4. Código de *Cryojab 2.0*

4.1. Código del modulo principal

Código de ***subestructuras.h***

```
#if !defined subestructura_hpp
#define subestructura_hpp
#include "stdio.h"
#include "stdlib.h"
#include "math.h"
#include "maths.h"

class Subestructura
{
//Declaracion de variables y funciones privadas, solo accesibles a la clase subestructura
private:

    //Número de nodos en direccion x e y
    int indice[4];
    long nx,ny;
    long ipath;

    //parametros del mallado del bloque
    long nc;
    long nn;

    //Longitud en direccion x e y
    float *dxNod,*dyNod;
    float dx,dy;
    float dz;

    //número de materiales
    int nMat;

    //Flotante para las temperaturas iniciales
    float tini;

    //Materiales
    float *kMat;
    float *roMat;
```

```
float *cpMat;
float *velMat;

//Materiales de los nodos
float *kNod;
float *roNod;
float *cpNod;
float *rocpNod;

//Número de material
int **númeroMaterial;
//vector de resistencias superficiales
float  *r;

//Matrices del sistema
float  *a,*b,*c,*d;

//matrices con autovalores
float  *e,*f,*p,*l,*invp,*h;

//matrices para el calculo de las temperaturas nodales
float  *st,*su,*n,*s,*sg;

//Matrices para el calculo del vector VP
float *spt,*spu,*spg;

//Matriz MA,VP y Q del acoplamiento
float *ma,*vp, *q;

//Vectores con las coordenadas x,y de los nodos y de los contornos
float *xNod,*yNod,*xcon,*ycon;

//posicion del vertice inferior izquierda
float  x,y;

//Tamaño del bloque
float lx;
float ly;

//Paso de tiempo en segundos
float dt;
float deltat;
```

```
//Número de pasos de tiempo
int nt;

//Número de contornos totales del problema
int n_con;

//Número de elementos de contorno (inicializacion a cero)
int n_elemcontorno;


//Vector con temperaturas nodales en los nodos y en el tiempo (nt*nn)
//TN (t,nodo) = TN(t+nt*nodo)
float *tnNod;

//Vector de generacion nodal (W/m3)
float *gNod;

//Vector de generacion nodal modificada por 1/ro*cp
float *gNod_rocp;

//Vector con temperaturas superficiales (excitaciones) tcc (0 a 2*nx+2ny-1) (0 a nc-1)
float *tcc;

//vector de temp de los nodos
float *tnn;

//vectores rcontorno y tcontorno
float *tcontorno;
float *rcontorno;
float *scontorno; //coordenadas de los nodos

//coordenadas incial y final el elemento del bloque (0,0)
float x1,x2;

//Resistencia termica  adyacente en caso de condicion de simetria
float  R_SIM;
//Resistencia termica adyacente  en caso de condicion de temperatura impuesta
float  R_TIM;
//Número PI
float PI;
```

```
//Declaracion de variables y funciones publicas, accesibles a todas las clases
public:

    //declaracion del constructor 1
    Subestructura(float);
    Subestructura(int nMatDat,  float tiniDat, float *kMatDat, float *roMatDat, float
    *cpMatDat, float *velMatDat, int **númeroMaterialDat,  int nxDat, int nyDat, int ntDat,
    float deltatDat, float lxDat, float lyDat);


    //destructor
    ~Subestructura(void);
    //Funciones de calculo
    float resistencia(float  k,float  d,float  r);
    void calcularMatrices2D(float dt, int t);
    void calcularMatricesTemperaturas(void);
    void calcularMatricesMAVP(void);
    void posicionNodos(float  *xNod,float  *yNod);
    void calcularTnNod(int t);
    void calcularVp(int t);
    void calcularQ(int t);
    void dameCondContorno(int posicion, float *tnn, float *Rtemp,float *coord_nodos, int t);
    void ponCondContorno(int posicion, float *tnn, float *Rtemp, float *coord_nodos,int t);
    void resolverTcc(float *tcc_real,int t);
    void igualarTccTcontorno(int t);
    void devolverTnNod(float *temp,int nt,int nn);
    void ponIndice(int posicion, int indice);
    void dameIndice(int posicion, int &indice);
    void ponGeneracion (float *gen, int t);
    void ponTGeneracion (float *gen, int t);
    void devolverq(float *fluq);
    int interpolador(float *x,float *tx,int nx,float *s,float *ts,int ns);
    };
#endif
```

