 * (VP + VE);

vs = 0.5 * (VS + VSE);


Fe = ue * AY;

Fw = uw * AY;

Fn = vn * AX;

Fs = vs * AX;


De = AY / (Re * dpe);

Dw = AY / (Re * dpw);

Dn = AX / (Re * dpn);

Ds = AX / (Re * dps);


dxe = De * (UE - UP);

dxw = Dw * (UP - UW);

dyn = Dn * (UN - UP);

dys = Ds * (UP - US);


Rx0[k2] = (dxe - dxw) + (dyn - dys) -( (Fe*ue -Fw*uw) + (Fn*un -
Fs*us) );

Rx0[k2] = Rx0[k2] / (AX * AY);
```

```
        }
}

//Ry

for(j=1; j<=ny-3; j++){
        for(i=1; i<=nx-2; i++){
                k = nx * j + i;
                k1 = nx1 * j + i;
                k2 = nx2 * j + i;

                AX = Vax[k1];
                AY = Vay[k1];

                Br = BetaRight[k];
                Bl = BetaLeft[k];

                dpe = Vdpe[k1];
                dpw = Vdpw[k1];
                dpn = Vdpn[k1];
                dps = Vdps[k1];

                VP = v0[k1];
                VE = v0[k1+1];
                VW = v0[k1-1];
                VN = v0[k1+nx1];
                VS = v0[k1-nx1];
```

```
UP = u0[k2];
UW = u0[k2-1];
UN = u0[k2+nx2];
UNW = u0[k2+nx2-1];

vn = 0.5 * (VP + VN);
vs = 0.5 * (VP + VS);
ve = 0.5 * ( (1-Br) * VP + (1+Br) * VE );
vw = 0.5 * ( (1-Bl) * VP + (1+Bl) * VW );
ue = 0.5 * (UP + UN);
uw = 0.5 * (UNW +UW);

Fe = ue * AY;
Fw = uw * AY;
Fn = vn * AX;
Fs = vs * AX;

De = AY / (Re * dpe);
Dw = AY / (Re * dpw);
Dn = AX / (Re * dpn);
Ds = AX / (Re * dps);

dxe = De * (VE - VP);
dxw = Dw * (VP - VW);
dyn = Dn * (VN - VP);
dys = Ds * (VP - VS);
```

```
                    Ry0[k1] = (dxe - dxw) + (dyn - dys) - ( (Fe*ve - Fw*vw) + (Fn*vn -
Fs*vs) );
                    Ry0[k1] = Ry0[k1] / (AX * AY);
                }
            }

            if(start == 1){
                Rx00 = Rx0;
                Ry00 = Ry0;
            }

        }

        //Cálculo de up y vp

        void CalcularUpVp(){
            int i, j;

            //up

            for(j=1; j<=ny-2; j++){
                for(i=1; i<=nx-3; i++){
                    k2 = nx2 * j + i;

                    up[k2] = u0[k2] + 0.5 * TimeStep * (3 * Rx0[k2] - Rx00[k2]);
                }
            }
```

```
    j = ny - 1;

    for(i=1; i<=nx-3; i++){
        k2 = nx2 * j + i;

        up[k2] = Umov;
    }

    //vp

    for(j=1; j<=ny-3; j++){
        for(i=1; i<=nx-2; i++){
            k1 = nx1 * j + i;

            vp[k1] = v0[k1] + 0.5 * TimeStep * (3 * Ry0[k1] - Ry00[k1]);
        }
    }

}

//Divergencia de Vp

void CalcularDiv(){
    int i, j, kk;
    double UP, UW, VP, VS;
    double AX, AY;

    for(j=0; j<=NY-1; j++){
```

```
            for(i=0; i<=NX-1; i++){
                kk = NX * j + i;
                k = nx * j + i;
                k1 = nx1 * j + i;
                k2 = nx2 * j + i;

                AX = Pax[k+1+nx];
                AY = Pay[k+1+nx];

                UP = up[k2+1+nx2];
                UW = up[k2+nx2];

                VP = vp[k1+1+nx1];
                VS = vp[k1+1];

                DIV[kk] = (UP - UW) / AX + (VP - VS) / AY;
            }
        }

}

//Matriz de coeficientes

void Coeficientes(){
    int i, j, kk;
    double dpe, dpw, dpn, dps;
    double AP, AE, AW, AN, AS;
    double Be, Bw, Bn, Bs;
```

```
double AX, AY;


for(j=0; j<=NY-1; j++){
        for(i=0; i<=NX-1; i++){
                kk = NX * j + i;
                k = nx * j + i;


                AX = Pax[k+1+nx];
                AY = Pay[k+1+nx];


                dpe = Pdpe[k+1+nx];
                dpw = Pdpw[k+1+nx];
                dpn = Pdpn[k+1+nx];
                dps = Pdps[k+1+nx];


                Be = BetaRight[k+1+nx];
                Bw = BetaLeft[k+1+nx];
                Bn = BetaTop[k+1+nx];
                Bs = BetaBot[k+1+nx];


                AE = (1 - Be) / (dpe * AX);
                AW = (1 - Bw) / (dpw * AX);
                AN = (1 - Bn) / (dpn * AY);
                AS = (1 - Bs) / (dps * AY);


                AP = - (AE + AW + AN + AS);


                ae[kk] = AE;
```

```
                aw[kk] = AW;
                an[kk] = AN;
                as[kk] = AS;
                ap[kk] = AP;
        }
}

j = 1;
i = int(NX*0.5);
    kk = NX * j + i;
    k = nx * j + i;

    AX = Pax[k+1+nx];
    AY = Pay[k+1+nx];

    dpe = Pdpe[k+1+nx];
    dpw = Pdpw[k+1+nx];
    dpn = Pdpn[k+1+nx];
    dps = Pdps[k+1+nx];

    AE = 1 / (dpe * AX);
    AW = 1 / (dpw * AX);
    AN = 1 / (dpn * AY);
    AS = 1 / (dps * AY);

    AP = - (AE + AW + AN + AS);

    AS = 0;
```

```
        ae[kk] = AE;
        aw[kk] = AW;
        an[kk] = AN;
        as[kk] = AS;
        ap[kk] = AP;

        Ensamble();
}

void Ensamble(){

    cambiarc(as.size(), 5, MAT, 0, 0, as.size() - 1, as);
    cambiarc(aw.size(), 5, MAT, 1, 0, aw.size() - 1, aw);
    cambiarc(ap.size(), 5, MAT, 2, 0, ap.size() - 1, ap);
    cambiarc(ae.size(), 5, MAT, 3, 0, ae.size() - 1, ae);
    cambiarc(an.size(), 5, MAT, 4, 0, an.size() - 1, an);

}

// Solver iteración

void SolverIteracion(){
    int i, j, kk;
    vector<double> pp;

    pp = GaussSeidel(MAT, DIV);
```

```
for(j=0; j<=NY-1; j++){
        for(i=0; i<=NX-1; i++){
                k = nx *j +i;
                kk = NX * j + i;

                p0[k+1+nx] = pp[kk];
        }
}

pp.clear();

i=0;
for(j=1; j<=ny-2; j++){
        k = nx * j + i;
        p0[k] = p0[k+1];
}

i=nx-1;
for(j=1; j<=ny-2; j++){
        k = nx * j + i;
        p0[k] = p0[k-1];
}

j=0;
for(i=0; i<=nx-1; i++){
        k = nx * j +i;
        p0[k] = p0[k+nx];
}
```

```
    j=ny-1;
    for(i=0; i<=nx-1; i++){
            k = nx * j + i;
            p0[k] = p0[k-nx];
    }

    j=0;
    i=int(nx*0.5);
    k = nx * j + i;
    p0[k] = 0;


}

//Calculo nuevas velocidades

void CalculoVelocidades (){
    int i, j;
    double PP, PN, PE;
    double AX, AY;
    double grad;

    //U

    for(j=1; j<=ny-2; j++){
            for(i=1; i<=nx-3; i++){
                    k = nx * j + i;
```

```
            k2 = nx2 * j + i;


            AX = Uax[k2];
            PP = p0[k];
            PE = p0[k+1];

            grad = (PE - PP) / AX;

            u[k2] = up[k2] - grad;
        }
}

j=ny-1;
for(i=1; i<=nx-3; i++){
        k = nx * j +i;
        k2 = nx2 * j + i;

        u[k2] = Umov;
}

//V

for(j=1; j<=ny-3; j++){
        for(i=1; i<=nx-2; i++){
            k = nx * j + i;
            k1 = nx1 * j + i;

            AY = Vay[k1];
```

```
                PP = p0[k];
                PN = p0[k+nx];

                grad = (PN - PP) / AY;

                v[k1] = vp[k1] - grad;
            }
        }

}

//Calculo error

double CalculoMaxError(){
    double MaxErrorU, MaxErrorV, MaxError;