Código de ***subestructuras.cpp***

```
#include "stdio.h"
#include "stdlib.h"
#include "math.h"
```

```
#include "maths.h"
#in clude "subestructura.h"

////////////////////////////////////////////////////////////////////////////////////////////////
//     Implementacion del constructor nº1
////////////////////////////////////////////////////////////////////////////////////////////////
Subestructura::Subestructura(int nMatDat,float tiniDat,float *kMatDat,float *roMatDat,
        float *cpMatDat,float *velMatDat,int **númeroMaterialDat,
        int nxDat,int nyDat,int ntDat,float deltatDat, float lxDat,float lyDat)
{
    //Lectura de los datos
    int i,j;
    nt = ntDat;
    nx = nxDat;
    ny = nyDat;
    lx = lxDat;
    ly = lyDat;
    tini = tiniDat;

    dx = lx/nx;
    dy = ly/ny;
    dz = float(1);

    deltat = deltatDat;
    dt = deltat;
    nMat = nMatDat;
    númeroMaterial = new int *[ny];
    for(i=0;i<ny;i++)
        númeroMaterial[i] = new int [nx];

    for(i=0;i<ny;i++)
        for(j=0;j<nx;j++)
            númeroMaterial[i][j] = númeroMaterialDat[i][j];
    kMat = new float [nMat];
    roMat = new float [nMat];
    cpMat = new float [nMat];
    velMat = new float [nMat];

    for(i=0;i<nMat;i++)
    {
        kMat[i] = kMatDat[i];
        roMat[i] = roMatDat[i];
```

```
        cpMat[i] = cpMatDat[i];
        velMat[i] = velMatDat[i];
    }


    ////////////////////////////////////////////////////////////////////////////////////////
    //      Valores para las constantes

    //Resistencia termica  adyacente en caso de condicion de simetria
    R_SIM = -1;
    //Resistencia termica adyacente  en caso de condicion de temperatura impuesta
    R_TIM = -2;

    //Número PI
    PI = float(3.141592);

    ////////////////////////////////////////////////////////////////////////////////////////
    //      Declaracion de variables del metodo constructor

    //Contadores
    int a;

    //Coordenadas del sistema de referencia de la subestructura respecto del stma global
    x1 = 0;
    x2 = 0;
    x = 0;
    y = 0;
    ipath =long(1);

    //Número total de nodos
    nc=2*nx+2*ny;
    nn=nx*ny;

    //Mallado
    dxNod = new float [nn];
    dyNod = new float [nn];
    for(i=0;i<nn;i++)
    {
        dxNod[i]= dx;
        dyNod[i]= dy;
    }
```

```
//Reservar las matrices de kNod, cpNod, roNod rocpNod
kNod = new float[nn];
roNod = new float[nn];
cpNod = new float[nn];
rocpNod= new float[nn];

//Coordenadas de los nodos
xNod = new float[nn];
yNod = new float[nn];

//Reservar los vectores de las matrices de calculo de la TN y los Q
n = new float  [nn*nc];
su = new float  [nn*nc];
sg = new float  [nn*nn];
st = new float  [nn*nn];
ma = new float [nc*nc];
vp = new float [nc];
q = new float [nc*nt];
tnNod = new float [nt*nn];
gNod = new float [nt*nn];
gNod_rocp = new float [nt*nn];
tcc = new float [nt*nc];
spt = new float [nc*nn];
spu = new float [nc*nc];
spg = new float [nc*nn];
tcontorno = new float [nc];
rcontorno = new float [nc];
scontorno = new float [nc];

//Poner los vectores a 0
for(i=0;i<nn*nc;i++)
{
    n[i] = float(0);
    su[i] = float(0);
    spt[i]= float(0);
    spg[i] = float(0);
}
for(i=0;i<nn*nn;i++)
{
    st[i] = float(0);
    sg[i] = float(0);
}
```

```
for(i=0;i<nc*nc;i++)
{
    ma[i] = float(0);
    spu[i] = float(0);
}

for(i=0;i<nc;i++)
{
    vp[i] = float(0);
    tcontorno[i] = float(0);
    rcontorno[i] = float(0);
}

for(i=0;i<nt*nc;i++)
{
    q[i] = float(0);
    tcc[i] = tini;
}
for(i=0;i<nt*nn;i++)
{
    tnNod[i] = float(0);
    gNod_rocp[i]=float(0);
    gNod[i]=float(0);
}

//La temperatura de los nodos en t=0 es la tini;
for(i=0;i<nn;i++)
{
    tnNod[0+nt*i] = tini;
}

//Escribir las matrices de propiedades.
for(i=0;i<ny;i++)
{
    for(j=0;j<nx;j++)
    {
        //a es el tipo de material expresado como un entero de 0 a N
        a = númeroMaterial[i][j];

        //Asignamos a la k,ro,cp de los nodos, la k ro cp del material correspondiente
        kNod[i+ny*j] = kMat[a];
        roNod[i+ny*j] = roMat[a];
```

```
            cpNod[i+ny*j] = cpMat[a];
            rocpNod[i+ny*j]=roNod[i+ny*j]*cpNod[i+ny*j];
        }
    }
    //Al final tendremos todos los nodos con sus propiedades agrupados en matrices.
}

////////////////////////////////////////////////////////////////////////////////////////////////
//     Destructor para liberar la memoria dinamica del sistema
////////////////////////////////////////////////////////////////////////////////////////////////
Subestructura::~Subestructura(void)
{
    int i;
    //Borrar la memoria reservada sobrante.
    delete [] a;
    delete [] b;
    delete [] c;
    delete [] d;
    delete [] e;
    delete [] f;
    delete [] h;
    delete [] p;
    delete [] l;
    delete [] invp;
    delete[] dxNod;
    delete[] dyNod;
    for(i=0;i<ny;i++)
        delete[] númeroMaterial[i];
    delete[] númeroMaterial;
    delete[] kMat;
    delete[] roMat;
    delete[] cpMat;
    delete[] velMat;
    delete[] kNod;
    delete[] roNod;
    delete[] cpNod;
    delete[] rocpNod;
    delete[] xNod;
    delete[] yNod;
    delete[] n;
    delete[] sg;
    delete[] su;
```

```
    delete[] st;
    delete[] ma;
    delete[] vp;
    delete[] q;
    delete[] tnNod;
    delete[] gNod;
    delete[] gNod_rocp;
    delete[] tcc;
    delete[] spt;
    delete[] spu;
    delete[] spg;
    delete[] rcontorno;
    delete[] scontorno;
    delete[] tcontorno;
}

////////////////////////////////////////////////////////////////////////////////////////////////
//      Finalidad: resistencia termica asociada a un nodo
//      Entrada: k, conductividad del nodo , d tamaño del nodo en la
//                  direccion deseada, r resistencia asociada a contorno
//                  o nodo adyacente.
//      Salida: por valor, resistencia asociada al nodo
//     Notas: utiliza las constantes definidas al comienzo del
//                fichero R_SIM, R_TIM.
////////////////////////////////////////////////////////////////////////////////////////////////

float  Subestructura::resistencia(float  k,float  d,float  r)
{
    //Si es condicion de simetria
    if(r == R_SIM)
    {
        return 0.;
    }

    //Si es condicion de temperatura impuesta
    if(r == R_TIM)
    {
        return 2*k/d;
    }

    //Si es condicion de conveccion o nodo adyacente, cuya resistencia sera dx/2k
    return 2*k/(2*k*r+d);
```

```
}

////////////////////////////////////////////////////////////////////////////////////////////////
//    Finalidad: Descripcion de cada bloque, calculo de las matrices A y B
//    Notas: utiliza una rutina de inversion de matrices general
//              de la biblioteca I.M.S.L. en FORTRAN. (LINRG)
//              Si las propiedades no son cosntantes, en cada paso de tiempo
//              habrán de calcularse las conductividades nodales y todas las matrices.
//    Datos: incremento de tiempo, dt. Instante de tiempo, t.
////////////////////////////////////////////////////////////////////////////////////////////////

void Subestructura::calcularMatrices2D(float dt, int t)
{
    //Contadores, i direccion vertical, j direccion horizontal
    long i = 0,k = 0,j = 0;

    //Resistencia de un nodo adyacente
    float R_INT;

    //Marices complejas paralos autovalores y autovectores
    f_complex *lcom;
    f_complex *pcom;

    //Reserva de memoria para las matrices a y b
    a = new float  [nn*nn];
    b = new float  [nc];
    c = new float [nc];
    d = new float [nc];
    e = new float  [nn];
    h = new float [nn];
    f = new float  [nn];
    l = new float  [nn];
    p = new float  [nn*nn];
    invp = new float  [nn*nn];
    lcom = new f_complex  [nn];
    pcom = new f_complex  [nn*nn];

    //Inicializar las matrices a y b
    for(i=0;i<nn;i++)
    {
        for(j=0;j<nn;j++)
        {
```

```
            a[i+nn*j] = 0;
        }
    }

    for(i=0;i<nc;i++)
    {
        b[i]=0;
    }

    for(i=0;i<nn;i++)
    {
        h[i]=0;
    }

    R_INT=float(0);

    //Rellenar la matriz A, matriz de rigidez
    for(j=0;j<nx;j++)
    {
        for(i=0;i<ny;i++)
        {
            //si no es un nodo extremo
            if(j!=0)
            {
                R_INT = (float)0.5*dxNod[(i+ny*(j-1))]/kNod[(i+ny*(j-1))];
                a[(i+ny*j)+nn*(i+ny*(j-1))]+=resistencia(kNod[(i+ny*j)],dxNod[(i+ny*j)],
R_INT)*dyNod[(i+ny*j)]/(rocpNod[i+ny*j]*dxNod[i+ny*j]*dyNod[i+ny*j]);
                a[(i+ny*j)+nn*(i+ny*j)]-=resistencia(kNod[(i+ny*j)],dxNod[(i+ny*j)],
R_INT)*dyNod[(i+ny*j)]/(rocpNod[i+ny*j]*dxNod[i+ny*j]*dyNod[i+ny*j]);
            }
            //si es un nodo extremo
            else
            {
                //Para j=0;
                a[(i+ny*0)+nn*(i+ny*0)]-=resistencia(kNod[(i+ny*0)],dxNod[(i+ny*0)],
rcontorno[i+2*nx])*dyNod[(i+ny*0)]/(rocpNod[i+ny*j]*dxNod[i+ny*j]*dyNod[i+ny*j]);
            }
            //si no es un nodo extremo
            if((j!=nx-1))
            {
                R_INT = (float)0.5*dxNod[(i+ny*(j+1))]/kNod[(i+ny*(j+1))];
                a[(i+ny*j)+nn*(i+ny*(j+1))]+=resistencia(kNod[(i+ny*j)],dxNod[(i+ny*j)],
R_INT)*dyNod[(i+ny*j)]/(rocpNod[i+ny*j]*dxNod[i+ny*j]*dyNod[i+ny*j]);
                a[(i+ny*j)+nn*(i+ny*j)]-=resistencia(kNod[(i+ny*j)],dxNod[(i+ny*j)],
```

```
R_INT)*dyNod[(i+ny*j)]/(rocpNod[i+ny*j]*dxNod[i+ny*j]*dyNod[i+ny*j]);
            }
            //si es un nodo extremo
            else
            {
                //Para j=nx-1;
                a[(i+ny*(nx-1))+nn*(i+ny*(nx-1))]-=resistencia(kNod[(i+ny*(nx-1))],   dxNod
[(i+ny*(nx-1))], rcontorno[i+2*nx+ny])*dyNod[(i+ny*(nx-1))] / (rocpNod [i+ny*j]* dxNod
[i+ny*j]*dyNod[i+ny*j]);
            }
            //si no es un nodo extremo
            if(i!=0)
            {
                R_INT = (float)0.5*dyNod[(i-1+ny*j)]/kNod[(i-1+ny*j)];
                a[(i+ny*j)+nn*((i-1)+ny*j)]+=resistencia(kNod[(i+ny*j)],dyNod[(i+ny*j)],
R_INT)*dxNod[(i+ny*j)]/(rocpNod[i+ny*j]*dxNod[i+ny*j]*dyNod[i+ny*j]);
                a[(i+ny*j)+nn*(i+ny*j)]-=resistencia(kNod[(i+ny*j)],dyNod[(i+ny*j)],
R_INT)*dxNod[(i+ny*j)]/(rocpNod[i+ny*j]*dxNod[i+ny*j]*dyNod[i+ny*j]);
            }
            //si es un nodo extremo
            else
            {
                //Para i=0;
                a[(0+ny*j)+nn*(0+ny*j)]-=resistencia(kNod[(0+ny*j)],dyNod[(0+ny*j)],
rcontorno[j])*dxNod[(0+ny*j)]/(rocpNod[i+ny*j]*dxNod[i+ny*j]*dyNod[i+ny*j]);
            }
            //si no es un nodo extremo
            if(i!=(ny-1))
            {
                R_INT = (float)0.5*dyNod[(i+1+ny*j)]/kNod[(i+1+ny*j)];
                a[(i+ny*j)+nn*((i+1)+ny*j)]+=resistencia(kNod[(i+ny*j)],dyNod[(i+ny*j)],
R_INT)*dxNod[(i+ny*j)]/(rocpNod[i+ny*j]*dxNod[i+ny*j]*dyNod[i+ny*j]);
                a[(i+ny*j)+nn*(i+ny*j)]-=resistencia(kNod[(i+ny*j)],dyNod[(i+ny*j)],
R_INT)*dxNod[(i+ny*j)]/(rocpNod[i+ny*j]*dxNod[i+ny*j]*dyNod[i+ny*j]);
            }
            //si es un nodo extremo
            else
            {
                //Para i=ny-1;
                a[((ny-1)+ny*j)+nn*((ny-1)+ny*j)]-=resistencia(kNod[((ny-1)+ny*j)], dyNod[((ny-
1)+ny*j)],rcontorno[j+nx])*dxNod[(ny-1+ny*j)]/    (rocpNod[i+ny*j]*dxNod
[i+ny*j]*dyNod[i+ny*j]);
            }
            //Incluir los terminos de generacion.
            a[(i+ny*j)+nn*(i+ny*j)] += gNod_rocp[t+nt*(i+ny*j)];
        }
```

```
    }

    //Matriz B, C y D almacenada como vector
    for(j=0;j<nx;j++)
    {
        b[j]=resistencia(kNod[(0+ny*j)],dyNod[(0+ny*j)], rcontorno[j])*dxNod [(0+
ny*j)]/(rocpNod [0+ny*j]*dxNod[0+ny*j]*dyNod[0+ny*j]);
        b[nx+j]=resistencia(kNod[(ny-1+ny*j)],dyNod[(ny-1+ny*j)], rcontorno[j+nx])*dxNod[(ny-
1+ny*j)]/(rocpNod[ny-1+ny*j]*dxNod[ny-1+ny*j] dyNod[ny-1+ny*j]);
    }
    for(i=0;i<ny;i++)
    {
        b[2*nx+i]=resistencia(kNod[(i+ny*0)],dxNod[(i+ny*0)],rcontorno[i+2*nx])* dyNod
[(i+ny*0)]/(rocpNod[i+ny*0]*dxNod[i+ny*0]*dyNod[i+ny*0]);
        b[2*nx+ny+i]=resistencia(kNod[(i+ny*(nx-1))],dxNod[(i+ny*(nx-1))], [i+2*nx+ny])*dyNod
[(i+ny*(nx-1))]/ (rocpNod[i+ny*(nx-1)]*dxNod[i+ny*(nx-1)]*dyNod[i+ny*(nx-1)]);
    }
    for(j=0;j<nx;j++)
    {
        c[j]=(2*kNod[ny*j]*dxNod[ny*j]*dz)/dyNod[ny*j];
        c[nx+j]=(2*kNod[ny*(j+1)-1]*dxNod[ny*(j+1)-1]*dz)/dyNod[ny*(j+1)-1];
    }

    for(j=0;j<ny;j++)
    {
        c[2*nx+j]=(2*kNod[j]*dyNod[j]*dz)/dxNod[j];
        c[2*nx+ny+j]=(2*kNod[j+nn-ny]*dyNod[j+nn-ny]*dz)/dxNod[j+nn-ny];
    }
    for(j=0;j<nx;j++)
    {
        d[j]=(2*kNod[ny*j]*dxNod[ny*j]*dz)/dyNod[ny*j];
        d[nx+j]=(2*kNod[ny*(j+1)-1]*dxNod[ny*(j+1)-1]*dz)/dyNod[ny*(j+1)-1];
    }

    for(j=0;j<ny;j++)
    {
        d[2*nx+j]=(2*kNod[j]*dyNod[j]*dz)/dxNod[j];
        d[2*nx+ny+j]=(2*kNod[j+nn-ny]*dyNod[j+nn-ny]*dz)/dxNod[j+nn-ny];
    }

    //calculo de los autovalores y autovectores
    EVCRG(&nn,a,&nn,lcom,pcom,&nn);

    //Tomar la parte real de los autovectores y autovalores
```

```
    for(i=0;i<nn;i++)
    {
        for(j=0;j<nn;j++)
        {
            p[i+nn*j]=pcom[i+nn*j].re;
        }
        l[i]=lcom[i].re;
    }
    delete [] lcom;
    delete [] pcom;

    //Calcular la inversa de la matriz P de autovectores. Se almacenara en la misma direccion.
    LINRG(&nn,p,&nn,invp,&nn);
    printf("\nt = %d :  ", t);
    printf("Realizada la inversa");

    //Matrices E y F y H
    for(i=0;i<nn;i++)
    {
        if(l[i]==0)
        {
            e[i]=float(0.5)*dt;
            f[i]=float(0.5)*dt;
            h[i]=float(0);
        }
        else
        {
            e[i]=(float)(exp(l[i]*dt)*l[i]*dt+(float)1-exp(l[i]*dt))/(l[i]*l[i]*dt);
            f[i]=(float)(-l[i]*dt-(float)1+exp(l[i]*dt))/(l[i]*l[i]*dt);
            h[i]=(float)(exp(l[i]*dt)-(float)1)/(l[i]);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////
//    Obtención de las matrices ST,SU,N,SG
//    La solucion del campo de temperatura nodal viene dado a partir de las
//    cuatro matrices ST,SU,N,SG:
//    Tn(t+Dt) = ST*Tn(t) + SU*Tcc(t) + N*Tcc(t+Dt) + SG*{g/rocp}(t+Dt)
////////////////////////////////////////////////////////////////////////////////////////////////

void Subestructura::calcularMatricesTemperaturas(void)
```

```
{
    //Contadores, i direccion vertical, j direccion horizontal
    long i = 0,k = 0,j = 0,u,v,w;

    for(i=0;i<nn*nn;i++)
    {
        st[i]=0;
    }
    for(i=0;i<nn*nc;i++)
    {
        su[i]=0;
        n[i]=0;
    }

    //matriz N
    for(u=0;u<nn;u++)
    {
        for(v=0;v<nn;v++)
        {
            //abajo v
            for(w=0;w<nx;w++)
            {
                //abajo w
                n[u+nn*(w)]+=p[u+nn*v]*f[v]*invp[v+nn*(0+ny*w)]*b[w];
                //arriba w
                n[u+nn*(w+nx)]+=p[u+nn*v]*f[v]*invp[v+nn*((ny-1)+ny*w)]*b[w+nx];
            }
            for(w=0;w<ny;w++)
            {
                //izda w
                n[u+nn*(w+2*nx)]+=p[u+nn*v]*f[v]*invp[v+nn*(w+ny*0)]*b[w+2*nx];
                //dcha w
                n[u+nn*(w+2*nx+ny)]+=p[u+nn*v]*f[v]*invp[v+nn*(w+ny*(nx-
1))]*b[w+2*nx+ny];
            }
        }
    }

    //Matriz SU
    for(u=0;u<nn;u++)
     {
        for(v=0;v<nn;v++)
        {
```

```
            for(w=0;w<nx;w++)
            {
                //abajo w
                su[u+nn*(w)]+=p[u+nn*v]*e[v]*invp[v+nn*(0+ny*w)]*b[w];
                //arriba w
                su[u+nn*(w+nx)]+=p[u+nn*v]*e[v]*invp[v+nn*((ny-1)+ny*w)]*b[w+nx];
            }
            for(w=0;w<ny;w++)
            {
            //izda w
            su[u+nn*(w+2*nx)]+=p[u+nn*v]*e[v]*invp[v+nn*(w+ny*0)]*b[w+2*nx];
            //dcha w
            su[u+nn*(w+2*nx+ny)]+=p[u+nn*v]*e[v]*invp[v+nn*(w+ny*(nx-1))]*b[w+2*nx+ny];
            }
        }
    }

    //Matriz ST
    for(u=0;u<nn;u++)
    {
        for(w=0;w<nn;w++)
        {
            for(v=0;v<nn;v++)
            {
                st[u+nn*w]+=p[u+nn*v]*(float)exp(l[v]*dt)*invp[v+nn*w];
            }
        }
    }

    //MATRIZ SG
    for(u=0;u<nn;u++)
    {
        for(w=0;w<nn;w++)
        {
            for(v=0;v<nn;v++)
            {
                sg[u+nn*v]+=p[u+nn*v]*h[v]*invp[v+nn*w];
            }
        }
     }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//      Calcular las matrices del flujo de calor
//      expresado como Qcc(t) = MA·Tcc(t)
//      Crear la matriz MA = [C]·[P]·[F]·[P-1]+[D]
//      Calcular un conjunto de matrices que proporcione el vector VP de terminos
//      independientes en funcion de las temperaturas.
//      VP(t) = SPT*TN(t)+SPU*Tc(t) + SPG*{g/rocp}(t)
//      terminos que dependen de las temperaturas nodales
//      puesta a cero en el programa principal
////////////////////////////////////////////////////////////////////////////////////////////////


void Subestructura::calcularMatricesMAVP(void)
{
    //Contadores, i direccion vertical, j direccion horizontal
    long i = 0,k = 0,j = 0;

    //Se inicializa a cero.
    for(i=0;i<nc*nc;i++)
        ma[i]=float(0);

    for(j=0;j<nc;j++)
    {
        ma[(1+nc)*j]+=-c[j];
    }
    for(j=0;j<nx;j++)
    {
        for(i=0;i<nn;i++)
        {
            for(k=0;k<nx;k++)
            {
                ma[j+nc*k]+=c[j]*p[ny*j+nn*i]*f[i]*invp[i+nn*ny*k]*b[k];
                ma[j+nc*(nx+k)]+=c[j]*p[ny*j+nn*i]*f[i]*invp[i+nn*(ny*(1+k)-1)]*b[nx+k];
            }
            for(k=0;k<ny;k++)
            {
                ma[j+nc*(2*nx+k)]+=c[j]*p[ny*j+nn*i]*f[i]*invp[i+nn*k]*b[2*nx+k];
                ma[j+nc*(2*nx+ny+k)]+=c[j]*p[ny*j+nn*i]*f[i]* invp[i+nn*(nn-
ny+k)]*b[2*nx+ny+k];
            }
        }
    }
    for(j=0;j<nx;j++)
```

```
    {
        for(i=0;i<nn;i++)
        {
            for(k=0;k<nx;k++)
            {
                ma[(nx+j)+nc*k]+=c[nx+j]*p[(ny*(j+1)-1)+nn*i]*f[i]*invp[i+nn*ny*k]*b[k];
                ma[(nx+j)+nc*(nx+k)]+=c[nx+j]*p[(ny*(j+1)-1)+nn*i]*f[i]* invp[i+nn*(ny*(1+k)-
1)]*b[nx+k];
            }
            for(k=0;k<ny;k++)
            {
                ma[(nx+j)+nc*(2*nx+k)]+=c[nx+j]*p[(ny*(j+1)-1)+nn*i]*f[i]* invp[i+
nn*k]*b[2*nx+k];
                ma[(nx+j)+nc*(2*nx+ny+k)]+=c[nx+j]*p[(ny*(j+1)-1)+nn*i]*f[i]* invp[i+nn*(nn-
ny+k)]*b[2*nx+ny+k];
            }
        }
    }


    for(j=0;j<ny;j++)
    {
        for(i=0;i<nn;i++)
        {
            for(k=0;k<nx;k++)
            {
                ma[(2*nx+j)+nc*k]+=c[2*nx+j]*p[j+nn*i]*f[i]*invp[i+nn*ny*k]*b[k];
                ma[(2*nx+j)+nc*(nx+k)]+=c[2*nx+j]*p[j+nn*i]*f[i]*  invp[i+ nn*(ny*(1 + k)-
1)]*b[nx+k];
            }
            for(k=0;k<ny;k++)
            {
                ma[(2*nx+j)+nc*(2*nx+k)]+=c[2*nx+j]*p[j+nn*i]*f[i]*invp[i+nn*k]*b[2*nx+k];
                ma[(2*nx+j)+nc*(2*nx+ny+k)]+=c[2*nx+j]*p[j+nn*i]*f[i]* invp[i+nn*(nn-
ny+k)]*b[2*nx+ny+k];
            }
        }
    }
    for(j=0;j<ny;j++)
    {
        for(i=0;i<nn;i++)
        {
            for(k=0;k<nx;k++)
            {
                ma[(2*nx+ny+j)+nc*k]+=c[2*nx+ny+j]*p[nn-ny+j+nn*i]*f[i]* invp[i+
nn*ny*k]*b[k];
```

```
                ma[(2*nx+ny+j)+nc*(nx+k)]+=c[2*nx+ny+j]*p[nn-ny+j+nn*i]*f[i]* invp[i+
nn*(ny*(1+k)-1)]*b[nx+k];
            }
            for(k=0;k<ny;k++)
            {