    MaxErrorU = maxabs( restarv (u, u0) );
    MaxErrorV = maxabs( restarv (v, v0) );

    MaxError = fmax (MaxErrorU, MaxErrorV);

    return MaxError;
}

//Nuevo Time Step

void CalcularTimeStep(){
    int i, j;
```

```
    double Maxu, Maxv, MaxVel, Minx, Miny, MinX;
    double a, b;

    Minx = minval0(Pax);
    Miny = minval0(Pay);
    Maxu = maxabs(u);
    Maxv = maxabs(v);
    MinX = fmin( Minx, Miny);
    MaxVel = fmax(Maxu, Maxv);

    TimeStep = Cconv * MinX / MaxVel;
}

//Calculo de velocidades y presiones Medias

void CampoVelocidades(){
    int i, j;
    double UW, UP, VS, VP;

    for(j=1; j<ny-1; j++){
        for(i=1; i<nx-1; i++){
            k = nx * j + i;
            k1 = nx1 * j + i;
            k2 = nx2 * j + i;

            UW = u[k2-1];
            UP = u[k2];
```

```
                VS = v[k1-nx1];
                VP = v[k1];

                U[k] = 0.5 * (UP + UW);
                V[k] = 0.5 * (VP + VS);
            }
        }

        j=ny-1;
        for(i=1; i<nx-1; i++){
            k = nx * j + i;
            U[k] = Umov;
        }
    }

    void CampoPresiones() {

        P = dividir (p0, TimeStep);
    }

    //Comprovación continuidad

    void ComprovarContinuidad(){
        int i,j;
        double UP, UW, VP, VS;
        double AX, AY;
        double Up, Uw, Vp, Vs;
```

```
    for(j=1; j<ny-1; j++){
        for(i=1; i<nx-1; i++){
            k = nx * j + i;
            k1 = nx1 * j + i;
            k2 = nx2 * j + i;

            UP = u[k2];
            UW = u[k2-1];
            VP = v[k1];
            VS = v[k1-nx1];

            AX = Pax[k];
            AY = Pay[k];

            masa[k] = (UP - UW) * AY + (VP - VS) * AX;
        }
    }

    MaxMasaError = maxabs(masa);
}

//Obtención de datos

void GetData(){
    int i,j,k;
    vector<double> datax, datay, datau, datav, datapx, datapy;
    ofstream data1;
```

```
j=int(ny*0.5);

for(i=0; i<nx; i++){
        k = nx * j + i;

        datax.push_back(Xp[k]);
        datav.push_back(V[k]);
        datapx.push_back(P[k]);
}

i=int(nx*0.5);

for(j=0; j<ny; j++){
        k = nx * j + i;

        datay.push_back(Yp[k]);
        datau.push_back(U[k]);
        datapy.push_back(P[k]);
}

if(Re == 100){
     data1.open("Re_100.dat", std::ofstream::out);
}
if(Re == 400){
     data1.open("Re_400.dat", std::ofstream::out);
}
if(Re == 1000){
     data1.open("Re_1000.dat", std::ofstream::out);
```

```
        }
        if(Re == 3200){
            data1.open("Re_3200.dat", std::ofstream::out);
        }
        if(Re == 5000){
            data1.open("Re_5000.dat", std::ofstream::out);
        }
        if(Re == 7500){
            data1.open("Re_7500.dat", std::ofstream::out);
        }
        if(Re == 10000){
            data1.open("Re_10000.dat", std::ofstream::out);
        }

        data1<<"x"<<"  "<<"v"<<"  "<<"px"<<"  "<<"y"<<"  "<<"u"<<"  "<<"py"<<endl;

        for(i=0; i<datax.size(); i++){
            data1<<datax[i]<<"  "<<datav[i]<<"  "<<datapx[i]<<"  "<<datay[i]<<"  "<<datau[i]<<"  "<<datapy[i]<<endl;
        }

        datax.clear();
        datay.clear();
        datau.clear();
        datav.clear();
        datapx.clear();
        datapy.clear();
    }
```

```
vol_cont(int aa, int bb){
        int i, j;
        int Psize, Vsize, Usize, MATsize, SystemSize;

        nx = aa + 2;
        ny = bb + 2;

        nx1 = nx;
        ny1 = ny - 1;

        nx2 = nx - 1;
        ny2 = ny;

        Psize = nx * ny;
        Vsize = nx1 * ny1;
        Usize = nx2 * ny2;

        SystemSize = aa * bb;

        MATsize = 5 * SystemSize;

        masa.resize(Psize, 0);

        Xp.resize(Psize, 0);
        Yp.resize(Psize, 0);
        p0.resize(Psize, P0);
        U.resize(Psize, 0);
```

```
V.resize(Psize, 0);
P.resize(Psize, 0);
BetaTop.resize(Psize, 0);
BetaBot.resize(Psize, 0);
BetaRight.resize(Psize, 0);
BetaLeft.resize(Psize, 0);
Pax.resize(Psize, 0);
Pay.resize(Psize, 0);
Pdpe.resize(Psize, 0);
Pdpw.resize(Psize, 0);
Pdpn.resize(Psize, 0);
Pdps.resize(Psize, 0);

Xv.resize(Vsize, 0);
Yv.resize(Vsize, 0);
v.resize(Vsize, 0);
v0.resize(Vsize, V0);
Ry0.resize(Vsize, 0);
Ry00.resize(Vsize, 0);
vp.resize(Vsize, 0);
Vax.resize(Vsize, 0);
Vay.resize(Vsize, 0);
Vdpe.resize(Vsize, 0);
Vdpw.resize(Vsize, 0);
Vdpn.resize(Vsize, 0);
Vdps.resize(Vsize, 0);

Xu.resize(Usize, 0);
```

```
                Yu.resize(Usize, 0);
                u.resize(Usize, 0);
                u0.resize(Usize, U0);
                Rx0.resize(Usize, 0);
                Rx00.resize(Usize, 0);
                up.resize(Usize, 0);
                Uax.resize(Usize, 0);
                Uay.resize(Usize, 0);
                Udpe.resize(Usize, 0);
                Udpw.resize(Usize, 0);
                Udpn.resize(Usize, 0);
                Udps.resize(Usize, 0);

                MAT.resize(MATsize, 0);

                DIV.resize(SystemSize, 0);

                ap.resize(SystemSize, 1);
                ae.resize(SystemSize, 0);
                aw.resize(SystemSize, 0);
                an.resize(SystemSize, 0);
                as.resize(SystemSize, 0);
        }
};

int main (void){
    ofstream data;
    double error;
```

```
int it = 0;

int counter = 0;


vector<double> z;


vol_cont vc(NX,NY);


vc.coordenadas();

vc.Val_Ini();

vc.CalcularR(1);

vc.CalcularUpVp();

vc.CalcularDiv();

vc.Coeficientes();

vc.SolverIteracion();

vc.CalculoVelocidades();

vc.ComprovarContinuidad();


error = vc.CalculoMaxError();


vc.CalcularTimeStep();


cout<<"Time Step = "<<TimeStep<<" , error = "<<error<<" , Masa =
"<<MaxMasaError<<endl;


z = multi(0, vc.Xp);


mkdir("C:/Users/Miguel/Documents/ProyectoFinal/Driven_Cavity/FILE");
```

```
vc.CampoVelocidades();
vc.CampoPresiones();

it++;

write_vtu_2D (it, vc.nx, vc.ny, vc.Xp, vc.Yp, z, vc.U, vc.V, z, vc.P);

error = 100;
while(error>=MaxErrorPerm && it<=120000){
                counter ++;

                vc.u0 = vc.u;
                vc.v0 = vc.v;
                vc.Rx00 = vc.Rx0;
                vc.Ry00 = vc.Ry0;

                vc.CalcularR(2);
                vc.CalcularUpVp();
                vc.CalcularDiv();
                vc.SolverIteracion();
                vc.CalculoVelocidades();
                vc.ComprovarContinuidad();

                error = vc.CalculoMaxError();

                cout<<"It = "<<it<<" , Time Step = "<<TimeStep<<" , error =
"<<error<<" , Masa = "<<MaxMasaError<<endl;
```

```
                    it++;

                    if(counter == PERIODICIDAD){
                        vc.CampoVelocidades();
                        vc.CampoPresiones();

                        write_vtu_2D (it, vc.nx, vc.ny, vc.Xp, vc.Yp, z, vc.U, vc.V, z,
vc.P);

                        counter = 0;
                        vc.GetData();
                    }
    }

    vc.CampoVelocidades();
    vc.CampoPresiones();
    write_vtu_2D (it, vc.nx, vc.ny, vc.Xp, vc.Yp, z, vc.U, vc.V, z, vc.P);
    vc.GetData();