                ma[(2*nx+ny+j)+nc*(2*nx+k)]+=c[2*nx+ny+j]*p[nn-ny+j+nn*i]*f[i]*
invp[i+ nn*k]*b[2*nx+k];
                ma[(2*nx+ny+j)+nc*(2*nx+ny+k)]+=c[2*nx+ny+j]*p[nn-ny+j+nn*i]*f[i]*
invp[i+nn*(nn-ny+k)]*b[2*nx+ny+k];
            }
        }
    }
    //fin ma
    for(i=0;i<nn;i++)
    {
        for(k=0;k<nn;k++)
        {
            for(j=0;j<nx;j++)
            {
                spt[j+nc*k]=c[j]*p[ny*j+nn*i]*(float)exp(l[i]*dt)*invp[i+nn*k];
                spt[(nx+j)+nc*k]=c[nx+j]*p[ny*(j+1)-1+nn*i]* (float)exp(l[i]*dt)*invp[i+nn*k];
            }
            for(j=0;j<ny;j++)
            {
                spt[(2*nx+j)+nc*k]=c[2*nx+j]*p[j+nn*i]*(float)exp(l[i]*dt)*invp[i+nn*k];
                spt[(2*nx+j+ny)+nc*k]=c[2*nx+ny+j]*p[nn-ny+j+nn*i]*(float)exp(l[i]*dt)*
invp[i+nn*k];
            }
        }
    }

    //terminos que dependen de las temperaturas en los contornos
    for(j=0;j<nx;j++)
    {
        for(i=0;i<nn;i++)
        {
            for(k=0;k<nx;k++)
            {
                spu[j+nc*k]=c[j]*p[ny*j+nn*i]*e[i]*invp[i+nn*ny*k]*b[k];
                spu[j+nc*(nx+k)]=c[j]*p[ny*j+nn*i]*e[i]*invp[i+nn*(ny*(1+k)-1)]*b[nx+k];
            }
            for(k=0;k<ny;k++)
            {
                spu[j+nc*(2*nx+k)]=c[j]*p[ny*j+nn*i]*e[i]*invp[i+nn*k]*b[2*nx+k];
```

```
                spu[j+nc*(2*nx+k+ny)]=c[j]*p[ny*j+nn*i]*e[i]* [i+nn*(nn-ny+k)]*b[2*nx+ny+k];
            }
        }
    }

    for(j=0;j<nx;j++)
    {
        for(i=0;i<nn;i++)
        {
            for(k=0;k<nx;k++)
            {
                spu[(nx+j)+nc*k]=c[nx+j]*p[ny*(1+j)-1+nn*i]*e[i]*invp[i+nn*ny*k]*b[k];
                spu[nx+j+nc*(k+nx)]=c[nx+j]*p[ny*(1+j)-1+nn*i]*e[i]* invp[i+nn*(ny*(1+k)-
1)]*b[nx+k];
            }
            for(k=0;k<ny;k++)
            {
                spu[(nx+j)+nc*(2*nx+k)]=c[nx+j]*p[ny*(1+j)-1+nn*i]*e[i]* invp[i+
nn*k]*b[2*nx+k];
                spu[(nx+j)+nc*(2*nx+ny+k)]=c[nx+j]*p[ny*(1+j)-1+nn*i]*e[i]*  invp[i+ nn*(nn-
ny+k)]*b[2*nx+ny+k];
            }
        }
    }

    for(j=0;j<ny;j++)
    {
        for(i=0;i<nn;i++)
        {
            for(k=0;k<nx;k++)
            {
                spu[(2*nx+j)+nc*(k)]=c[2*nx+j]*p[j+nn*i]*e[i]*invp[i+nn*ny*k]*b[k];
                spu[(2*nx+j)+nc*(k+nx)]=c[2*nx+j]*p[j+nn*i]*e[i]* invp[i+nn*(ny*(1+k)-
1)]*b[nx+k];
            }
            for(k=0;k<ny;k++)
            {
                spu[(2*nx+j)+nc*(2*nx+k)]=c[2*nx+j]*p[j+nn*i]*e[i]*invp[i+nn*k]*b[2*nx+k];
                spu[(2*nx+j)+nc*(2*nx+ny+k)]=c[2*nx+j]*p[j+nn*i]*e[i]* invp[i+nn*(nn-
ny+k)]*b[2*nx+ny+k];
            }
        }
    }
```

```
    for(j=0;j<ny;j++)
    {
        for(i=0;i<nn;i++)
        {
            for(k=0;k<nx;k++)
            {
                spu[(2*nx+ny+j)+nc*k]=c[2*nx+ny+j]*p[nn-ny+j+nn*i]*e[i]* invp[i
+nn*ny*k]*b[k];
                spu[(2*nx+ny+j)+nc*(k+nx)]=c[2*nx+ny+j]*p[nn-ny+j+nn*i]*e[i]* invp[i
+nn*(ny*(1+k)-1)]*b[nx+k];
            }
            for(k=0;k<ny;k++)
            {
                spu[(2*nx+ny+j)+nc*(2*nx+k)]=c[2*nx+ny+j]*p[nn-ny+j+nn*i]*e[i]* invp[i+
nn*k]*b[2*nx+k];
                spu[(2*nx+ny+j)+nc*(2*nx+ny+k)]=c[2*nx+ny+j]*p[nn-ny+j+nn*i]*e[i]* invp[i+
nn*(nn-ny+k)]*b[2*nx+ny+k];
            }
        }
    }

  //MATRIZ SPG
    for(i=0;i<nn;i++)
    {
        for(k=0;k<nn;k++)
        {
            for(j=0;j<nx;j++)
            {
                spg[j+nc*k]=c[j]*p[ny*j+nn*i]*h[i]*invp[i+nn*k];
                spg[(nx+j)+nc*k]=c[nx+j]*p[ny*(j+1)-1+nn*i]*h[i]*invp[i+nn*k];
            }
            for(j=0;j<ny;j++)
            {
                spg[(2*nx+j)+nc*k]=c[2*nx+j]*p[j+nn*i]*h[i]*invp[i+nn*k];
                spg[(2*nx+j+ny)+nc*k]=c[2*nx+ny+j]*p[nn-ny+j+nn*i]*h[i]*invp[i+nn*k];
            }
        }
    }
    //fin vp
}

////////////////////////////////////////////////////////////////////////////////////////////
//      Construcción de la matriz tnNod, las temperaturas nodales, dando como entrada
//      el instante de tiempo t y como salida los valores de tnNod por referencia.
```

```
//      tnNod[tiempo+nt*num de nodo].
////////////////////////////////////////////////////////////////////////////////////////////////////

void Subestructura::calcularTnNod(int t)
{
    int m, k;
    for(k=0;k<nn;k++)
    {
        tnNod[t+nt*k] = float(0);
        for(m=0;m<nn;m++)
        {
            tnNod[t+nt*k]+=st[k+nn*m]*tnNod[(t-1)+nt*m];
            tnNod[t+nt*k]+=sg[k+nn*m]*gNod_rocp[(t)+nt*m];
        }
        for(m=0;m<nc;m++)
        {
            tnNod[t+nt*k]+=su[k+nn*m]*tcc[(t-1)+nt*m];
        }
        for(m=0;m<nc;m++)
        {
            tnNod[t+nt*k]+=n[k+nn*m]*tcc[(t)+nt*m];
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////
//      Construcción de la matriz vp, matriz de acoplamiento, dando como entrada el
//      instante de tiempo t y como salida los valores de vp por referencia.
////////////////////////////////////////////////////////////////////////////////////////////////////

void Subestructura::calcularVp(int t)
{
    int m,k;
    for(k=0;k<nc;k++)
    {
        vp[k]=float(0);
        for(m=0;m<nn;m++)
        {
            //Temperatura nodal
            vp[k]+=spt[k+nc*m]*tnNod[(t-1)+nt*m];
            //Generacion nodal
            vp[k]+=spg[k+nc*m]*gNod_rocp[(t-1)+nt*m];
```

```
        }
        for(m=0;m<nc;m++)
        {
            vp[k]+=spu[k+nc*m]*tcc[(t-1)+nt*m];
        }
    }
}


////////////////////////////////////////////////////////////////////////////////////////////////
//    Construcción de la matriz q, flujo de calor dando como entrada el instante
//    de tiempo t y como salida los valores de q por referencia.
////////////////////////////////////////////////////////////////////////////////////////////////

void Subestructura::calcularQ(int t)
{
    int m,k;

    for(k=0;k<nc;k++)
    {
        q[t+nt*k]=float(0);
        for(m=0;m<nc;m++)
        {
            q[t+nt*k]+= ma[k+nc*m]*tcc[t+nt*m];
        }

        for(m=0;m<nc;m++)
        {
            q[t+nt*k]+= vp[k];
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////
//     devolver las condiciones de contorno segun la
//     posicion de contorno
//         Abajo        = 1
//         Arriba       = 2
//         Izquierda    = 3
//         Derecha      = 4
////////////////////////////////////////////////////////////////////////////////////////////////

void Subestructura::dameCondContorno(int posicion, float *tnn, float *Rtemp, float *
```

```
coord_nodos,  int t)
{
    int i,k;
    if(posicion==1)
    {
        for(i=0;i<nx;i++)
        {
            for(k=0;k<nx;k++)
            {
                tnn[i] = tnNod[t+nt*ny*k];

                if(kNod[ny*k]!=0)
                {
                    Rtemp[i] = (dyNod[ny*k]) / (2*kNod[ny*k]);
                }
                else
                {
                    Rtemp[i] = (float)100000000000000;
                }

            }
        }
    }
    else if(posicion==2)
    {
        for(i=0;i<nx;i++)
        {
            for(k=0;k<nx;k++)
            {
                tnn[i] = tnNod[t+nt*(ny*k+ny-1)];

                if(kNod[(ny*k+ny-1)]!=0)
                {
                    Rtemp[i] = (dyNod[(ny*k+ny-1)]) / (2*kNod[(ny*k+ny-1)]);
                }
                else
                {
                    Rtemp[i] = (float)100000000000000;
                }
            }
        }
    }
```

```
else if(posicion==3)
{
    for(i=0;i<ny;i++)
    {
        for(k = 0;k < ny;k++)
        {
            tnn[i] = tnNod[t+nt*k];

            if(kNod[k]!=0)
            {
                Rtemp[i] = (dyNod[k]) / (2*kNod[k]);
            }
            else
            {
                Rtemp[i] = (float)100000000000000;
            }
        }
    }
}
else
{
    for(i=0;i<ny;i++)
    {
        for(k=0;k<ny;k++)
        {
            tnn[i] = tnNod[t+nt*(nn-ny+k)];

            if(kNod[(nn-ny+k)]!=0)
            {
                Rtemp[i] = (dyNod[(nn-ny+k)]) / (2*kNod[(nn-ny+k)]);
            }
            else
            {
                Rtemp[i] = (float)100000000000000;
            }
        }
    }
}

posicionNodos(xNod,yNod);
```

```
    if(posicion==1)
    {
        for(k=0;k<nx;k++)
        {
            coord_nodos[k] = xNod[ny*k];
        }
    }
    else if(posicion==2)
    {
        for(k=0;k<nx;k++)
        {
            coord_nodos[k] = xNod[ny*k+ny-1];
        }
    }
    else if(posicion==3)
    {
        for(k=0;k<ny;k++)
        {
            coord_nodos[k] = yNod[(k)];
        }
    }
    else
    {
        for(k=0;k<ny;k++)
        {
            coord_nodos[k] = yNod[nn-ny+k];
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////
//      poner las condiciones de contorno segun la
//      posicion de contorno
//          Abajo        = 1
//          Arriba       = 2
//          Izquierda    = 3
//          Derecha      = 4
////////////////////////////////////////////////////////////////////////////////////////////////

void Subestructura::ponCondContorno(int posicion, float *tnn, float *Rtemp, float *
coord_nodos,int t)
{
    int k;
```

```
if(posicion==1)
{
    for(k=0;k<nx;k++)
    {
        tcontorno[k] = tnn[(k)];
    }
}
else if(posicion==2)
{
    for(k=0;k<nx;k++)
    {
        tcontorno[k+nx] = tnn[(k)];
    }
}
else if(posicion==3)
{
    for(k=0;k<ny;k++)
    {
        tcontorno[k+2*nx] = tnn[(k)];
    }
}
else
{
    for(k=0;k<ny;k++)
    {
        tcontorno[k+2*nx+ny] = tnn[(k)];
    }
}


if(posicion==1)
{
    for(k=0;k<nx;k++)
    {
        rcontorno[k] = Rtemp[k];
    }
}
else if(posicion==2)
{
    for(k=0;k<nx;k++)
    {
        rcontorno[k+nx] = Rtemp[k];
```

```
        }
    }
    else if(posicion==3)
    {
        for(k=0;k<ny;k++)
        {
            rcontorno[k+2*nx] = Rtemp[k];
        }
    }
    else
    {
        for(k=0;k<ny;k++)
        {
            rcontorno[k+2*nx+ny] = Rtemp[k];
        }
    }

    if(posicion==1)
    {
        for(k=0;k<nx;k++)
        {
            scontorno[k] = coord_nodos[k];
        }

    else if(posicion==2)
    {
        for(k=0;k<nx;k++)
        {
            scontorno[k+nx] = coord_nodos[k];
        }

    else if(posicion==3)
    {
        for(k=0;k<ny;k++)
        {
            scontorno[k+2*nx] = coord_nodos[k];
        }

    else
    {
        for(k=0;k<ny;k++)
        {
```

```
            scontorno[k+2*nx+ny] = coord_nodos[k];
        }
    }
}




////////////////////////////////////////////////////////////////////////////////////////////////
//    Funcion destinada a crear un vector dependiente del tiempo tcc para guardar las
//    temperaturas de los contonos incluídos en tcontorno, ya que tcontorno se escribe
//    y borra para cada paso de tiempo
////////////////////////////////////////////////////////////////////////////////////////////////

void Subestructura::igualarTccTcontorno(int t)
{
    int i;
    for(i=0;i<nc;i++)
        tcc[t+nt*i]=tcontorno[i];
}

////////////////////////////////////////////////////////////////////////////////////////////////
//    Finalidad : Calculo de la posicion de los nodos
//    Efecto : rellena los vectores xNod,yNod, usados en dameCondCont
//    Entrada : punteros a los
//                vectores de resultados xNod e yNod.
//    Salida : por referencia, vectores de resultados xNod e yNod.
////////////////////////////////////////////////////////////////////////////////////////////////

void Subestructura::posicionNodos(float  *xNod,float  *yNod)
{
    //Declaraciones
    //Contadores
    int g,j;

    //Calculo de la posicion de los nodos
    //incializacion a (x,y) para todos los nodos

    for(g=0;g<ny;g++)
    {
        for(j=0;j<nx;j++)
```

```
        {
            xNod[(g+ny*j)]=x;
            yNod[(g+ny*j)]=y;
        }
    }
    //calculo de la posicion
    for(g=0;g<ny;g++)
    {
        for(j=0;j<nx;j++)
        {
            //Si es el primer nodo en direccion horizontal
            if(j==0)
            {
            xNod[(g+ny*0)]+=(float).5*dxNod[(g+ny*0)];
            }
            else
            {
                xNod[(g+ny*j)]=xNod[(g+ny*(j-1))]+(float)0.5*dxNod[(g+ny*j)]+
                    (float)0.5*dxNod[(g+ny*(j-1))];
            }
            //Si es el primer nodo en direccion vertical
            if(g==0)
            {
            yNod[(0+ny*j)]+=(float).5*dyNod[(0+ny*j)];
            }
            else
            {
                yNod[(g+ny*j)]=yNod[(g-1+ny*j)]+(float)0.5*dyNod[(g+ny*j)]+
                    (float)0.5*dyNod[(g-1+ny*j)];
            }
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////
//    Función que calcula la temperatura en los nodos donde hay generacion (nodos con
//    velocidad del material distinta de cero).
////////////////////////////////////////////////////////////////////////////////////////////////

void Subestructura::ponGeneracion (float *gen, int t)
{
    int i;
```

```
    int j;
    int nMater;

    for(j=0;j<nx;j++)
    {
        for(i=0;i<ny;i++)
        {
            nMater = númeroMaterial[i][j];
            if(velMat[nMater]!=0)
            {
                gNod[t+nt*(i+ny*j)]=gen[(i+ny*j)];
                gNod_rocp[t+nt*(i+ny*j)]=gNod[t+nt*(i+ny*j)]/(roNod[i+ny*j]*cpNod[i+ny*j]);
            }

            else
            {
                gNod[t+nt*(i+ny*j)] = float(0);
                gNod_rocp[t+nt*(i+ny*j)] = float(0);
            }
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////
//    Funcion que interpola de entre una serie de valores
//    datos s,ts,ns,x,nx
//    resultados tx (debe estar reservado)
////////////////////////////////////////////////////////////////////////////////////////////////////

int Subestructura::interpolador(float *x,float *tx,int nx,float *s,float *ts,int ns)
{
    int i,j;
    for(i=0;i<nx;i++)
    {
        if(x[i]<=s[0])
        {
            if((s[1]-s[0])!=float(0))
            {
                tx[i] = ts[0];
                tx[i] += (x[i]-s[0])*(ts[1]-ts[0])/(s[1]-s[0]);
            }
            else
```

```
            {
                tx[i] = (float)0.5*(ts[0]+ts[1]);
            }
        }
        for(j=0;j<ns;j++)
        {
            if((s[j]<=x[i])&&(x[i]<=s[j+1]))
            {
                if((s[j+1]-s[j])!=float(0))
                {
                    tx[i] = ts[j];
                    tx[i] += (x[i]-s[j])*(ts[j+1]-ts[j])/(s[j+1]-s[j]);
            }
            else
            {
                tx[i] = (float)0.5*(ts[j]+ts[j+1]);
            }
            break;
        }
    }
    if(x[i]>=s[ns-1])
    {
        if((s[ns-1]-s[ns-2])!=float(0))
        {
            tx[i] = ts[ns-2];
            tx[i] += (x[i]-s[ns-2])*(ts[ns-1]-ts[ns-2])/(s[ns-1]-s[ns-2]);
        }
        else
        {
            tx[i] = float(0.5)*(ts[ns-2]+ts[ns-1]);
        }
        break;
    }
    return int(0);
}

////////////////////////////////////////////////////////////////////////////////////////////////////
//    Funcion que iguala las temperaturas de todos los nodos tnNod a un vector auxiliar
//    temp, para pasarlo al main. Una vez allí, será el vector temp el que se escriba
//    en los archivos de temperaturas nodales.
////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
void Subestructura::devolverTnNod(float *temp,int nt,int nn)
{
    int i, t;
    for(t=0;t<nt;t++)
    {
        for(i=0;i<nn;i++)
        {
            temp[t+nt*(i)] = tnNod[t+nt*(i)];
        }
    }
}




////////////////////////////////////////////////////////////////////////////////////////////////////
//    Funcion que pone indices a las subestructuras segun su ubicacion dentro del sistema
//    posicion:    1 = abajo,
//                 2 = arriba
//                 3 = izda.
//                 4 = dcha.
////////////////////////////////////////////////////////////////////////////////////////////////////

void Subestructura::ponIndice(int posicion, int indiceDat)
{
    if(posicion<1||posicion>4)
    {
        printf("\nLa posicion debe estar entre 1 y 4.");
        exit(78);
    }
    indice[posicion-1]=indiceDat;
}

////////////////////////////////////////////////////////////////////////////////////////////////////
//    Funcion que pide indices a las subestructuras segun su ubicacion dentro del sistema
//    posicion:    1 = abajo,
//                 2 = arriba
//                 3 = izda.
//                 4 = dcha.
////////////////////////////////////////////////////////////////////////////////////////////////////

void Subestructura::dameIndice(int posicion, int &indiceDat)
```

```
{
    if(posicion<1||posicion>4)
    {
        printf("\nLa posicion debe estar entre 1 y 4.");
        exit(78);
    }
    indiceDat=indice[posicion-1];
}

////////////////////////////////////////////////////////////////////////////////////////////////
//    Función destinada a escribir un vector que nos proporcione las temperaturas de
//    contorno reales, no las temperaturas de contorno equivalentes (son las utilizadas
//    para calcular las temperaturas nodales). Por ello se llama tcc_real.
////////////////////////////////////////////////////////////////////////////////////////////////

void Subestructura::resolverTcc(float *tcc_real,int t)
{
    int i,j;

    for(i=0;i<nx;i++)
    {
        if(tcc[t+nt*i]==0&&rcontorno[i]!=0)
        {
            tcc_real[t+nt*i] = (((dx*kNod[i])/(2*rcontorno[i]))*tcc[t+nt*i]+ tnNod[t+
nt*(ny*i)])/(1+(dx*kNod[i])/(2*rcontorno[i]));
        }
        else
        {
            tcc_real[t+nt*i] = tcc[t+nt*i];
        }
    }

    for(i=0;i<nx;i++)
    {
        if(tcc[t+nt*(i+nx)]==0&&rcontorno[(i+nx)]!=0)
        {
            tcc_real[t+nt*(i+nx)] = (((dx*kNod[(i+nx)])/(2*rcontorno[(i+nx)]))* tcc[t+ nt*(i
+nx)]+tnNod[t+nt*((ny*(i+1))-1)])/(1+(dx*kNod[(i+nx)])/ 2*rcontorno[(i+nx)]));
        }
        else
        {
            tcc_real[t+nt*(i+nx)] = tcc[t+nt*(i+nx)];
        }
```

```
    }

    for(j=0;j<ny;j++)
    {
        if(tcc[t+nt*(j+2*nx)]==0&&rcontorno[(j+2*nx)]!=0)
        {
            tcc_real[t+nt*(j+2*nx)] = (((dx*kNod[(j+2*nx)])/ (2*rcontorno[(j+ 2*nx)]))
*tcc[t+nt*(j+2*nx)]+tnNod[t+nt*j])/ (1+(dx*kNod[(j+2*nx)])/(2*rcontorno[(j+2*nx)]));
        }
        else
        {
            tcc_real[t+nt*(j+2*nx)] = tcc[t+nt*(j+2*nx)];
        }
    }

    for(j=0;j<ny;j++)
    {
        if(tcc[t+nt*(j+2*nx+ny)]==0&&rcontorno[(j+2*nx+ny)]!=0)
        {
            tcc_real[t+nt*(j+2*nx+ny)] = (((dx*kNod[(j+2*nx+ny)])/ (2*rcontorno[(j+
2*nx+ny)]))*tcc[t+nt*(j+2*nx+ny)]+ tnNod[t+nt*(ny*(nx-1)+j)])/(1+(dx*kNod[(j+2*nx+ny)])/
(2*rcontorno[(j+2*nx+ny)]));
        }
        else
        {
            tcc_real[t+nt*(j+2*nx+ny)] = tcc[t+nt*(j+2*nx+ny)];
        }
    }

}

////////////////////////////////////////////////////////////////////////////////////////////////
//     Funcion que pone en el nodo de generacion la temperatura de dicha generacion
////////////////////////////////////////////////////////////////////////////////////////////////

void Subestructura::ponTGeneracion (float *gen, int t)
{
    int i;
    int j;
    int nMat;

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