}
```

4. Burguers Equation in Fourier Space

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <vector>
#include <fstream>
#include <direct.h>
#include <funciones_T.h>
using namespace std;

int const Nx = 20;
double const Re = 40;
double const TimeStep = 1e-3;
int const Nxsize = Nx+1;
int const Xsize = 2*Nx+1;
double const MaxErrorPermitido = 1e-8;

vector<comp> u(Nxsize, 0), u0(Nxsize, 0);
vector<comp> H0(Nxsize, 0), H00(Nxsize, 0);
vector<double> E(Nxsize, 0);
double error;
double Time;

comp Conveccion (int K){
    int i;
    vector<int> P, Q;
    double RE, Im;
```

```
int Pp, Qq;
comp Pc, Qc;
comp sol(0, 0);
comp aux, aux1;

if(K < 2){
    cout<<"Error, no se calculan K's menores de 2"<<endl;
}

if(K >= 2){
    for(i=-Nx+K; i<=Nx; i++){
            P.push_back(i);
    }
    for(i=Nx; i>=-Nx+K; i--){
        Q.push_back(i);
    }

    for(i=0; i<P.size(); i++){
        Pp = P[i];
        Qq = Q[i];

        if(Pp >= 0){
            Pc = u0[Pp];
        }
        if(Pp < 0){
            Pc = u0[abs(Pp)];
            Pc = conj(Pc);
        }
```

```
            if(Qq >= 0){
                Qc = u0[Qq];
            }
            if(Qq < 0){
                Qc = u0[abs(Qq)];
                Qc = conj(Qc);
            }

            aux = resize_complejos (double(Qq), 0);
            Qc = multi_complejos (Qc, aux);
            aux1 = multi_complejos (Pc, Qc);
            sol = suma_complejos(sol, aux1);
        }

        aux = resize_complejos (0, 1);
        sol = multi_complejos (sol, aux);

    }

    return sol;
}

comp CalcularH0 (int K){
    comp Uk, Ck, Const;
    double ConstRe;
    comp sol;
```

```
    ConstRe = K * K / Re;
    Const = resize_complejos (ConstRe, 0);

    Uk = u0[K];
    Ck = Conveccion(K);

    sol = multi_complejos (Const, Uk);
    sol = suma_complejos (sol, Ck);

    return sol;
}

comp CalcularU (int K){
    comp U0;
    comp h0, h00;
    comp sol, auxT, aux, auxS;
    double TimeConst;

    U0 = u0[K];
    h00 = H00[K];
    h0 = CalcularH0(K);
    H0[K] = h0;

    TimeConst = TimeStep * 0.5;

    auxT = resize_complejos (TimeConst, 0);
    aux = resize_complejos (3, 0);
    h0 = multi_complejos (h0, aux);
```

```
    auxS = restar_complejos ( h0, h00);
    auxS = multi_complejos (auxS, auxT);

    sol = restar_complejos (U0, auxS);

    return sol;
}

void ValoresIniciales (){
    int i;
    double k;
    k=0;

    for(i=0; i<u0.size(); i++){
        if(k != 0){
            u0[i] = resize_complejos (1/k, 0);
        }
        if(k == 0){
            u0[i] = resize_complejos (k, 0);
        }

        k++;
    }

    for(i=2; i<H00.size(); i++){
        H00[i] = CalcularH0(i);

    }
```

```
}

double Comp_Error(){
	int i;
	double ReError, ImError, MaxError;
	double RE, Im, RE0, Im0;
	vector<double> ErrorRe, ErrorIm;

	for(i=0; i<u.size(); i++){
		RE = real (u[i]);
		Im = imag (u[i]);
		RE0 = real (u0[i]);
		Im0 = imag (u0[i]);
		ErrorRe.push_back (RE-RE0);
		ErrorIm.push_back (Im-Im0);
	}

	ReError = maxabs (ErrorRe);
	ImError = maxabs (ErrorIm);
	MaxError = fmax (ReError, ImError);

	return MaxError;
}

vector<double> Calcular_Energia (vector<comp> U){
		int i;
		comp Un, Ucon;
		double En;
```

```
        vector<double> sol(Nxsize, 0);

        for(i=0; i<U.size(); i++){
            Un = U[i];
            Ucon = conj (Un);
            En = abs (multi_complejos (Un, Ucon));
            sol[i] = En;
        }

        return sol;
}

void GetData(){
    int i, j, k;
    double RE, Im;
    vector<double> dataK, dataUre, dataUim, dataE;
    ofstream data1, data2;

    for(i=0; i<Nxsize; i++){
        RE = real (u[i]);
        Im = imag (u[i]);

        dataK.push_back(i);
        dataUre.push_back(RE);
        dataUim.push_back(Im);
    }

    data1.open("Coeficientes_U.dat", std::ofstream::out);
```

```
    data1<<"k"<<"  "<<"U real"<<"  "<<"U imaginaria"<<endl;

    for(i=0; i<dataK.size(); i++){
        data1<<dataK[i]<<"  "<<dataUre[i]<<"  "<<dataUim[i]<<endl;
    }

    dataK.clear();
    dataUre.clear();
    dataUim.clear();
    data1.close();

    for(i=0; i<Nxsize; i++){
        dataK.push_back(i);
        dataE.push_back(E[i]);
    }

    data2.open("Energia.dat", std::ofstream::out);
    data2<<"k"<<"  "<<"Energia"<<endl;

    for(i=0; i<dataE.size(); i++){
        data2<<dataK[i]<<"  "<<dataE[i]<<endl;
    }

    dataK.clear();
    dataE.clear();
    data2.close();

}
```

```
int main (void){

    comp aux;
    int k;
    int it;

    ValoresIniciales();

    u[0] = resize_complejos (0, 0);
    u[1] = resize_complejos (1, 0);

    for(k=2; k<u.size(); k++){
            u[k] = CalcularU(k);
    }

    error = Comp_Error();

    it = 1;
    Time = TimeStep * double(it);
    cout<<"It= "<<" , Time= "<<Time<<" , MaxError= "<<error<<endl;

    while (error >= MaxErrorPermitido){
        it++;

        u0 = u;
        H00 = H0;
```

```
        u[0] = resize_complejos (0, 0);
        u[1] = resize_complejos (1, 0);

        for(k=2; k<u.size(); k++){
                u[k] = CalcularU(k);
        }

        error = Comp_Error();
        Time = TimeStep * double(it);
        cout<<"It= "<<it<<" , Time= "<<Time<<" , MaxError= "<<error<<endl;
    }

    E = Calcular_Energia(u);
    GetData();
}
```

5. Differentially Heated Cavity

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <vector>
#include <fstream>
#include <direct.h>
#include <time.h>
#include <funciones_DHC.h>
//#include <functions.h>
using namespace std;

// Geometría

int const NX = 15;
int const NY = NX;
double const LX = 1;
double const LY = LX;
int const ESTRUCTURADA=0;
// Fluido

double const U0 = 0;
double const V0 = 0;
double const P0 = 0;
double const T0 = 0;
double const Ra = 1e6; //Solo cambiar este valor
double const Pr=0.71;
```

```
double const Re=1/Pr;


// Datos numéricos


double const Cconv = 0.35;

double const Cvisc = 0.2;

double const Cdif=Cvisc;

double const MaxErrorPerm = 1e-5;

double MaxMasaError;

int PERIODICIDAD = 1;


// Tiempo


double TimeStep;

double Time;


//Volumen de Control


class vol_cont{

    int k, k1, k2; // cada uno se corresponde respectivamente con el puntero de presión, velocidad v y velocidad u

    public:


        int nx, ny, nx1, ny1, nx2, ny2;

        vector<double> masa;

        vector<double> Xp, Yp, Xv, Yv, Xu, Yu;

        vector<double> p0;

        vector<double> u, u0, v, v0, up, vp;
```

```
vector<double> ap, ae, aw, an, as;

vector<double> MAT, DIV;

vector<double> U, V, P;

vector<double> Rx0, Rx00, Ry0, Ry00;

vector<double> BetaTop, BetaBot, BetaLeft, BetaRight;

vector<double> TetaTop, TetaBot, TetaLeft, TetaRight;

vector<double> Pax, Pay, Uax, Uay, Vax, Vay;

vector<double> Pdpe, Pdpw, Pdpn, Pdps, Udpe, Udpw, Udpn, Udps, Vdpe, Vdpw, Vdpn, Vdps;

vector<double> t, t0;

vector<double> h0, h00;

vector<double> MatU, MatV, MatT;

vector<double> Hu0, Hu00, Hv0, Hv00, Ht0, Ht00;

vector<double> Uap, Uae, Uaw, Uan, Uas;

vector<double> Vap, Vae, Vaw, Van, Vas;

vector<double> Tap, Tae, Taw, Tan, Tas;

vector<double> Bu, Bv, Bt;


//Coordenadas


void coordenadas(){

    int i,j;

    vector<double> h;

    vector<double> aux;

    double aux1, denom;


    if(ESTRUCTURADA==0){

                for(i=1; i<=nx-1; i++){
```

```
        if(i<=nx*0.5){
                h.push_back(double(i));
        }
        if(i>nx*0.5){
                h.push_back(double(nx-i));
        }
}

h.push_back(0);
Xp[0]=0;
denom=sum_el(h);

for(i=0; i<nx-1; i++){
        aux=selc(h.size(), 1, h, 0, 0, i);
        aux1=sum_el(aux);