```
        {
            nMat = númeroMaterial[i][j];
            if(velMat[nMat]!=0)
            {
                tnNod[t+nt*(i+ny*j)]=gen[(i+ny*j)];
            }
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////
//     Funcion para llevar al main los flujos de calor de cada contorno, para poder
//     escribirlos por fichero.
////////////////////////////////////////////////////////////////////////////////////////////////

void Subestructura::devolverq(float *fluq)
{
    for(int t=0;t<nt;t++)
    {
        for( int k=0;k<nc;k++)
        {
            fluq[t+nt*k] = q[t+nt*k];
        }
    }
}
```

Código de ***main.cpp***

```
#include "stdio.h"
#include "stdlib.h"
#include "math.h"
#include "subestructura.h"
#include "string.h"
#include "assert.h"

void main(void)
{
    //Punteros a utilizar.
    int t,i,j,u,aux,k,p;
```

```
//Punteros de ficheros
FILE *pf1,*pf2,*pf3,*pf4,*pf5,*pf6,*pf, *pff;
////////////////////////////////////////////////////////////////////////////////////////////
//número de subestructuras, a leer del fichero6.txt
int ns;

//Datos de tiempo, a leer del archivo6.txt
int ntDat;
float deltatDat;

//El ficrero6.txt lleva informacion del número de subestructuras y de variables de tiempo
char * nombreFicheroTiempo;
nombreFicheroTiempo = new char[200];
strcpy(nombreFicheroTiempo,"fichero6.txt");
pf6 = fopen(nombreFicheroTiempo,"r");
fscanf(pf6,"%d",&ns);
fscanf(pf6,"%ld",&ntDat);
fclose(pf6);

////////////////////////////////////////////////////////////////////////////////////////////
//Número de material (una matriz para cada subestructura)
int ***númeroMaterialDat;
númeroMaterialDat = new int ** [ns];

//Número de nodos, uno para cada subestructura
int *nxDat;
int *nyDat;
//Longitud de los lados.
float *lxDat,*lyDat;
//dx=lxDat/nxDat;
float *dx,*dy;
//Reserva de memoria para cada subestructura
nxDat = new int [ns];
nyDat = new int [ns];
lxDat = new float [ns];
lyDat = new float [ns];
dx = new float [ns];
dy = new float [ns];

//Vector Flotante para las temperaturas iniciales de las subestructuras
float *tiniDat;
```

```
    tiniDat = new float [ns];
    for(i=0;i<ns;i++)
        tiniDat[i]=float(0);

    //Los ficheros que nos dan la informacion de cada subestructura
    //se van a llamar fichero1_0.txt, fichero1_1.txt, etc...
    char ** nombreFicheroNodos;
    nombreFicheroNodos = new char *[ns];
    char *caena;
    caena = new char [7];
    for(i=0;i<ns;i++)
    {
        nombreFicheroNodos[i] = new char[200];
        strcpy(nombreFicheroNodos[i],"fichero1");
        sprintf(caena,"_%d.txt",i);
        strcat(nombreFicheroNodos[i],caena);
    }

    for(u=0;u<ns;u++)
    {
        pf1 = fopen(nombreFicheroNodos[u],"r");
        //Leer los datos de los nodos y paso de tiempo
        fscanf(pf1,"%ld",&nxDat[u]);
        fscanf(pf1,"%ld",&nyDat[u]);
        fscanf(pf1,"%f",&lxDat[u]);
        fscanf(pf1,"%f",&lyDat[u]);
        fscanf(pf1,"%f",&tiniDat[u]);

        númeroMaterialDat[u] = new int *[nyDat[u]];
        for(i=0;i<nyDat[u];i++)
            númeroMaterialDat[u][i] = new int [nxDat[u]];
        for(i=0;i<nyDat[u];i++)
            for(j=0;j<nxDat[u];j++)
                fscanf(pf1,"%d %d %d",&aux,&aux,&(númeroMaterialDat[u][i][j]));
        fclose(pf1);
    }

    float sumax;
    sumax=0;
    float sumay;
    sumay=0;
    float sumaxy;
```

```
    for(i=0;i<ns;i++)
    {
        dx[i] = lxDat[i]/nxDat[i];
        dy[i] = lyDat[i]/nyDat[i];
        sumax += dx[i]/ns;
        sumay += dy[i]/ns;
    }
    sumaxy=(sumax+sumay)/2;