        Xp[i+1]=aux1/denom;
}

aux.clear();
h.clear();

aux=selc(Xp.size(), 1, Xp, 0, 0, nx-1);

for(i=1; i<ny;i++){
        cambiarf(ny, nx, Xp, i, 0, nx-1, aux);
}
```

```
Xp=multi(LX, Xp);

for(i=1; i<=ny-1; i++){
        if(i<=ny*0.5){
                h.push_back(double(i));
        }
        if(i>ny*0.5){
                h.push_back(double(ny-i));
        }
}

h.push_back(0);
Yp[0]=0;
denom=sum_el(h);

for(i=0; i<ny-1; i++){
        aux=selc(h.size(), 1, h, 0, 0, i);
        aux1=sum_el(aux);
        Yp[i+1]=aux1/denom;
}

aux.clear();
h.clear();

aux=selc(Yp.size(), 1, Yp, 0, 0, ny-1);

for(i=0; i<nx; i++){
        cambiarc(ny,nx, Yp, i, 0, ny-1, aux);
```

```
                }


                Yp=multi(LY, Yp);
}

double ax = LX / (nx - 2);
double ay = LY / (ny - 2);

if(ESTRUCTURADA == 1){
for(j=0; j<ny; j++){
        for(i=0; i<nx; i++){
                k = nx * j + i;

                Xp[k] = ax * 0.5 + ax * (i-1);
                Yp[k] = ay * 0.5 + ay * (j-1);

                if(i == 0){
                    Xp[k] = 0;
                }
                if(j == 0){
                    Yp[k] = 0;
                }
                if(i == nx-1){
                    Xp[k] = 1;
                }
                if(j == ny-1){
                    Yp[k] = 1;
                }
```

```
        }
}
}

// Malla de V. Coordenadas

double yn, ys;

for(j=0; j<=ny-2; j++){
        for(i=0; i<=nx-1; i++){
                k = nx * j + i;
                k1 = nx1 * j + i;

                Xv[k1] = Xp[k];
        }
}

for(j=1; j<=ny-3; j++){
        for(i=0; i<=nx-1; i++){
                k = nx * j + i;
                k1 = nx1 * j + i;

                yn = Yp[k+nx];
                ys = Yp[k];

                Yv[k1] = 0.5 * (yn + ys);
        }
}
```

```
j = ny - 2;

for(i=0; i<=nx-1; i++){
        k1 = nx1 * j +i;

        Yv[k1] = LY;
}

//Malla de U. coordenadas

double xe, xw;

for(j=0; j<=ny-1; j++){
        for(i=0; i<=nx-2; i++){
                k = nx * j + i;
                k2 = nx2 *j + i;

                Yu[k2] = Yp[k];
        }
}

for(j=0; j<=ny-1; j++){
        for(i=1; i<=nx-3; i++){
                k = nx *j +i;
                k2 = nx2 * j + i;

                xe = Xp[k+1];
```

```
                xw = Xp[k];

                Xu[k2] = 0.5 * (xe + xw);
            }
        }

        i = nx - 2;

        for(j=0; j<=ny-1; j++){
            k2 = nx2 * j + i;

            Xu[k2] = LX;
        }

    }

    // Valores Iniciales

    void Val_Ini(){
        int i, j;
        double aux1, aux2, aux3;

        //Top

        j = ny - 2;

        for(i=1; i<=nx-2; i++){
            k = nx * j + i;
```

```
        BetaTop[k] = 1;
        TetaTop[k] = 1;
}

//Bottom

j = 1;

for(i=1; i<=nx-2; i++){
        k = nx* j + i;
        BetaBot[k] = 1;
        TetaBot[k] = 1;
}

//Right

i = nx - 2;

for(j=1; j<=ny-2; j++){
        k = nx * j + i;
        BetaRight[k] = 1;
}

//Left

i = 1;

for (j=1; j<=ny-2; j++){
```

```
    k = nx * j + i;
    BetaLeft[k] = 1;
}

//Cálculo distancias
//P

double Br, Bl, Bt, Bb;
double XE, XW, YN, YS, XP, YP;

for(j=1; j<=ny-2; j++){
        for(i=1; i<=nx-2; i++){
                k = nx * j + i;

                XP = Xp[k];
                XE = Xp[k+1];
                XW = Xp[k-1];
                YP = Yp[k];
                YN = Yp[k+nx];
                YS = Yp[k-nx];

                Br = BetaRight[k];
                Bl = BetaLeft[k];
                Bt = BetaTop[k];
                Bb = BetaBot[k];

                Pax[k] = 0.5 * ( (1+Br) * XE - (1+Bl) * XW +(Bl-Br) * XP);
                Pay[k] = 0.5 * ( (1+Bt) * YN - (1+Bb) * YS +(Bb-Bt) * YP);
```

```
                Pdpe[k] = XE - XP;
                Pdpw[k] = XP - XW;
                Pdpn[k] = YN - YP;
                Pdps[k] = YP - YS;

        }
}

double XEp, XPp;

//U

for(j=1; j<=ny-2; j++){
        for(i=1; i<=nx-3; i++){
                k = nx * j + i;
                k2 = nx2 * j + i;

                XP = Xu[k2];
                XE = Xu[k2+1];
                XW = Xu[k2-1];
                YP = Yu[k2];
                YN = Yu[k2+nx2];
                YS = Yu[k2-nx2];

                Br = BetaRight[k];
                Bl = BetaLeft[k];
                Bt = BetaTop[k];
```

```
                Bb = BetaBot[k];


                XEp = Xp[k+1];
                XPp = Xp[k];


                Uax[k2] = XEp - XPp;
                Uay[k2] = 0.5 * ( (1+Bt) * YN - (1+Bb) * YS + (Bb-Bt) * YP);


                Udpe[k2] = XE - XP;
                Udpw[k2] = XP - XW;
                Udpn[k2] = YN - YP;
                Udps[k2] = YP - YS;
        }
}

double YNp, YPp;

//V

for(j=1; j<=ny-3; j++){
        for(i=1; i<=nx-2; i++){
                k = nx * j + i;
                k1 = nx1 * j + i;


                XP = Xv[k1];
                XE = Xv[k1+1];
                XW = Xv[k1-1];
                YP = Yv[k1];
```

```
                YN = Yv[k1+nx1];
                YS = Yv[k1-nx1];

                Br = BetaRight[k];
                Bl = BetaLeft[k];
                Bt = BetaTop[k];
                Bb = BetaBot[k];

                YNp = Yp[k+nx];
                YPp = Yp[k];

                Vax[k1] = 0.5 * ( (1+Br) * XE - (1+Bl) * XW + (Bl-Br) * XP);
                Vay[k1] = YNp - YPp;

                Vdpe[k1] = XE - XP;
                Vdpw[k1] = XP - XW;
                Vdpn[k1] = YN - YP;
                Vdps[k1] = YP - YS;
        }
}

//Temperatura

i=nx-1;
for(j=0; j<=ny-1; j++){
        k=nx*j+i;
        t0[k]=-0.5;
}
```

```
    i=0;
    for(j=0; j<=ny-1; j++){
        k=nx*j+i;
        t0[k]=0.5;
    }

    //Time Step

    aux1 = minval0 (Pax);
    aux2 = minval0 (Pay);
    aux3 = fmax( aux1, aux2 );

    TimeStep = Cconv * aux3;
    aux1=Cdif*aux3*aux3;

    TimeStep=fmin(TimeStep, aux1);
    aux1=Cvisc*Re*aux3*aux3;

    TimeStep=fmin(TimeStep,aux1);

}

//Cálculo T
void CalcularT(int start){
    int i, j;

    //Cálculo de H
```

```
double dxe, dxw, dyn, dys;
double UU, VV, UP, UW, VP, VS;
double dx, dy;
double TP, TE, TW, TN, TS;
double DPE, DPW, DPN, DPS;
double Te, Tw, Tn, Ts;
double AX, AY;
double cntx, cnty;
double AP, AE, AW, AN, AS;
double H0, H00;

for(j=1; j<=ny-2; j++){
        for(i=1; i<=nx-2; i++){
                k=nx*j+i;
                k1=nx1*j+i;
                k2=nx2*j+i;

                AX=Pax[k];
                AY=Pay[k];

                cntx= -0.5*TimeStep/AX;
                cnty= -0.5*TimeStep/AY;

                TP=t0[k];
                TE=t0[k+1];
                TW=t0[k-1];
                TN=t0[k+nx];
```

```
TS=t0[k-nx];

UW=u0[k2-1];
UP=u0[k2];
VS=v0[k1-nx1];
VP=v0[k1];

UU=0.5*(UP+UW);
VV=0.5*(VP+VS);

DPE=Pdpe[k];
DPW=Pdpw[k];
DPN=Pdpn[k];
DPS=Pdps[k];

Te=TetaRight[k];
Tw=TetaLeft[k];
Tn=TetaTop[k];
Ts=TetaBot[k];

dxe=(TE-TP)*(1-Te)/DPE;
dxw=(TP-TW)*(1-Tw)/DPW;
dyn=(TN-TP)*(1-Tn)/DPN;
dys=(TP-TS)*(1-Ts)/DPS;

dx=(TE-TW)/(DPE+DPW);
dy=(TN-TS)/(DPN+DPS);
```

```
                    Ht0[k]=-(UU*dx+VV*dy);
                    H0=Ht0[k];

                    if(start==1){
                        Ht00[k]=Ht0[k];
                    }

                    H00=Ht00[k];

                    Bt[k]=TP+0.5*TimeStep*((dxe-dxw)/AX + (dyn-dys)/AY + (3*H0-
H00));

                    //Coeficientes

                    AE=cntx*(1-Te)/DPE;
                    AW=cntx*(1-Tw)/DPW;
                    AN=cnty*(1-Tn)/DPN;
                    AS=cnty*(1-Ts)/DPS;

                    AP=1-(AE+AW+AN+AS);

                    Tap[k]=AP;
                    Tae[k]=AE;
                    Taw[k]=AW;
                    Tan[k]=AN;
                    Tas[k]=AS;
                }
            }
```

```
j=0;
for(i=1; i<=nx-2; i++){
        k=nx*j+i;
        Tap[k]=1;
        Tae[k]=0;
        Taw[k]=0;
        Tan[k]=-1;
        Tas[k]=0;
}

j=ny-1;
for(i=1; i<=nx-2; i++){
        k=nx*j+i;
        Tap[k]=1;
        Tae[k]=0;
        Taw[k]=0;
        Tan[k]=0;
        Tas[k]=-1;
}

i=0;
for(j=0; j<=ny-1; j++){
        k=nx*j+i;
        Tap[k]=1;
        Tae[k]=0;
        Taw[k]=0;
        Tan[k]=0;
```

```
            Tas[k]=0;
            Bt[k]=0.5;
        }

        i=nx-1;
        for(j=0; j<=ny-1; j++){
            k=nx*j+i;
            Tap[k]=1;
            Tae[k]=0;
            Taw[k]=0;
            Tan[k]=0;
            Tas[k]=0;
            Bt[k]=-0.5;
        }

        cambiarc(Tas.size(), 5, MatT, 0, 0, Tas.size()-1, Tas);
        cambiarc(Taw.size(), 5, MatT, 1, 0, Taw.size()-1, Taw);
        cambiarc(Tap.size(), 5, MatT, 2, 0, Tap.size()-1, Tap);
        cambiarc(Tae.size(), 5, MatT, 3, 0, Tae.size()-1, Tae);
        cambiarc(Tan.size(), 5, MatT, 4, 0, Tan.size()-1, Tan);

        t=GaussSeidel(MatT,Bt);
    }

    //Cálculo de up y vp

    void CalcularUpVp(int start){
        int i, j;
```

```
double AX, AY;
double Br, Bl, Bt, Bb;
double dpe, dpw, dpn, dps;
double UP, UE, UW, UN, US, UNW;
double VP, VE, VW, VN, VS, VSE;
double ue, uw, un, us;
double ve, vw, vn, vs;
double De, Dw, Dn, Ds;
double Fe, Fw, Fn, Fs;
double dxe, dxw, dyn, dys;
double T, TP, TN;
double H0, H00;
double cntx, cnty;
double AP, AE, AW, AN, AS;

//U

for(j=1; j<=ny-2; j++){
        for(i=1; i<=nx-3; i++){
                k=nx*j+i;
                k1=nx1*j+i;
                k2=nx2*j+i;

                AX=Uax[k2];
                AY=Uay[k2];

                cntx=-0.5*TimeStep/(Re*AX);
                cnty=-0.5*TimeStep/(Re*AY);
```

```
Bt=BetaTop[k];
Bb=BetaBot[k];

dpe=Udpe[k2];
dpw=Udpw[k2];
dpn=Udpn[k2];
dps=Udps[k2];

UP=u0[k2];
UE=u0[k2+1];
UW=u0[k2-1];
UN=u0[k2+nx2];
US=u0[k2-nx2];

VP=v0[k1];
VE=v0[k1+1];
VS=v0[k1-nx1];
VSE=v0[k1-nx1+1];

ue=0.5*(UP+UE);
uw=0.5*(UP+UW);
un=0.5*((1-Bt)*UP+(1+Bt)*UN);
us=0.5*((1-Bb)*UP+(1+Bb)*US);
vn=0.5*(VP+VE);
vs=0.5*(VS+VSE);

Fe=ue*AY;
```

```
Fw=uw*AY;

Fn=vn*AX;

Fs=vs*AX;


De=AY/(Re*dpe);

Dw=AY/(Re*dpw);

Dn=AX/(Re*dpn);

Ds=AX/(Re*dps);


dxe=De*(UE-UP);

dxw=Dw*(UP-UW);

dyn=Dn*(UN-UP);

dys=Ds*(UP-US);


Hu0[k2]=-((Fe*ue-Fw*uw)+(Fn*un-Fs*us))/(AX*AY);

H0=Hu0[k2];

if(start==1){

        Hu00[k2]=Hu0[k2];

}


H00=Hu00[k2];

Bu[k2]=UP+TimeStep*0.5*(((dxe-dxw)+(dyn-
dys))/(AX*AY)+(3*H0-H00));


AE=cntx/dpe;

AW=cntx/dpw;

AN=cnty/dpn;

AS=cnty/dps;
```

```
                AP=1-(AE+AW+AN+AS);

                Uap[k2]=AP;
                Uae[k2]=AE;
                Uaw[k2]=AW;
                Uan[k2]=AN;
                Uas[k2]=AS;
        }
}

//V

for(j=1; j<=ny-3; j++){
        for(i=1; i<=nx-2; i++){
                k=nx*j+i;
                k1=nx1*j+i;
                k2=nx2*j+i;

                AX=Vax[k1];
                AY=Vay[k1];

                cntx=-0.5*TimeStep/(Re*AX);
                cnty=-0.5*TimeStep/(Re*AY);

                Br=BetaRight[k];
                Bl=BetaLeft[k];
```

```
dpe=Vdpe[k1];
dpw=Vdpw[k1];
dpn=Vdpn[k1];
dps=Vdps[k1];

VP=v0[k1];
VE=v0[k1+1];
VW=v0[k1-1];
VN=v0[k1+nx1];
VS=v0[k1-nx1];

UP=u0[k2];
UW=u0[k2-1];
UN=u0[k2+nx2];
UNW=u0[k2+nx2-1];

TP=t0[k];
TN=t0[k+nx];
T=0.5*(TP+TN);

vn=0.5*(VP+VN);
vs=0.5*(VP+VS);
ve=0.5*((1-Br)*VP+(1+Br)*VE);
vw=0.5*((1-Bl)*VP+(1+Bl)*VW);
ue=0.5*(UP+UN);
uw=0.5*(UNW+UW);

Fe=ue*AY;
```

```
Fw=uw*AY;
Fn=vn*AX;
Fs=vs*AX;

De=AY/(Re*dpe);
Dw=AY/(Re*dpw);
Dn=AX/(Re*dpn);
Ds=AX/(Re*dps);

dxe=De*(VE-VP);
dxw=Dw*(VP-VW);
dyn=Dn*(VN-VP);
dys=Ds*(VP-VS);

Hv0[k1]=(Ra*Pr*T)-((Fe*ve-Fw*vw)+(Fn*vn-Fs*vs))/(AX*AY);
H0=Hv0[k1];
if(start==1){
        Hv00[k1]=Hv0[k1];
}
H00=Hv00[k1];

Bv[k1]=VP+0.5*TimeStep*(((dxe-dxw)+(dyn-
dys))/(AX*AY)+(3*H0-H00));

AE=cntx/dpe;
AW=cntx/dpw;
AN=cnty/dpn;
AS=cnty/dps;
```

```
                AP=1-(AE+AW+AN+AS);