////////////////////////////////////////////////////////////////////////////////////////////
    //número de materiales
    int nMatDat;

    //Materiales
    float *kMatDat;
    float *roMatDat;
    float *cpMatDat;
    float *velMatDat;

    //El fichero2.txt llevara informacion generica de los tipos de materiales.
    char * nombreFicheroMateriales;
    nombreFicheroMateriales = new char[200];
    strcpy(nombreFicheroMateriales,"fichero2.txt");
    pf2 = fopen(nombreFicheroMateriales,"r");
    fscanf(pf2,"%d",&nMatDat);
    kMatDat = new float [nMatDat];
    roMatDat = new float [nMatDat];
    cpMatDat = new float [nMatDat];
    velMatDat = new float [nMatDat];
    for(i=0;i<nMatDat;i++)
        fscanf(pf2,"%f %f %f
%f",&(kMatDat[i]),&(roMatDat[i]),&(cpMatDat[i]),&(velMatDat[i]));
    fclose(pf2);

    float sumak;
    sumak=0;
    float sumaro;
    sumaro=0;
    float sumacp;
    sumacp=0;
```

```
    for(i=0;i<nMatDat;i++)
    {
        sumak+=kMatDat[i]/nMatDat;
        sumaro+=roMatDat[i]/nMatDat;
        sumacp+=cpMatDat[i]/nMatDat;
    }

    float alfa;
    alfa=sumak/(sumaro*sumacp);
    //deltat es el tiempo que se pone al resolver la integral de Duhamel
deltatDat= (sumaxy*sumaxy)/alfa;


////////////////////////////////////////////////////////////////////////////////////////
    //El archivo fichero4.txt tiene: ntDat, deltat, número de tipos de contorno
    //(númeroTipoContorno) y todos los tipos de contorno de la forma siguiente:
    // primero = indice de tipo de contorno. IMPORTANTE el primer tipo es el uno, NO el cero
    //     segundo y tercero = temperatura del contorno de la forma T=a+b*t.
    //     cuarto = resistencia del contorno.
    //Solo se cogeran y guardaran los tres últimos en la matriz tipoContorno
    //tipoContorno[1][1] será el coeficiente "a" del tipo 1 de contorno.
    //tipoContorno[1][2] será el coeficiente "b" del tipo 1 de contorno.
    //tipoContorno[1][3] será el rc del tipo 1 de contorno.
    //tipoContorno[2][1] será el coeficiente "a" del tipo 2 de contorno.....

    char * nombreFicheroBD;
    nombreFicheroBD = new char [200];
    strcpy(nombreFicheroBD,"fichero4.txt");
    float **tipoContorno;
    int númeroTipoContorno;
    float auxiliar;

    pf4 = fopen(nombreFicheroBD,"r");
    fscanf(pf4,"%d",&númeroTipoContorno);
    tipoContorno = new float *[númeroTipoContorno];
    for(i=1;i<númeroTipoContorno+1;i++)
    {
        tipoContorno[i] = new float[3];
        for (j=1;j<4;j++)
        {
            tipoContorno[i][j] = float(0.0);
        }
        fscanf(pf4, "%f %f %f %f", &auxiliar, &tipoContorno[i][1],&tipoContorno[i][2],
```

```
            &tipoContorno[i][3]);
    }
    fclose (pf4);

    ////////////////////////////////////////////////////////////////////////////////
    //Leemos la generacion desde fichero5.txt. Suponemos que sólo hay una loncha de
espesor.
    //Por tanto, solo se utiliza ponGeneracion, y toda la generacion se coge
    //del fichero de generacion.También supondremos que en cada subestructura,
    //la generación es la misma para toda la subestructura.
    //El fichero de generacion contendrá una generacion constante
    //o lineal con el tiempo, de la forma gen=a+bt. Si es constante, b=0.
    //IMPORTANTE: en el fichero5 se ordenaran los a's y b's desde la subestructura
    //cero a la ultima siguiendo un orden consecutivo.
    float *a,*b;
    float **genDat;
    a=new float[ns];
    b=new float[ns];
    char* FicheroGeneracion;
    FicheroGeneracion="fichero5.txt";
    genDat = new float*[ns];
    for(i=0;i<ns;i++)
    {
        genDat[i] = new float[nxDat[i]*nyDat[i]];
    }
    pf5 = fopen(FicheroGeneracion,"r");
    for(i=0;i<ns;i++)
    {
        fscanf(pf5,"%f %f",&a[i],&b[i]);
    }
    fclose (pf5);


////////////////////////////////////////////////////////////////////////////////
    //Preparamos la lectura de las condiciones de contorno
    //Las condiciones de contorno dependen de la subestructura
    //por tanto estaran incluidas en el bucle principal.
    //Vamos a crear ns archivos, uno por cada subestructura.
    //El archivo fichero3_0.txt será el archivo de las cond de cont de la subestructura 0.
    //El archivo fichero3_1.txt será el archivo de las cond de cont de la subestructura 1...

    char ** nombreFicheroContorno;
    nombreFicheroContorno = new char *[ns];
```

```
char *cadena;
cadena = new char [7];
for(i=0;i<ns;i++)
{
    nombreFicheroContorno[i] = new char[200];
    strcpy(nombreFicheroContorno[i],"fichero3");
    sprintf(cadena,"_%d.txt",i);
    strcat(nombreFicheroContorno[i],cadena);
}

//Declaracion y reserva de memoria de las condiciones de contorno
float ***tcc1;
float ***tcc2;
float ***tcc3;
float ***tcc4;
float **rcc1;
float **rcc2;
float **rcc3;
float **rcc4;
float **xcc1;
float **xcc2;
float **xcc3;
float **xcc4;

tcc1 = new float **[ns];
tcc2 = new float **[ns];
tcc3 = new float **[ns];
tcc4 = new float **[ns];
rcc1 = new float *[ns];
rcc2 = new float *[ns];
rcc3 = new float *[ns];
rcc4 = new float *[ns];
xcc1 = new float *[ns];
xcc2 = new float *[ns];
xcc3 = new float *[ns];
xcc4 = new float *[ns];

for(i=0;i<ns;i++)
{
    tcc1[i] = new float *[ntDat];
    tcc2[i] = new float *[ntDat];
    tcc3[i] = new float *[ntDat];
```

```
        tcc4[i] = new float *[ntDat];
        rcc1[i] = new float [nxDat[i]];
        rcc2[i] = new float [nxDat[i]];
        rcc3[i] = new float [nyDat[i]];
        rcc4[i] = new float [nyDat[i]];
        xcc1[i] = new float [nxDat[i]];
        xcc2[i] = new float [nxDat[i]];
        xcc3[i] = new float [nyDat[i]];
        xcc4[i] = new float [nyDat[i]];

        for(j=1;j<ntDat;j++)
        {
            tcc1[i][j] = new float [nxDat[i]];
            tcc2[i][j] = new float [nxDat[i]];
            tcc3[i][j] = new float [nyDat[i]];
            tcc4[i][j] = new float [nyDat[i]];

            for(p=0;p<nxDat[i];p++)
            {
                tcc1[i][j][p] = float(0);
                tcc2[i][j][p] = float(0);
                rcc1[i][p] = float(0);
                rcc2[i][p] = float(0);
                xcc1[i][p] = float(0);
                xcc2[i][p] = float(0);
            }
            for(p=0;p<nyDat[i];p++)
            {
                tcc3[i][j][p] = float(0);
                tcc4[i][j][p] = float(0);
                rcc3[i][p] = float(0);
                rcc4[i][p] = float(0);
                xcc3[i][p] = float(0);
                xcc4[i][p] = float(0);
            }
        }
    }


////////////////////////////////////////////////////////////////////////////////////////////
    //Preparamos los archivos de resultados de temperaturas nodales, de contorno y flujos.
    //Los resultados dependen de la subestructura y del paso de tiempo,
    //por tanto estaran incluidas en el bucle principal.
```

```
    //Vamos a crear ns*ntDat archivos, uno por cada subestructura y por cada paso de
tiempo.
    //El archivo resultad_0_1.txt será:
    //el archivo de los resultados de la subestructura 0 en el paso de tiempo 1.
    //El archivo resultad_3_5.txt será:
    //el archivo de los resultados de la subestructura 3 en el paso de tiempo 5.

    char ** resultadosNodos;
    resultadosNodos = new char *[ns];
    char *cadenab;
    cadenab = new char [7];
    for(i=0;i<ns;i++)
    {
        resultadosNodos[i] = new char[200];
    }
    for(i=0;i<ns;i++)
    {
        strcpy(resultadosNodos[i],"resultad");
        sprintf(cadenab,"_%d.txt",i);
        strcat(resultadosNodos[i],cadenab);
    }

    char ** resultadosContornos;
    resultadosContornos = new char *[ns];
    char *cadenac;
    cadenac = new char [7];
    for(i=0;i<ns;i++)
    {
        resultadosContornos[i] = new char[200];
    }
    for(i=0;i<ns;i++)
    {
        strcpy(resultadosContornos[i],"resultadCont");
        sprintf(cadenac,"_%d.txt",i);
        strcat(resultadosContornos[i],cadenac);
    }

    char ** resultadosFlujos;
    resultadosFlujos = new char *[ns];
    char *cadenad;
    cadenad = new char [7];
    for(i=0;i<ns;i++)
    {
```

```
        resultadosFlujos[i] = new char[200];
    }
    for(i=0;i<ns;i++)
    {
        strcpy(resultadosFlujos[i],"resultadFlujos");
        sprintf(cadenad,"_%d.txt",i);
        strcat(resultadosFlujos[i],cadenad);
    }



////////////////////////////////////////////////////////////////////////////////////////////
    //Vectores auxiliares para escribir las temperaturas de los nodos, las temperaturas de
    //los contornos, y los flujos por cada contorno.
    float **temp;
    temp = new float *[ns];
    for(i=0;i<ns;i++)
        temp[i] = new float[nxDat[i]*nyDat[i]*ntDat];
    float **tcc_real;
    tcc_real = new float *[ns];
    for(i=0;i<ns;i++)
        tcc_real[i] = new float[(2*nxDat[i]+2*nyDat[i])*ntDat];
    float **fluq;
    fluq = new float *[ns];
    for(i=0;i<ns;i++)
        fluq[i] = new float[(2*nxDat[i]+2*nyDat[i])*ntDat];



////////////////////////////////////////////////////////////////////////////////////////////
    //Declaracion de variables entre las cuales están las clases
    Subestructura **s;
    s=new Subestructura *[ns];
    //Crear las subestructuras.
    for(i=0; i<ns;i++)
    {
        s[i] = new Subestructura(nMatDat,tiniDat[i],kMatDat,roMatDat,cpMatDat,velMatDat,
            númeroMaterialDat[i],nxDat[i],nyDat[i],ntDat,deltatDat,lxDat[i],lyDat[i]);
    }
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
    //Borramos memoria dinamica hasta ahora.
    for(i=0;i<ns;i++)
        delete nombreFicheroNodos[i];
    delete[] nombreFicheroNodos;
    delete[] nombreFicheroMateriales;
    delete[] nombreFicheroTiempo;
    delete[] tiniDat;
    delete[] númeroMaterialDat;
    delete[] lxDat;
    delete[] lyDat;
    delete[] kMatDat;
    delete[] roMatDat;
    delete[] cpMatDat;
    delete[] velMatDat;
    delete[] nombreFicheroBD;

////////////////////////////////////////////////////////////////////////////////////////////////




////////////////////////////////////////////////////////////////////////////////////////////////
    //Comenzamos bucle de tiempo y de subestructura.

    //Comenzamos en t=1 porque en 0 las temperaturas son las tiniDat.
    for(t=1;t<ntDat;t++)
    {
        for(i=0; i<ns;i++)
        {
            //Leer las condiciones de contorno del fichero3_i.txt
            //van a almacenar 4 filas:cada fila se corresponde con una condicion
            //de contorno en el siguiente orden: abajo, arriba, izqda, dcha.
            //Y el número sera la cond de contorno segun el criterio siguiente:
            //     menor o igual a cero es el indice de la subestructura colindante
            //     mayor o igual a uno es el tipo de la condicion de contorno exterior almacenada
            //    en la base de datos de cond contorno (fichero 4)

            pf3 = fopen(nombreFicheroContorno[i],"r");

            int *indiceDat;
            indiceDat = new int[4];
```

```
//"indiceDat" va de cero a tres; almacena la informacion codificada de
//la cond contorno. "indice" por contra va de 1 a 4 de forma que cuando
//lo pongamos o lo pidamos habra que poner un número entre 1 y 4.
for(k=0; k<4;k++)
{
    indiceDat[k] = int (0);
    fscanf(pf3,"%d",&(indiceDat[k]));
}
rewind( pf3 );
fclose( pf3 );

for(k=1; k<5;k++)
{
    s[i]->ponIndice(k,indiceDat[k-1]);
}

//Comenzamos a asignar valores a tc, rc y xc.de acuerdo a
//las cond contorno que se hayan leido
for(int kk=0;kk<4;kk++)
{
    s[i]->dameIndice(kk+1,indiceDat[kk]);

    //si es de temperatura o flujo (>=1)
    if (indiceDat[kk] > 0)
    {
        if (kk == 0)
        {
            //Abajo
            for(k=0;k<nxDat[i];k++)
            {
                tcc1[i][t][k] = float(tipoContorno[indiceDat[0]][1]+
                    tipoContorno[indiceDat[0]][2]*t);
                rcc1[i][k] = float(tipoContorno[indiceDat[0]][3]);
                xcc1[i][k] =  k*dx[i];
            }
            s[i]->ponCondContorno(1,tcc1[i][t],rcc1[i],xcc1[i],t);
        }

        else if (kk == 1)
        {
            //Arriba
            for(k=0;k<nxDat[i];k++)
```