                Vap[k1]=AP;
                Vae[k1]=AE;
                Vaw[k1]=AW;
                Van[k1]=AN;
                Vas[k1]=AS;
            }
        }

        cambiarc(Uas.size(), 5, MatU, 0, 0, Uas.size()-1, Uas);
        cambiarc(Uaw.size(), 5, MatU, 1, 0, Uaw.size()-1, Uaw);
        cambiarc(Uap.size(), 5, MatU, 2, 0, Uap.size()-1, Uap);
        cambiarc(Uae.size(), 5, MatU, 3, 0, Uae.size()-1, Uae);
        cambiarc(Uan.size(), 5, MatU, 4, 0, Uan.size()-1, Uan);

        cambiarc(Vas.size(), 5, MatV, 0, 0, Vas.size()-1, Vas);
        cambiarc(Vaw.size(), 5, MatV, 1, 0, Vaw.size()-1, Vaw);
        cambiarc(Vap.size(), 5, MatV, 2, 0, Vap.size()-1, Vap);
        cambiarc(Vae.size(), 5, MatV, 3, 0, Vae.size()-1, Vae);
        cambiarc(Van.size(), 5, MatV, 4, 0, Van.size()-1, Van);

        up=GaussSeidel(MatU, Bu);
        vp=GaussSeidel(MatV, Bv);
    }

    //Divergencia de Vp
```

```
void CalcularDiv(){
    int i, j, kk;
    double UP, UW, VP, VS;
    double AX, AY;

    for(j=0; j<=NY-1; j++){
        for(i=0; i<=NX-1; i++){
            kk = NX * j + i;
            k = nx * j + i;
            k1 = nx1 * j + i;
            k2 = nx2 * j + i;

            AX = Pax[k+1+nx];
            AY = Pay[k+1+nx];

            UP = up[k2+1+nx2];
            UW = up[k2+nx2];

            VP = vp[k1+1+nx1];
            VS = vp[k1+1];

            DIV[kk] = (UP - UW) / AX + (VP - VS) / AY;
        }
    }

}
```

```
//Matriz de coeficientes


void Coeficientes(){
    int i, j, kk;
    double dpe, dpw, dpn, dps;
    double AP, AE, AW, AN, AS;
    double Be, Bw, Bn, Bs;
    double AX, AY;

    for(j=0; j<=NY-1; j++){
        for(i=0; i<=NX-1; i++){
            kk = NX * j + i;
            k = nx * j + i;

            AX = Pax[k+1+nx];
            AY = Pay[k+1+nx];

            dpe = Pdpe[k+1+nx];
            dpw = Pdpw[k+1+nx];
            dpn = Pdpn[k+1+nx];
            dps = Pdps[k+1+nx];

            Be = BetaRight[k+1+nx];
            Bw = BetaLeft[k+1+nx];
            Bn = BetaTop[k+1+nx];
            Bs = BetaBot[k+1+nx];

            AE = (1 - Be) / (dpe * AX);
```

```
            AW = (1 - Bw) / (dpw * AX);
            AN = (1 - Bn) / (dpn * AY);
            AS = (1 - Bs) / (dps * AY);

            AP = - (AE + AW + AN + AS);

            ae[kk] = AE;
            aw[kk] = AW;
            an[kk] = AN;
            as[kk] = AS;
            ap[kk] = AP;
        }
}

j = 1;
i = int(NX*0.5);
    kk = NX * j + i;
    k = nx * j + i;

    AX = Pax[k+1+nx];
    AY = Pay[k+1+nx];

    dpe = Pdpe[k+1+nx];
    dpw = Pdpw[k+1+nx];
    dpn = Pdpn[k+1+nx];
    dps = Pdps[k+1+nx];

    AE = 1 / (dpe * AX);
```

```
        AW = 1 / (dpw * AX);
        AN = 1 / (dpn * AY);
        AS = 1 / (dps * AY);

        AP = - (AE + AW + AN + AS);

        AS = 0;

        ae[kk] = AE;
        aw[kk] = AW;
        an[kk] = AN;
        as[kk] = AS;
        ap[kk] = AP;

        Ensamble();
}

void Ensamble(){

    cambiarc(as.size(), 5, MAT, 0, 0, as.size() - 1, as);
    cambiarc(aw.size(), 5, MAT, 1, 0, aw.size() - 1, aw);
    cambiarc(ap.size(), 5, MAT, 2, 0, ap.size() - 1, ap);
    cambiarc(ae.size(), 5, MAT, 3, 0, ae.size() - 1, ae);
    cambiarc(an.size(), 5, MAT, 4, 0, an.size() - 1, an);

}

// Solver iteración
```

```
void SolverIteracion(){
    int i, j, kk;
    vector<double> pp;

    pp = GaussSeidel(MAT, DIV);

    for(j=0; j<=NY-1; j++){
        for(i=0; i<=NX-1; i++){
            k = nx *j +i;
            kk = NX * j + i;

            p0[k+1+nx] = pp[kk];
        }
    }

    pp.clear();

    i=0;
    for(j=1; j<=ny-2; j++){
        k = nx * j + i;
        p0[k] = p0[k+1];
    }

    i=nx-1;
    for(j=1; j<=ny-2; j++){
        k = nx * j + i;
        p0[k] = p0[k-1];
```

```
    }

    j=0;
    for(i=0; i<=nx-1; i++){
        k = nx * j +i;
        p0[k] = p0[k+nx];
    }

    j=ny-1;
    for(i=0; i<=nx-1; i++){
        k = nx * j + i;
        p0[k] = p0[k-nx];
    }

    j=0;
    i=int(nx*0.5);
    k = nx * j + i;
    p0[k] = 0;


}

//Calculo nuevas velocidades

void CalculoVelocidades (){
    int i, j;
    double PP, PN, PE;
    double AX, AY;
```

```
double grad;

//U

for(j=1; j<=ny-2; j++){
        for(i=1; i<=nx-3; i++){
                k = nx * j + i;
                k2 = nx2 * j + i;

                AX = Uax[k2];
                PP = p0[k];
                PE = p0[k+1];

                grad = (PE - PP) / AX;

                u[k2] = up[k2] - grad;
        }
}

//V

for(j=1; j<=ny-3; j++){
        for(i=1; i<=nx-2; i++){
                k = nx * j + i;
                k1 = nx1 * j + i;

                AY = Vay[k1];
                PP = p0[k];
```

```
                PN = p0[k+nx];

                grad = (PN - PP) / AY;

                v[k1] = vp[k1] - grad;
            }
        }

}

//Calculo error

double CalculoMaxError(){
    double MaxErrorU, MaxErrorV, MaxError;

    MaxErrorU = maxabs( restarv (u, u0) );
    MaxErrorV = maxabs( restarv (v, v0) );

    MaxError = fmax (MaxErrorU, MaxErrorV);

    return MaxError;
}

//Nuevo Time Step

void CalcularTimeStep(){
    int i, j;
```

```
    double Maxu, Maxv, MaxVel, Minx, Miny, MinX;
    double a, b;
    double aux1;

    Minx = minval0(Pax);
    Miny = minval0(Pay);
    Maxu = maxabs(u);
    Maxv = maxabs(v);
    MinX = fmin( Minx, Miny);
    MaxVel = fmax(Maxu, Maxv);

    if(MaxVel==0){
            TimeStep=Cdif*Minx*Minx;
            aux1=Cvisc*Re*Minx*Minx;
            TimeStep=fmin(TimeStep, aux1);
    }
    if(MaxVel!=0){
            TimeStep = Cconv * MinX / MaxVel;
    }
}

//Calculo de velocidades y presiones Medias

void CampoVelocidades(){
    int i, j;
    double UW, UP, VS, VP;

    for(j=1; j<ny-1; j++){
```

```
            for(i=1; i<nx-1; i++){
                k = nx * j + i;
                k1 = nx1 * j + i;
                k2 = nx2 * j + i;

                UW = u[k2-1];
                UP = u[k2];
                VS = v[k1-nx1];
                VP = v[k1];

                U[k] = 0.5 * (UP + UW);
                V[k] = 0.5 * (VP + VS);
            }
        }
    }

    void CampoPresiones() {

        P = dividir (p0, TimeStep);
    }

    //Comprovación continuidad

    void ComprovarContinuidad(){
        int i,j;
        double UP, UW, VP, VS;
        double AX, AY;
        double Up, Uw, Vp, Vs;
```

```
    for(j=1; j<ny-1; j++){
        for(i=1; i<nx-1; i++){
            k = nx * j + i;
            k1 = nx1 * j + i;
            k2 = nx2 * j + i;

            UP = u[k2];
            UW = u[k2-1];
            VP = v[k1];
            VS = v[k1-nx1];

            AX = Pax[k];
            AY = Pay[k];

            masa[k] = (UP - UW) * AY + (VP - VS) * AX;
        }
    }

    MaxMasaError = maxabs(masa);
}

//Obtención de datos

void GetData(){
            int i,j,k;
            vector<double> Nux0,Nux05,Nux1,Nuy05;
            vector<double> NuxCond0,NuxCond05,NuxCond1,NuyCond05;
```

```
                    vector<double> NuxConv0,NuxConv05,NuxConv1,NuyConv05;
                    vector<double> UU,VV;
                    vector<double> X,Y;
                    vector<double> Tx05,Ty05;
                    ofstream data;

                    if(Ra==1e4){

data.open("Distribucion_Final_Ra_1e4.dat",std::ofstream::out);
        }
        if(Ra==1e5){
                data.open("Distribucion_Final_Ra_1e5.dat", std::ofstream::out);
        }
        if(Ra==1e6){
                data.open("Distribucion_Final_Ra_1e6.dat", std::ofstream::out);
        }


                    Y = selc(ny,nx,Yp,0,0,ny-1);

                            Nux0 = Nusselt(ny,0,0,'x','x');
                            Nux05 = Nusselt(ny,int(nx*0.5),0,'x','x');
                            Nux1 = Nusselt(ny,nx-1,nx-1,'x','x');

                            NuxCond0 = Nusselt(ny,0,0,'x','d');
                            NuxCond05 = Nusselt(ny,int(nx*0.5),0,'x','d');
                            NuxCond1 = Nusselt(ny,nx-1,nx-1,'x','d');
```

```
                NuxConv0 = Nusselt(ny,0,0,'x','v');
                NuxConv05 = Nusselt(ny,int(nx*0.5),0,'x','v');
                NuxConv1 = Nusselt(ny,nx-1,nx-1,'x','v');

                Tx05 = selc(ny,nx,t,int(nx*0.5),0,ny-1);

                UU = selc(ny,nx,U,int(nx*0.5),0,ny-1);


            X = selr(ny,nx,Xp,0,0,nx-1);

                Nuy05 = Nusselt(nx,0,int(ny*0.5),'y','x');
                NuyCond05 = Nusselt(nx,0,int(ny*0.5),'y','d');
                NuyConv05 = Nusselt(nx,0,int(ny*0.5),'y','v');

                VV = selr(ny,nx,V,int(nx*0.5),0,ny-1);

                Ty05 = selr(ny,nx,t,int(ny*0.5),0,nx-1);

            data<<"X"<<"      "<<"Y"<<"      "<<"Nux0"<<"      "<<"Nux05"<<"
"<<"Nux1"<<"    ";
            data<<"NuxCond0"<<"                    "<<"NuxCond05"<<"
"<<"NuxCond1"<<"    ";
            data<<"NuxConv0"<<"                    "<<"NuxConv05"<<"
"<<"NuxConv1"<<"    ";
            data<<"Tx05"<<"    "<<"UU"<<"    "<<"VV"<<"    ";
            data<<"Nuy05"<<"    "<<"NuyCond05"<<"    "<<"NuyConv05"<<"
"<<"Ty05"<<endl;

            for(i=0; i<X.size(); i++){
```

```
                    data<<X.at(i)<<"        "<<Y.at(i)<<"        "<<Nux0.at(i)<<"
"<<Nux05.at(i)<<"    "<<Nux1.at(i)<<"    ";

                    data<<NuxCond0.at(i)<<"            "<<NuxCond05.at(i)<<"
"<<NuxCond1.at(i)<<"    ";

                    data<<NuxConv0.at(i)<<"            "<<NuxConv05.at(i)<<"
"<<NuxConv1.at(i)<<"    ";

                    data<<Tx05.at(i)<<"    "<<UU.at(i)<<"    "<<VV.at(i)<<"    ";

                    data<<Nuy05.at(i)<<"                "<<NuyCond05.at(i)<<"
"<<NuyConv05.at(i)<<"    "<<Ty05.at(i)<<endl;


                }


                data.close();


            }



    vector<double> Nusselt(int dim , int I , int J , char Coord , char Nuss){
    int i,j;
    double UU,VV;
    double TP,TE,TW , TN,TS;
    double dpx,dpy;
    double dpe,dpw,dpn,dps;
    double Qconv,Qcond;
    vector<double> Nu(dim,0.),NuConv(dim,0.),NuCond(dim,0.);


        if(Coord == 'x'){
            i=I;


            for(j=1; j<ny-1; j++){
```

```
k = nx*j + i;


UU = U.at(k);
TP = t.at(k);


if( i != 0 && i!= nx-1 ){
        TE = t.at(k+1);
        TW = t.at(k-1);
        dpe = Pdpe.at(k);
        dpw = Pdpw.at(k);
}
if( i == 0 ){
        TE = t.at(k+1);
        TW = t.at(k);
        dpe = Pdpw.at(k+1);
        dpw = 0.;
}
if( i == nx-1 ){
        TE = t.at(k);
        TW = t.at(k-1);
        dpe = 0.;
        dpw = Pdpe.at(k-1);
}



dpx = dpe + dpw;


Qcond = - (TE-TW)/dpx;
```

```
                Qconv = UU*TP;


                NuConv.at(j) = ( Qconv );
                NuCond.at(j) = ( Qcond );
                Nu.at(j) = ( Qcond + Qconv );
        }

}



if(Coord == 'y'){
        j = J;
        for(i=1; i<nx-1; i++){
        k = nx*j + i;


                VV = V.at(k);
                TP = t.at(k);


                if( j != 0 && j != ny-1 ){
                        TN = t.at(k+nx);
                        TS = t.at(k-nx);
                        dpn = Pdpn.at(k);
                        dps = Pdps.at(k);
                }
                if( j == 0 ){
                        TN = t.at(k+nx);
                        TS = t.at(k);
                        dpn = Pdps.at(k+nx);
```

```
                dps = 0.;
            }
            if( j == ny-1 ){
                TN = t.at(k);
                TS = t.at(k-nx);
                dpn = 0.;
                dps = Pdpn.at(k-1);
            }

            dpy = dps + dpn ;

            Qcond = - ( TN - TS )/dpy;
            Qconv =  VV*TP;

            NuConv.at(i) = ( Qconv );
            NuCond.at(i) = ( Qcond );
            Nu.at(i) = ( Qcond + Qconv );

        }

}


if(Nuss == 'd'){
        return NuCond;
}
if(Nuss == 'v'){
        return NuConv;
```

```
	}
	if(Nuss != 'd' && Nuss != 'v'){
		return Nu;
	}

}


double NusseltMedio(int dim , int I , int J , char Coord , char Nuss,bool start){
int i,j;
vector<double> x,nuss,aux;
double sol;

	if( Coord == 'x' ){
		x = selc(ny,nx,Yp,0,0,ny-1);
	}
	if( Coord == 'y' ){
		x = selr(ny,nx,Xp,0,0,nx-1);
	}

	nuss = Nusselt(dim , I , J , Coord , Nuss);


	if( start == true ){
		sol = TrapezoidRule(nuss,x);
	}
	if( start == false ){
		sol = GaussQuad(nuss,x);
```

```
        }

return sol;
}

 vol_cont(int aa, int bb){
        int i, j;
        int Psize, Vsize, Usize, MATsize, SystemSize;

        nx = aa + 2;
        ny = bb + 2;

        nx1 = nx;
        ny1 = ny - 1;

        nx2 = nx - 1;
        ny2 = ny;

        Psize = nx * ny;
        Vsize = nx1 * ny1;
        Usize = nx2 * ny2;

        SystemSize = aa * bb;

        MATsize = 5 * SystemSize;

        MatT.resize(5*Psize, 0);
        Ht0.resize(Psize, 0);
```

```
Ht00.resize(Psize, 0);
Tap.resize(Psize, 1);
Tae.resize(Psize, 0);
Taw.resize(Psize, 0);
Tan.resize(Psize, 0);
Tas.resize(Psize, 0);
Bt.resize(Psize, 0);

masa.resize(Psize, 0);

Xp.resize(Psize, 0);
Yp.resize(Psize, 0);
p0.resize(Psize, P0);
t.resize(Psize, T0);
t0.resize(Psize, T0);
h0.resize(Psize, T0);
h00.resize(Psize, T0);
U.resize(Psize, 0);
V.resize(Psize, 0);
P.resize(Psize, 0);
BetaTop.resize(Psize, 0);
BetaBot.resize(Psize, 0);
BetaRight.resize(Psize, 0);
BetaLeft.resize(Psize, 0);
TetaTop.resize(Psize, 0);
TetaBot.resize(Psize, 0);
TetaLeft.resize(Psize, 0);
TetaRight.resize(Psize, 0);
```

```
Pax.resize(Psize, 0);
Pay.resize(Psize, 0);
Pdpe.resize(Psize, 0);
Pdpw.resize(Psize, 0);
Pdpn.resize(Psize, 0);
Pdps.resize(Psize, 0);

MatV.resize(5*Vsize, 0);
Hv0.resize(Vsize, 0);
Hv00.resize(Vsize, 0);
Vap.resize(Vsize, 1);
Vae.resize(Vsize, 0);
Vaw.resize(Vsize, 0);
Van.resize(Vsize, 0);
Vas.resize(Vsize, 0);
Bv.resize(Vsize, 0);
Xv.resize(Vsize, 0);
Yv.resize(Vsize, 0);
v.resize(Vsize, 0);
v0.resize(Vsize, V0);
Ry0.resize(Vsize, 0);
Ry00.resize(Vsize, 0);
vp.resize(Vsize, 0);
Vax.resize(Vsize, 0);
Vay.resize(Vsize, 0);
Vdpe.resize(Vsize, 0);
Vdpw.resize(Vsize, 0);
Vdpn.resize(Vsize, 0);
```