```
                        {
                            tcc2[i][t][k] = float(tipoContorno[indiceDat[1]][1]+ tipoContorno
[indiceDat[1]][2]*t);
                            rcc2[i][k] = float(tipoContorno[indiceDat[1]][3]);
                            xcc2[i][k] =  k*dx[i];
                        }
                        s[i]->ponCondContorno(2,tcc2[i][t],rcc2[i],xcc2[i],t);
                    }

                    else if (kk == 2)
                    {
                        //Izda
                        for(k=0;k<nyDat[i];k++)
                        {
                            tcc3[i][t][k] = float(tipoContorno[indiceDat[2]][1]+ tipoContorno
[indiceDat[2]][2]*t);
                            rcc3[i][k] = float(tipoContorno[indiceDat[2]][3]);
                            xcc3[i][k] = k*dy[i];
                        }
                        s[i]->ponCondContorno(3,tcc3[i][t],rcc3[i],xcc3[i],t);
                    }

                    else //kk == 3
                    {
                        //Dcha
                        for(k=0;k<nyDat[i];k++)
                        {
                            tcc4[i][t][k] = float(tipoContorno[indiceDat[3]][1]+ [indiceDat[3]][2]*t);
                            rcc4[i][k] = float(tipoContorno[indiceDat[3]][3]);
                            xcc4[i][k] =  k*dy[i];
                        }
                        s[i]->ponCondContorno(4,tcc4[i][t],rcc4[i],xcc4[i],t);
                    }
                } // se cierra el bucle if de temperaturas o flujos en el contorno

                //condicion de contorno de contacto (<=0)
                else
                {
                    // nos aseguramos que es menor o igual de cero
                    assert (indiceDat[kk]<1);

                    //toca con la otra subestructura por abajo
                    if(kk == 0)
```

```
                {
                    p=abs(indiceDat[kk]);
                    //primer paso PEDIR COND CONTORNO
                    if(t==1)
                    {
                        s[i]->dameCondContorno(1,tcc2[p][t],rcc2[p],xcc2[p],0);
                    }
                    else
                    {
                        s[i]->dameCondContorno(1,tcc2[p][t],rcc2[p],xcc2[p],t-1);
                    }

                    //segundo paso INTERPOLAR
                    s[i]->interpolador(xcc1[i],tcc1[i][t],nxDat[i],
                        xcc2[abs(indiceDat[kk])],tcc2[abs(indiceDat[kk])][t], [abs(indiceDat
[kk])]);

                    //tercer paso PONER COND CONTORNO
                    s[i]->ponCondContorno(1,tcc1[i][t],rcc1[i],xcc1[i],t);
                }

                //toca con la otra subestructura por arriba
                else if (kk == 1)
                {
                    p=abs(indiceDat[kk]);
                    //primer paso PEDIR COND CONTORNO
                    if(t==1)
                    {
                        s[i]->dameCondContorno(2,tcc1[p][t],rcc1[p],xcc1[p],0);
                    }
                    else
                    {
                        s[i]->dameCondContorno(2,tcc1[p][t],rcc1[p],xcc1[p],t-1);
                    }

                    //segundo paso INTERPOLAR
                    s[i]->interpolador(xcc2[i],tcc2[i][t],nxDat[i],
                        xcc1[abs(indiceDat[kk])],tcc1[abs(indiceDat[kk])][t], nxDat[abs
(indiceDat[kk])]);

                    //tercer paso PONER COND CONTORNO
                    s[i]->ponCondContorno(2,tcc2[i][t],rcc2[i],xcc2[i],t);
                }
```

```
                    //toca con la otra subestructura por izqda
                    else if (kk == 2)
                    {
                        p=abs(indiceDat[kk]);
                        //primer paso PEDIR COND CONTORNO
                        if(t==1)
                        {
                            s[i]->dameCondContorno(3,tcc4[p][t],rcc4[p],xcc4[p],0);
                        }
                        else
                        {
                            s[i]->dameCondContorno(3,tcc4[p][t],rcc4[p],xcc4[p],t-1);
                        }

                        //segundo paso INTERPOLAR
                        s[i]->interpolador(xcc3[i],tcc3[i][t],nyDat[i],
                            xcc4[abs(indiceDat[kk])],tcc4[abs(indiceDat[kk])][t], nyDat
[abs(indiceDat[kk])]);

                        //tercer paso PONER COND CONTORNO
                        s[i]->ponCondContorno(3,tcc3[i][t],rcc3[i],xcc3[i],t);
                    }

                    //kk==3 toca con la otra subestructura por derecha
                    else
                    {
                        p=abs(indiceDat[kk]);
                        //primer paso PEDIR COND CONTORNO
                        if(t==1)
                        {
                            s[i]->dameCondContorno(4,tcc3[p][t],rcc3[p],xcc3[p],0);
                        }
                        else
                        {
                            s[i]->dameCondContorno(4,tcc3[p][t],rcc3[p],xcc3[p],t-1);
                        }

                        //segundo paso INTERPOLAR
                        s[i]->interpolador(xcc4[i],tcc4[i][t],nyDat[i], xcc3[abs(indiceDat
[kk])],tcc3[abs(indiceDat[kk])][t], nyDat[abs(indiceDat[kk])]);

                        //tercer paso PONER COND CONTORNO
```

```
                s[i]->ponCondContorno(4,tcc4[i][t],rcc4[i],xcc4[i],t);
            }
        }
    }//fin del for kk (indice de las cond de contorno)

    //A continuacion se escribe la generacion en el paso de tiempo actual en gendat.
    for(j=0;j<nxDat[i]*nyDat[i];j++)
    {
        // como empieza en t=1 en el primer paso de tiempo la generacion
        //valdra A y se incrementara con B*t en pasos posteriores
        genDat[i][j] = a[i] + b[i] * (t);
    }
    //A continuacion se pone la generacion.
    s[i]->ponGeneracion(genDat[i],t);

    //Crear las matrices de la descripción interna de la subestructura
    s[i]->calcularMatrices2D(deltatDat,t);
    //Devuelve las matrices que dan el campo de temperaturas nodal en un instante en
    //función de los campos de temps de los instantes anteriores y de los contornos.
    s[i]->calcularMatricesTemperaturas();
    //Devuelve las matrices del sistema
    s[i]->calcularMatricesMAVP();

    //igualarTccTcontorno crea un vector Tcc que sí depende del tiempo, mientras que
    //Tcontorno no depende del tiempo, sólo ha sido cogido de los archivos de texto.
    if ( t == 1)
        {
            //para el primer paso de tiempo se toma t=0 porque en
            //t=1 no tenemos una estimacion de las temperaturas.
            s[i]->igualarTccTcontorno(0);
        }
    else
        {
            s[i]->igualarTccTcontorno(t);
        }

    //Calculamos las temperaturas Nodales.
    s[i]->calcularTnNod(t);
    s[i]->ponTGeneracion(genDat[i],t);
    //no se si falta aqui la siguiente funcion
    s[i]->resolverTcc(tcc_real[i],t);
```

```
            //Calculo final de las temperaturas y los flujos de calor.
            s[i]->calcularVp(t);
            s[i]->calcularQ(t);

            //Devuelve un vector de temperaturas por cada subestructura para poder
            //escribirla en los correspondientes archivos de resultados
            s[i]->devolverTnNod(temp[i],ntDat,nxDat[i]*nyDat[i]);
            s[i]->devolverq(fluq[i]);

        }//fin del ns
    }//fin del ntDat

    ////////////////////////////////////////////////////////////////////////////////////////
    //Escribir en archivo de soluciones las temperaturas nodales

        for(i=0;i<ns;i++)
        {
            pf = fopen(resultadosNodos[i],"w");
            fprintf(pf,"\nTemperaturas Nodales");
            fprintf(pf,"\nNúmerosubest = %d",i);
            fprintf(pf,"\n");
            for(t=1;t<ntDat;t++)
            {
                fprintf(pf,"\ntDat = %d",t);
                fprintf(pf,"\n");

                for(k=0;k<nyDat[i];k++)
                {
                    fprintf(pf,"\n");
                    for(j=0;j<nxDat[i];j++)
                    {
                        fprintf(pf," %f",temp[i][t+ntDat*(k+nyDat[i]*j)]);
                    }
                }
                fprintf(pf,"\n");
                fprintf(pf,"\n");
            }

            pff = fopen(resultadosContornos[i],"w");
            fprintf(pff,"\nTemperaturas Contornos");
            fprintf(pff,"\nNúmerosubest = %d",i);
            fprintf(pff,"\n");
```

```
        for(t=1;t<ntDat;t++)
        {
            fprintf(pff,"\ntDat = %d",t);
            fprintf(pff,"\n");
            fprintf(pff,"\nabajo");
            for(j=0;j<nxDat[i];j++)
                fprintf(pff,"\n %d %f",j,tcc_real[i][t+ntDat*j]);
            fprintf(pff,"\narriba");
            for(j=nxDat[i];j<(2*nxDat[i]);j++)
                fprintf(pff,"\n %d %f",j,tcc_real[i][t+ntDat*j]);
            fprintf(pff,"\nizquierda");
            for(k=(2*nxDat[i]);k<(2*nxDat[i]+nyDat[i]);k++)
                fprintf(pff,"\n %d %f",k,tcc_real[i][t+ntDat*k]);
            fprintf(pff,"\nderecha");
            for(k=(2*nxDat[i]+nyDat[i]);k<(2*nxDat[i]+2*nyDat[i]);k++)
                fprintf(pff,"\n %d %f",k,tcc_real[i][t+ntDat*k]);
            fprintf(pff,"\n");
            fprintf(pff,"\n");
        }

        FILE *pff;
        pfff = fopen(resultadosFlujos[i],"w");
        fprintf(pfff,"\nFlujos Contornos");
        fprintf(pfff,"\nNúmerosubest = %d",i);
        fprintf(pfff,"\n");
        for(t=1;t<ntDat;t++)
        {
            fprintf(pfff,"\ntDat = %d",t);
            fprintf(pfff,"\n");
            fprintf(pfff,"\nabajo");
            for(j=0;j<nxDat[i];j++)
                fprintf(pfff,"\n %d %f",j,fluq[i][t+ntDat*j]);
            fprintf(pfff,"\narriba");
            for(j=nxDat[i];j<(2*nxDat[i]);j++)
                fprintf(pfff,"\n %d %f",j,fluq[i][t+ntDat*j]);
            fprintf(pfff,"\nizquierda");
            for(k=(2*nxDat[i]);k<(2*nxDat[i]+nyDat[i]);k++)
                fprintf(pfff,"\n %d %f",k,fluq[i][t+ntDat*k]);
            fprintf(pfff,"\nderecha");
            for(k=(2*nxDat[i]+nyDat[i]);k<(2*nxDat[i]+2*nyDat[i]);k++)
                fprintf(pfff,"\n %d %f",k,fluq[i][t+ntDat*k]);
            fprintf(pfff,"\n");
```

```
            }
            fclose(pfff);
            fclose(pf);
            fclose(pff);
        }


////////////////////////////////////////////////////////////////////////////////////////////
    //Borramos memoria dinamica.
    for(p=0;p<ns;p++)
    {
        delete rcc1[p];
        delete rcc2[p];
        delete rcc3[p];
        delete rcc4[p];
        delete xcc1[p];
        delete xcc2[p];
        delete xcc3[p];
        delete xcc4[p];
        for(t=0;t<ntDat;t++) //24.01.05 cambio t=1 por t=0
        {
            delete tcc1[p][t];
            delete tcc2[p][t];
            delete tcc3[p][t];
            delete tcc4[p][t];
        }
        delete tcc1[p];
        delete tcc2[p];
        delete tcc3[p];
        delete tcc4[p];
        delete nombreFicheroContorno[p];
    }

    delete[] rcc1;
    delete[] rcc2;
    delete[] rcc3;
    delete[] rcc4;

    delete[] xcc1;
    delete[] xcc2;
    delete[] xcc3;
    delete[] xcc4;
```

```
    delete[] dx;
    delete[] dy;

    delete[] tcc1;
    delete[] tcc2;
    delete[] tcc3;
    delete[] tcc4;

    delete[] nombreFicheroContorno;

    for(p=0;p<ns;p++)
    {
        delete s[p];
        delete temp[p];
        delete genDat[p];
        delete tcc_real[p];
        delete fluq[p];
    }

    delete[] s;
    delete[] temp;
    delete[] tcc_real;
    delete[] fluq;
    delete[] nxDat;
    delete[] nyDat;
    delete[] a;
    delete[] b;
    delete[] genDat;

    delete[] resultadosNodos;
    delete[] resultadosContornos;
    delete[] resultadosFlujos;
    delete[] resultadosjuntos;

        delete[] cae;

        delete[] caena;

        delete[] cadenab;

        delete[] cadenac;
```

```
        delete[] cadenad;

        delete[] cadenakk;

        delete[] cadena;

    ////////////////////////////////////////////////////////////////////////////////////////////////
}//fin del main
```

4.2. Código del modulo Cryojab-color

Código de ***cryojab-color.php***

```
<?php
// Función principal del modulo cryojab_color. Recibe como parámetros el tamaño en píxeles
// y la resolución de la imagen
function cryojab_color ($tam_sub,$tam_img_x,$tam_img_y){
    // Se llama a la función que calcula el color dominante
    $colores=recorre_imagen ();

    // Se llama a la función que muestrea los colores
    $imagen=identifica_pixeles ($colores);

    // Se llama a la función que genera la imagen muestreada
    crea_sampled ($imagen,$colores,$tam_sub,$tam_img_x,$tam_img_y);

    // Se llama a la función que genera los ficheros de tipo 1
    crea_ficheros ($imagen,$tam_sub,$tam_img_x,$tam_img_y);
}

// Funcion que recorre la imagen en formato textual para determinar los colores
// seleccionados y el color predominante para realizar una aproximación.
// Requiere la existencia de los ficheros "colores.txt" e "imagen.txt"
function recorre_imagen (){
    // Extraemos los colores seleccionados
    if ($gestor=fopen("colores.txt","r")) {
        $contenido=fgets($gestor);
        sscanf($contenido, "Negro:%d-Blanco:%d-Rojo:%d-Verde:%d-Azul:%d-Morado:%d-
```

```
Amarillo:%d-Celeste:%d", $colores["negro"], $colores["blanco"], $colores["rojo"],
$colores["verde"],$colores["azul"],$colores["morado"],$colores["amarillo"],$colores["celeste"
]);
        fclose($gestor);
    }

    // Definimos el margen de selección de los colores
    $margen=125;