```
Vdps.resize(Vsize, 0);


MatU.resize(5*Usize, 0);
Hu0.resize(Usize, 0);
Hu00.resize(Usize,0);
Uap.resize(Usize, 1);
Uae.resize(Usize, 0);
Uaw.resize(Usize, 0);
Uan.resize(Usize, 0);
Uas.resize(Usize, 0);
Bu.resize(Usize, 0);
Xu.resize(Usize, 0);
Yu.resize(Usize, 0);
u.resize(Usize, 0);
u0.resize(Usize, U0);
Rx0.resize(Usize, 0);
Rx00.resize(Usize, 0);
up.resize(Usize, 0);
Uax.resize(Usize, 0);
Uay.resize(Usize, 0);
Udpe.resize(Usize, 0);
Udpw.resize(Usize, 0);
Udpn.resize(Usize, 0);
Udps.resize(Usize, 0);


MAT.resize(MATsize, 0);


DIV.resize(SystemSize, 0);
```

```
                ap.resize(SystemSize, 1);
                ae.resize(SystemSize, 0);
                aw.resize(SystemSize, 0);
                an.resize(SystemSize, 0);
                as.resize(SystemSize, 0);
        }
};

int main (void){
    ofstream data, DATA;
    double error;
    int it = 0;
    int counter = 0;
    const clock_t begin_time = clock();
    vector<double> z, NU;
    double Nu0, Nu05, Nu1;
    double NuCond0, NuCond05, NuCond1;
    double NuConv0, NuConv05, NuConv1;

    if(Ra==1e4){
            DATA.open("Datos_Instantaneos_Ra_1e4.dat", std::ofstream::out);
    }
    if(Ra==1e5){
            DATA.open("Datos_Instantaneos_Ra_1e5.dat", std::ofstream::out);
    }
    if(Ra==1e6){
            DATA.open("Datos_Instantaneos_Ra_1e6.dat", std::ofstream::out);
```

```
}

vol_cont vc(NX,NY);

vc.coordenadas();
vc.Val_Ini();
vc.CalcularUpVp(1);
vc.CalcularDiv();
vc.Coeficientes();
vc.SolverIteracion();
vc.CalculoVelocidades();
vc.ComprovarContinuidad();
vc.CalcularT(1);

error = vc.CalculoMaxError();

Nu0=vc.NusseltMedio(vc.ny, 0, 0, 'x', 'x', true);
NuCond0=vc.NusseltMedio(vc.ny, 0, 0, 'x', 'd', true);
NuConv0=vc.NusseltMedio(vc.ny, 0, 0, 'x', 'v', true);
Nu05=vc.NusseltMedio(vc.ny, int(vc.nx*0.5), 0, 'x', 'x', true);
NuCond05=vc.NusseltMedio(vc.ny, int(vc.nx*0.5), 0, 'x', 'd', true);
NuConv05=vc.NusseltMedio(vc.ny, int(vc.nx*0.5), 0, 'x', 'v', true);
Nu1=vc.NusseltMedio(vc.ny, vc.nx-1, 0, 'x', 'x', true);
NuCond1=vc.NusseltMedio(vc.ny, vc.nx-1, 0, 'x', 'd', true);
NuConv1=vc.NusseltMedio(vc.ny, vc.nx-1, 0, 'x', 'v', true);

Time=float(clock()-begin_time)/CLOCKS_PER_SEC;
```

```
    cout<<"Time Step = "<<TimeStep<<" , Error = "<<error<<" , Masa = "<<MaxMasaError<<endl;

    DATA<<"it"<<"  "<<"t"<<"  "<<"Error"<<"  "<<"Masa"<<"  ";
    DATA<<"Nu0"<<"  "<<"NuCond0"<<"  "<<"NuConv0"<<"  ";
    DATA<<"Nu05"<<"  "<<"NuCond05"<<"  "<<"NuConv05"<<"  ";
    DATA<<"Nu1"<<"  "<<"NuCond1"<<"  "<<"NuConv1"<<"  "<<endl;
    DATA<<it<<"  "<<Time<<"  "<<error<<"  "<<MaxMasaError<<"  ";
    DATA<<Nu0<<"  "<<NuCond0<<"  "<<NuConv0<<"  ";
    DATA<<Nu05<<"  "<<NuCond05<<"  "<<NuConv05<<"  ";
    DATA<<Nu1<<"  "<<NuCond1<<"  "<<NuConv1<<"  "<<endl;

    vc.CalcularTimeStep();


    z = multi(0, vc.Xp);

mkdir("C:/Users/Miguel/Documents/ProyectoFinal/Differentially_heated_cavity/FILE");

    it++;


    error = 100;
    while(error>=MaxErrorPerm){
                        counter ++;

                        vc.u0 = vc.u;
                        vc.v0 = vc.v;
```

```
vc.Hu00 = vc.Hu0;

vc.Hv00 = vc.Hv0;

vc.t0 = vc.t;

vc.Ht00 = vc.Ht0;


vc.CalcularUpVp(2);

vc.CalcularDiv();

vc.SolverIteracion();

vc.CalculoVelocidades();

vc.ComprovarContinuidad();

vc.CalcularT(2);


error = vc.CalculoMaxError();


cout<<"It = "<<it<<" , Time Step = "<<TimeStep<<" , error = "<<error<<" , Masa = "<<MaxMasaError<<endl;


vc.CalcularTimeStep();


it++;


if(counter == PERIODICIDAD){

        Nu0=vc.NusseltMedio(vc.ny, 0, 0, 'x', 'x', true);

        NuCond0=vc.NusseltMedio(vc.ny, 0, 0, 'x', 'd', true);

        NuConv0=vc.NusseltMedio(vc.ny, 0, 0, 'x', 'v', true);

        Nu05=vc.NusseltMedio(vc.ny, int(vc.nx*0.5), 0, 'x', 'x', true);

        NuCond05=vc.NusseltMedio(vc.ny, int(vc.nx*0.5), 0, 'x', 'd',
true);
```

```
                    NuConv05=vc.NusseltMedio(vc.ny, int(vc.nx*0.5), 0, 'x', 'v',
true);
                    Nu1=vc.NusseltMedio(vc.ny, vc.nx-1, 0, 'x', 'x', true);
                    NuCond1=vc.NusseltMedio(vc.ny, vc.nx-1, 0, 'x', 'd', true);
                    NuConv1=vc.NusseltMedio(vc.ny, vc.nx-1, 0, 'x', 'v', true);

                    Time=float(clock()-begin_time)/CLOCKS_PER_SEC;

                    DATA<<it<<"          "<<Time<<"          "<<error<<"
"<<MaxMasaError<<"    ";
                    DATA<<Nu0<<"    "<<NuCond0<<"    "<<NuConv0<<"    ";
                    DATA<<Nu05<<"    "<<NuCond05<<"    "<<NuConv05<<"
";
                    DATA<<Nu1<<"    "<<NuCond1<<"    "<<NuConv1<<"
"<<endl;

                counter = 0;
            }
    }

    vc.CampoVelocidades();
    write_vtu_2D (it, vc.nx, vc.ny, vc.Xp, vc.Yp, z, vc.U, vc.V, z, vc.t);

    Nu0=vc.NusseltMedio(vc.ny, 0, 0, 'x', 'x', true);
    NuCond0=vc.NusseltMedio(vc.ny, 0, 0, 'x', 'd', true);
    NuConv0=vc.NusseltMedio(vc.ny, 0, 0, 'x', 'v', true);
    Nu05=vc.NusseltMedio(vc.ny, int(vc.nx*0.5), 0, 'x', 'x', true);
    NuCond05=vc.NusseltMedio(vc.ny, int(vc.nx*0.5), 0, 'x', 'd', true);
    NuConv05=vc.NusseltMedio(vc.ny, int(vc.nx*0.5), 0, 'x', 'v', true);
    Nu1=vc.NusseltMedio(vc.ny, vc.nx-1, 0, 'x', 'x', true);
```

```
    NuCond1=vc.NusseltMedio(vc.ny, vc.nx-1, 0, 'x', 'd', true);
    NuConv1=vc.NusseltMedio(vc.ny, vc.nx-1, 0, 'x', 'v', true);

    Time=float(clock()-begin_time)/CLOCKS_PER_SEC;

    DATA<<it<<"    "<<Time<<"    "<<error<<"    "<<MaxMasaError<<"    ";
    DATA<<Nu0<<"    "<<NuCond0<<"    "<<NuConv0<<"    ";
    DATA<<Nu05<<"    "<<NuCond05<<"    "<<NuConv05<<"    ";
    DATA<<Nu1<<"    "<<NuCond1<<"    "<<NuConv1<<"    "<<endl;

    DATA.close();

    vc.GetData();

}
```