    // Inicializamos el array de contadores de los colores
    $contadores["negro"]=0;
    $contadores["blanco"]=0;
    $contadores["rojo"]=0;
    $contadores["verde"]=0;
    $contadores["azul"]=0;
    $contadores["morado"]=0;
    $contadores["amarillo"]=0;
    $contadores["celeste"]=0;

    if ($gestor=fopen("imagen.txt","r")) {
        $contenido=fgets($gestor);
        while (!feof($gestor)){
            // Extraemos de cada linea las componentes RGB de cada pixel
            sscanf($contenido, "%d,%d: (%d,%d,%d", $pix_x, $pix_y, $comp_R, $comp_G,
$comp_B);

            // Si el color negro esta seleccionado e identificamos el pixel como tal
            // incrementamos el contador
            if ($colores["negro"]!=-1)
                if ($comp_R<$margen and $comp_G<$margen and $comp_B<$margen)
                    $contadores["negro"]++;

            // Si el color blanco esta seleccionado e identificamos el pixel como tal
            // incrementamos el contador
            if ($colores["blanco"]!=-1)
                if ($comp_R>$margen and $comp_G>$margen and $comp_B>$margen)
                    $contadores["blanco"]++;

            // Si el color rojo esta seleccionado e identificamos el pixel como tal
            // incrementamos el contador
```

```
        if ($colores["rojo"]!=-1)
            if ($comp_R>$margen and $comp_G<$margen and $comp_B<$margen)
                $contadores["rojo"]++;

        // Si el color verde esta seleccionado e identificamos el pixel como tal
        // incrementamos el contador
        if ($colores["verde"]!=-1)
            if ($comp_R<$margen and $comp_G>$margen and $comp_B<$margen)
                $contadores["verde"]++;

        // Si el color azul esta seleccionado e identificamos el pixel como tal
        // incrementamos el contador
        if ($colores["azul"]!=-1)
            if ($comp_R<$margen and $comp_G<$margen and $comp_B>$margen)
                $contadores["azul"]++;

        // Si el color morado esta seleccionado e identificamos el pixel como
        // tal incrementamos el contador
        if ($colores["morado"]!=-1)
            if ($comp_R>$margen and $comp_G<$margen and $comp_B>$margen)
                $contadores["morado"]++;

        // Si el color amarillo esta seleccionado e identificamos el pixel como
        // tal incrementamos el contador
        if ($colores["amarillo"]!=-1)
            if ($comp_R>$margen and $comp_G>$margen and $comp_B<$margen)
                $contadores["amarillo"]++;

        // Si el color celeste esta seleccionado e identificamos el pixel como tal
        // incrementamos el contador
        if ($colores["celeste"]!=-1)
            if ($comp_R<$margen and $comp_G>$margen and $comp_B>$margen)
                $contadores["celeste"]++;

        $contenido=fgets($gestor);
    }

    fclose ($gestor);

    // Creamos la entrada color predominante
```

```
$colores["pred"]=-1;
$aux=0;

// Si el negro está seleccionado y aparece más que el color actual, se selecciona
if ($colores["negro"]!=-1 and $aux<$contadores["negro"]){
    $colores["pred"]=$colores["negro"];
    $aux=$contadores["negro"];
}

// Si el blanco está seleccionado y aparece más que el color actual, se selecciona
if ($colores["blanco"]!=-1 and $aux<$contadores["blanco"]){
    $colores["pred"]=$colores["blanco"];
    $aux=$contadores["blanco"];
}

// Si el rojo está seleccionado y aparece más que el color actual, se selecciona
if ($colores["rojo"]!=-1 and $aux<$contadores["rojo"]){
    $colores["pred"]=$colores["rojo"];
    $aux=$contadores["rojo"];
}

// Si el verde está seleccionado y aparece más que el color actual, se selecciona
if ($colores["verde"]!=-1 and $aux<$contadores["verde"]){
    $colores["pred"]=$colores["verde"];
    $aux=$contadores["verde"];
}

// Si el azul está seleccionado y aparece más que el color actual, se selecciona
if ($colores["azul"]!=-1 and $aux<$contadores["azul"]){
    $colores["pred"]=$colores["azul"];
    $aux=$contadores["azul"];
}

// Si el morado está seleccionado y aparece más que el color actual, se selecciona
if ($colores["morado"]!=-1 and $aux<$contadores["morado"]){
    $colores["pred"]=$colores["morado"];
    $aux=$contadores["morado"];
}

// Si el amarillo está seleccionado y aparece más que el color actual, se selecciona
```

```
        if ($colores["amarillo"]!=-1 and $aux<$contadores["amarillo"]){
            $colores["pred"]=$colores["amarillo"];
            $aux=$contadores["amarillo"];
        }

        // Si el celeste está seleccionado y aparece más que el color actual, se selecciona
        if ($colores["celeste"]!=-1 and $aux<$contadores["celeste"]){
            $colores["pred"]=$colores["celeste"];
            $aux=$contadores["celeste"];
        }
    }

    // Se devuelve el array con los colores seleccionados
    return $colores;
}

// Función que identifica los colores seleccionados en los pixeles de la imagen en
// formato textual. Recibe como parametro el array con los colores seleccionados
function identifica_pixeles ($colores){
    // Definimos el margen de identificación de colores
    $margen=125;

    if ($gestor=fopen("imagen.txt","r")) {
        // Avanzamos dos lineas, para empezar con la primera linea que contiene la
        // información que deseamos
        $contenido=fgets($gestor);
        $contenido=fgets($gestor);

        while (!feof($gestor)){
            // Inicializamos la variable que almacenará el color de cada pixel
            $color=-1;

            // Extraemos la información de la linea
            sscanf($contenido, "%d,%d: (%d,%d,%d", $pix_x, $pix_y, $comp_R, $comp_G,
$comp_B);

            // Si el negro está seleccionado y aparece más que el color actual, se selecciona
            if ($colores["negro"]!=-1)
                if ($comp_R<$margen and $comp_G<$margen and comp_B<$margen)
                    $color=$colores["negro"];
```

```
// Si el blanco está seleccionado y aparece más que el color actual, se selecciona
if ($colores["blanco"]!=-1)
    if ($comp_R>$margen and $comp_G>$margen and $comp_B>$margen)
        $color=$colores["blanco"];

// Si el rojo está seleccionado y aparece más que el color actual, se selecciona
if ($colores["rojo"]!=-1)
    if ($comp_R>$margen and $comp_G<$margen and $comp_B<$margen)
        $color=$colores["rojo"];

// Si el verde está seleccionado y aparece más que el color actual, se selecciona
if ($colores["verde"]!=-1)
    if ($comp_R<$margen and $comp_G>$margen and $comp_B<$margen)
        $color=$colores["verde"];

// Si el azul está seleccionado y aparece más que el color actual, se selecciona
if ($colores["azul"]!=-1)
    if ($comp_R<$margen and $comp_G<$margen and $comp_B>$margen)
        $color=$colores["azul"];

// Si el morado está seleccionado y aparece más que el color actual, se selecciona
if ($colores["morado"]!=-1)
    if ($comp_R>$margen and $comp_G<$margen and $comp_B>$margen)
        $color=$colores["morado"];

// Si el amarillo está seleccionado y aparece más que el color actual, se selecciona
if ($colores["amarillo"]!=-1)
    if ($comp_R>$margen and $comp_G>$margen and $comp_B<$margen)
        $color=$colores["amarillo"];

// Si el celeste está seleccionado y aparece más que el color actual, se selecciona
if ($colores["celeste"]!=-1)
    if ($comp_R<$margen and $comp_G>$margen and $comp_B>$margen)
        $color=$colores["celeste"];

// Si no se ha identificado con ningún color, se selecciona el color predominante
if ($color==-1)
    $color=$colores["pred"];
```

```
            // Se almacena en el array imagen de dos dimensiones
            $imagen[$pix_x][$pix_y]=$color;

            $contenido=fgets($gestor);
        }

        fclose ($gestor);
    }

    // Se devuelve el array con la imagen almacenada
    return $imagen;
}

// Función que reduce las coordenadas de un pixel a una subestructura. Recibe
// como parámetro la coordenada y el tamaño de la subestructura
function acota_pixel ($pix, $tam_subs){
     $result=$pix;

    // Mientras que la coordenada sobrepasa el tamaño de la subetructura, se
    // decrementa en un tamaño igual al tamaño de la subestructura
      while ($result>=$tam_subs)
         $result-=$tam_subs;

      // Se devuelve la coordenada acotada
      return $result;
}

// Función que crea los ficheros de tipo 1. Recibe como parámetros el array con la
// imagen almacenada, el tamaño de la subestructura y la resolución de la imagen
function crea_ficheros ($imagen,$tam_sub,$tam_img_x,$tam_img_y){
    // Calculamos el número de subestructuras en horizontal y en vertical
    $num_subs_x=(int)($tam_img_x/$tam_sub);
    $num_subs_y=(int)($tam_img_y/$tam_sub);

    // Calculamos el número total de subestructuras
    $num_subs_total=$num_subs_x*$num_subs_y;

    // Inicializamos el índice de subestructuras
    $indice_sub=0;
```

```
while($indice_sub<$num_subs_total){
    // Calculamos los topes de los píxeles de cada subestructura
    $tope_x=(($indice_sub%$num_subs_x)+1)*$tam_sub;
    $tope_y=((int)(($indice_sub)/$num_subs_x))*$tam_sub;

    if ($gestor=fopen("fichero1_".$indice_sub.".txt","w+")) {
        // Se inicializa el índice horizontal en función de la subestructura
        $indice_x=$tope_x-$tam_sub;

        while ($indice_x<$tope_x){
            // Se inicializa el índice vertical en función de la subestructura
            $indice_y=$tope_y+$tam_sub-1;

            while ($indice_y>=$tope_y){
                // Se acotan las coordenadas
                $aux_x=acota_pixel($indice_x,$tam_sub);
                $aux_y=acota_pixel($indice_y,$tam_sub);
                $aux_y=$tam_sub-1-$aux_y;

                // Se almacena en un array auxiliar los datos de la subestructura
                $auxiliar[$aux_x][$aux_y]=$imagen[$indice_x][$indice_y];

                $indice_y--;
            }
            $indice_x++;
        }

        // Inicializamos el indice auxiliar para la coordenada horizontal
        $aux_x=0;
        while ($aux_x<$tam_sub){
            // Inicializamos el indice auxiliar para la coordenada vertical
            $aux_y=0;
            while ($aux_y<$tam_sub){
                // Imprimimos la linea en el fichero
                $contenido=$aux_x."\t".$aux_y."\t".$auxiliar[$aux_x][$aux_y]."\n";
                fwrite($gestor, $contenido);

                $aux_y++;
            }
            $aux_x++;
```

```
            }

            fclose ($gestor);
        }

        $indice_sub++;
    }
}

// Función que genera la imagen muestreada
function crea_sampled ($imagen,$colores,$tam_sub,$tam_img_x,$tam_img_y){
    // Calculamos el número de subestructuras en horizontal y en vertical
    $num_subs_x=(int)($tam_img_x/$tam_sub);
    $num_subs_y=(int)($tam_img_y/$tam_sub);

    if ($gestor=fopen("sampled.txt","w+")) {
        // Imprimimos la linea de identificación para ImageMagick
        $contenido="# ImageMagick pixel enumeration:
".($num_subs_x*$tam_sub).",".($num_subs_y*$tam_sub).",255,RGB\n";
        fwrite($gestor,$contenido);

        // Imprimimos caca linea en función del color identificado
        $indice_y=0;
        while ($indice_y<($num_subs_y*$tam_sub)){
            $indice_x=0;
            while ($indice_x<($num_subs_x*$tam_sub)){
                $contenido=$indice_x.",".$indice_y.": ";

                if ($imagen[$indice_x][$indice_y]==$colores["negro"])
                    $contenido.="(  0,  0,  0)  #000000  black\n";

                if ($imagen[$indice_x][$indice_y]==$colores["blanco"])
                    $contenido.="(255,255,255)  #FFFFFF  white\n";

                if ($imagen[$indice_x][$indice_y]==$colores["rojo"])
                    $contenido.="(255,  0,  0)  #FF0000  red\n";

                if ($imagen[$indice_x][$indice_y]==$colores["verde"])
                    $contenido.="(  0,255,  0)  #00FF00  lime\n";
```

```
                if ($imagen[$indice_x][$indice_y]==$colores["azul"])
                    $contenido.="( 0, 0,255) #0000FF blue\n";

                if ($imagen[$indice_x][$indice_y]==$colores["celeste"])
                    $contenido.="( 0,255,255) #00FFFF cyan\n";

                if ($imagen[$indice_x][$indice_y]==$colores["morado"])
                    $contenido.="(255, 0,255) #FF00FF magenta\n";

                if ($imagen[$indice_x][$indice_y]==$colores["amarillo"])
                    $contenido.="(255,255, 0) #FFFF00 yellow\n";

                fwrite($gestor,$contenido);

                $indice_x++;
            }

            $indice_y++;
        }

        fclose($gestor);
    }
}
?>
```

4.3. Código de las funciones de generación de ficheros, tratamiento de la salida y funciones auxiliares

Codigo de ***functions.php***

```
<?php
// Función que genera los ficheros de tipo 0. Recibe como parámetros de entrada el número
// de nodos en horizontal o vertical de una subestructura, el número de subestructuras en
// horizontal y en vertical, el tamaño de la subestructura en horizontal y en vertical, y la
// temperatura inicial de las subestructuras
function fichero0 ($num_nodos, $num_sub_x, $num_sub_y, $tam_sub_x, $tam_sub_y,
```

```
$temp_ini){
    // Se define una variable tope que será el número de subestructuras de la imagen
    $tope=$num_sub_x*$num_sub_y;

    // Se define una variable contador para númerar las subestructuras
    $numerador=0;

    // Mientras el contador de subestructuras no llegue al tope, se generan ficheros
    while ($numerador<$tope){

        // Se define la variable $fichero que almacena el nombre del fichero a crear
        $fichero="kernel/fichero0_".$numerador.".txt";

        // Se crea el fichero y se rellena con los datos necesarios
        if ($gestor=fopen($fichero,"w+")) {
            $contenido=$num_nodos."\t".$num_nodos."\n\n";
            fwrite($gestor, $contenido);

            $contenido=$tam_sub_x."\t".$tam_sub_y."\n\n";
            fwrite($gestor, $contenido);

            $contenido=$temp_ini."\n\n";
            fwrite($gestor, $contenido);

            fclose($gestor);

            // Se incrementa el contador para proseguir con el siguiente fichero
            $numerador++;
        }
    }
}

// Función que se encarga de generar los ficheros de tipo 1. Recibe como parámetros de
// entrada un array con los colores seleccionados en su orden y el tamaño de las
// subestructuras
function fichero1 ($colores, $tam_sub,$tam_img_x,$tam_img_y){
    // Se cambia al directorio kernel
    $directorio=getcwd();
    chdir($directorio."/kernel");
```

```
    // Se genera el fichero "colores.txt" necesario para la ejecución de cryojab_color
    if ($gestor=fopen("colores.txt","w+")) {
        $contenido="Negro:".$colores["negro"]."-Blanco:".$colores["blanco"]."-
Rojo:".$colores["rojo"]."-Verde:".$colores["verde"]."-Azul:".$colores["azul"]."-
Morado:".$colores["morado"]."-Amarillo:".$colores["amarillo"]."-
Celeste:".$colores["celeste"]."
";
        fwrite($gestor, $contenido);
        fclose($gestor);

        // Se llama a la funcion cryojab_color que genera los ficheros tipo 1 y la imagen
        // muestreada
        $aux_x=$tam_img_x--;
        $aux_y=$tam_img_y--;
        cryojab_color ($tam_sub,$aux_x,$aux_y);

        // Se convierte el fichero con la imagen muestreada a PNG mediante ImageMagick
        $ejecucion="convert sampled.txt sampled.png";
        exec ($ejecucion);

        unlink ("sampled.txt");
    }

    // Se vuelve al directorio de trabajo original
    chdir($directorio);
}

// Función que se encarga de generar los ficheros de tipo 2. Recibe como parámetros de
// entrada un array con los datos de los materiales y el número de materiales
function fichero2 ($datos, $num_mat){
    if ($gestor=fopen("kernel/fichero2.txt","w+")) {
        $contenido=$num_mat."\n\n";
        fwrite($gestor, $contenido);

        // Se recorre el array de datos. Para cada material se tienen 4 campos, por lo que
        // se define un índice que cuando llegue a cuatro, añadirá un salto de linea
        $aux=0;
        foreach ($_POST as $indice => $valor){
            $aux++;
            // Si aún no se ha llegado al cuarto campo y no es el submit, se imprime el dato
```

```
            if ($aux<4 and $valor!="Siguiente"){
                fwrite($gestor,$valor);
                fwrite ($gestor,"\t");
            }

            // Si es el cuarto campo, se comprueba si es un checkbox mirando si tiene la @.
            // Si lo es, se imprime el valor. Si no lo es, se imprime un cero, se imprime
            // el dato actual (que será el primer dato de la siguiente linea) y se pone
            // $aux a 1 (pues ya se ha impreso un campo de la siguiente linea)
            if ($aux==4){
                if (stristr ($indice, '@')){
                    fwrite ($gestor,$valor);
                    fwrite ($gestor,"\n");
                    $aux=0;
                }
                else{
                    fwrite ($gestor, "0");
                    fwrite ($gestor,"\n");
                    if ($valor!="Siguiente"){
                        fwrite($gestor,$valor);
                        fwrite ($gestor,"\t");
                    }
                    $aux=1;
                }
            }
        }

        fclose($gestor);
    }
}

// Función que se encarga de la generación de los ficheros de tipo 3. Recibe como parámetros
// el número de subestructuras en horizontal y en vertical, así como las 4 condiciones de los
// lados de la imagen
function fichero3 ($num_subs_x,$num_subs_y,$cond_up,$cond_down,$cond_left,
$cond_right){
    // Se definen las siguientes variables: $numerador (contador para la subestructura actual)
    // y $tope (marca el número de subestructuras totales en la imagen
    $numerador=0;
    $tope=$num_subs_x*$num_subs_y;
```

```
// Se definen las variables para almacenar las subestructuras o lados limitrofes
$superior=0;
$inferior=0;
$izquierdo=0;
$derecho=0;

// Se entra en un bucle que, mientras no se llegue a la última subestructura, irá calculando
// las subestructuras o lados limitrofes para cada subestructura
while ($numerador<$tope){
    // Si el número de la subestructura es menor que el número de subestructuras en
    // horizontal inferior al número de subestructuras en horizontal significa que la
    // subestructura pertenece al lado inferior, por lo que la condición inferior sera la del
    // lado inferior de la imagen. Si no, la subestructura limitrofe sera la actual menos el
    // número de de subestructuras en horizontal
    if ($numerador <$num_subs_x)
        $inferior=$cond_down;
    else
        $inferior=-($numerador-$num_subs_x);

    // Si el número de la subestructura es mayor que el tope menos el número de
    // subestructuras horizontal significa que la subestructura pertenece al lado superior,
    // por lo que la condición superior sera la del lado superior de la imagen. Si no, la
    // subestructura limitrofe sera la actual más el número de subestructuras en horizontal
    if ($numerador>=($tope-$num_subs_x))
        $superior=$cond_up;
    else
        $superior=-($numerador+$num_subs_x);

    // Si el resto de la división entre la subestructura actual y el número de subestructuras
    // en  horizontal es cero, significa que está en el lado izquierdo, por lo que la condinción
    // será la del lado izquierdo de la imagen. Si no, la subestructura limitrofe por la
    // izquierda será la actual menos 1
    if (($numerador%$num_subs_x)==0)
        $izquierdo=$cond_left;
    else
        $izquierdo=-($numerador-1);

    // Si el resto de la división entre la subestructura actual más 1 y el número de
    // subestructuras en horizontal es cero, significa que la subestructura pertenece al lado
```

```
        // derecho, por lo que la condición es la del lado derecho de la imagen. Si no, la
        // subestructura limítrofe por el lado derecho sera la actual más 1
        if (((($numerador+1)%$num_subs_x)==0)
            $derecho=$cond_right;
        else
            $derecho=-($numerador+1);

        // Se crea el fichero con los datos correspondientes
        $fichero="kernel/fichero3_".$numerador.".txt";
        if ($gestor=fopen($fichero,"w+")) {
            $contenido=$superior."\n".$inferior."\n".$izquierdo."\n".$derecho."\n";
            fwrite($gestor, $contenido);
            fclose($gestor);
        }

        // Se incrementa el índice para proseguir con la siguiente subestructura
        $numerador++;
    }
}

// Función que se encarga de generar los ficheros de tipo 4. Recibe como parámetros un array
// con los datos, y el número de condiciones de contorno
function fichero4 ($datos, $num_cond){
    if ($gestor=fopen("kernel/fichero4.txt","w+")) {
        // Se almacena en el fichero el número de condiciones
        $contenido=$num_cond."\n\n";
        fwrite($gestor, $contenido);

        // Para cada condicion, si es adiabática, se imprimen los datos convenientes.
        // Si no es adiabática, se introducen los datos del formulario
        $indice=1;
        while ($indice<=$num_cond){
            if ($datos["adiab_".$indice]==1){
                $contenido = $indice."\t0\t0\t10000000000\n";
                fwrite ($gestor,$contenido);
            }
            else{
                $contenido = $indice."\t".$datos["a_".$indice]."\t".$datos["b_".$indice]."\t0\n";
                fwrite ($gestor,$contenido);
            }
```

```
            // Se incrementa el índice para la siguiente condición
            $indice++;
        }

        fclose($gestor);
    }
}

// Función que se encarga de generar los ficheros de tipo 5. Recibe como parámetros
// el número de subestructuras, la temperatura inicial del liquido y la velocidad
// de enfriamiento del mismo
function fichero5($num_subs,$temp_ini,$vel){
    if ($gestor=fopen("kernel/fichero5.txt","w+")) {
        // Se añade una linea por subestructura
        $contador=0;
        while ($contador<$num_subs){
            $contenido=$temp_ini."\t".$vel."\n";
            fwrite($gestor, $contenido);
            $contador++;
        }

        fclose($gestor);
    }
}

// Función que se encarga de generar los ficheros de tipo 6. Recibe como párametros
// el número de subestructuras en horizontal y en vertical, así como el tiempo
// de simulación
function fichero6($num_subs_x,$num_subs_y,$tiempo){
    if ($gestor=fopen("kernel/fichero6.txt","w+")) {

$contenido=($num_subs_x*$num_subs_y)."\n\n".$tiempo."\n\n".$num_subs_x."\n\n".$num
_subs_y."\n";
        fwrite($gestor, $contenido);
        fclose($gestor);
    }
}

// Función que se encarga de la generación necesarios para la visualización de los
```

```
// resultados mediante Matlab. Recibe como parámetros el fichero con los resultados
// y el tiempo a visualizar
function call_matlab ($fichero,$tiempo){
    // Se cambia al directorio kernel
     $directorio=getcwd();
     chdir($directorio."/kernel");

    if ($gestor=fopen($fichero,"r")) {
        if ($gestor2=fopen("startup.m","w+")) {

            // Se introduce la matriz con los datos de la simulación para el tiempo
            // seleccionado
            $contenido="A=[\n";
            fwrite($gestor2, $contenido);
            while (!feof($gestor)){
               $contenido=fgets($gestor);
                fwrite($gestor2, $contenido);
            }

            $contenido="]\n";
            fwrite($gestor2, $contenido);

            $contenido="B=max(A)\n";
            fwrite($gestor2, $contenido);

            $contenido="v_max=max(B)\n";
            fwrite($gestor2, $contenido);

            $contenido="D=mean(A)\n";
            fwrite($gestor2, $contenido);

            $contenido="media=mean(D)\n";
            fwrite($gestor2, $contenido);

            $contenido="F=std(A)\n";
            fwrite($gestor2, $contenido);

            $contenido="desv=std(F)\n";
            fwrite($gestor2, $contenido);
```

```
            // Se introducen las sentencias de lenguaje matlab necesarias para obtener
            // la imagen en un fichero
            $contenido="figure\n";
            fwrite($gestor2, $contenido);

            $contenido="h=pcolor(A)\n";
            fwrite($gestor2, $contenido);

            $contenido="shading interp\n";
            fwrite($gestor2, $contenido);

            $contenido="colorbar\n";
            fwrite($gestor2, $contenido);

            $contenido="title ({'Instante de Tiempo: ".$tiempo."';['Valor Máximo: ',
num2str(v_max), ' | Valor Medio: ', num2str(media), ' | Desviación Típica: ',
num2str(desv)]})\n";
            fwrite($gestor2, $contenido);

            $contenido="set (gca,'ydir','reverse')\n";
            fwrite($gestor2, $contenido);

            $contenido="axis image\n";
            fwrite($gestor2, $contenido);

            // Se obtiene el directorio actual
            $work_dir=getcwd();

            $contenido="saveas (1, '".$work_dir."\figura.png')\n";
            fwrite($gestor2, $contenido);

            $contenido="quit";
            fwrite($gestor2, $contenido);

            fclose($gestor);
            fclose($gestor2);

            // Se ejecuta Matlab con el fichero de ordenes que hemos creado
            exec ("matlab startup.m");
```

```
            // Se elimina el fichero de Matlab generado
            unlink ("startup.m");
        }
    }

    // Se vuelve al directorio de trabajo original
    chdir($directorio);
}

// Función que se encarga del borrado de los ficheros generados para la simulación
function limpieza (){
    // Se cambia al directorio kernel
    $directorio=getcwd();
    chdir($directorio."/kernel");

    // Si se eliminan todos los ficheros salvo los ejecutables
    foreach (glob("*.*") as $fichero)
        if ($fichero!="modulo1.exe" and $fichero!="convert.exe" and $fichero!=
"cryojab_color.exe")
        unlink ($fichero);

    // Se vuelve al directorio de trabajo original
    chdir($directorio);
}

// Función que extrae en un fichero de texto la matriz de resultados para un instante
// dado. Recibe como parámetro el instante solicitado
function ext_res ($instante){
    if ($gestor=fopen("kernel/resultadosjuntos.txt","r")) {
        if ($gestor2=fopen("kernel/res_inst_".$instante.".txt","w+")) {
            // Se inicializan las variables $compara y $aux
            $compara=FALSE;
            $aux=-1;

            // Mientras no se llegue al final del fichero y no se encuentre el tiempo
            // solicitado, se recorre el fichero con todos los resultados
            while (!feof($gestor) and $aux!=$instante){
               // Se extrae una linea
                $contenido=fgets($gestor);
                // Se mira si contiene el caracter '=' y se extrae a partir de él
```

```
                $compara = stristr ($contenido, '=');
                // Se extrae el instante que viene tras el '='
                sscanf($compara, "= %d",$aux);
            }

            // Se avanzan dos lineas, pues el fichero de resultados contiene dos lineas en blanco
            // a partir del instante que identifica la matriz
            $contenido=fgets($gestor);
            $contenido=fgets($gestor);

            // Se selecciona el siguiente instante y se inicializan las variables otra vez
            $instante_final=$instante+1;
            $compara=FALSE;
            $aux=-1;

            // Mientras no se llegue al final del fichero y no se detecte el identificador
            // del siguiente instante, se copia el contenido de un fichero en otro
            while (!feof($gestor) and $aux!=$instante_final){
               $contenido=fgets($gestor);
                $compara = stristr ($contenido, '=');
                sscanf($compara, "= %d",$aux);
                if ($aux!=$instante_final){
                    fwrite ($gestor2, $contenido);
                }
            }
            fclose ($gestor);
            fclose ($gestor2);
        }
    }
}

// Función que extrae las condiciones de contorno del fichero de tipo 4.
// Devuelve un array de dos dimensiones con todas las condiciones.
function show_cond (){
    // Se abre el fichero de tipo 4.
    if ($gestor=fopen("kernel/fichero4.txt","r")) {
        // Se saltan las 3 primeras lineas
        $contenido=fgets($gestor);
        $contenido=fgets($gestor);
        $contenido=fgets($gestor);
```

```
        $indice=1;
        // Se extraen las condiciones y se almacenan en el array
        while (!feof($gestor)){
            sscanf($contenido, "%d\t%d\t%d\t%d",$aux_indice,$aux_a,$aux_b,$aux_r);
            if ($indice==$aux_indice){
                $cond[$aux_indice]["a"]=$aux_a;
                $cond[$aux_indice]["b"]=$aux_b;
                $cond[$aux_indice]["r"]=$aux_r;
                $indice++;
            }
            $contenido=fgets($gestor);
        }
    }

    // Se devuelve el array
    return $cond;
}
?>
```