Método de Volúmenes Finitos
Ejercicios Prácticos

Santiago Chialvo

Noviembre 20, 2015

1. Problema 1: Manipuleo de Datos Geométricos

Para este ejercicio se pide que, dada una malla en formato nodos (xnod) y conectividades (icone) en 3D, se genere un código que calcule los arreglos de OpenFOAM

- (1) Faces.
- (2) Points.
- (3) Owner.
- (4) Neighbor.

Y las propiedades geométricas

- (1) Volumen de cada celda.
- (2) Vector de Área Normal.
- (3) Centroide de cada Celda.
- (4) Centroide de cada Cara.

A continuación se presenta el código desarrollado y se irá explicando parte por parte los pasos seguidos. El software utilizado para escribir el mismo fue Matlab.

El array *lenicone* contiene la cantidad de celdas presentes en la malla. Empezamos calculando el volúmen y el centroide de cada celda, dependiendo si se trata de un hexaedro o un tetraedro. Si se trata de un hexaedro, el volúmen se calcula dividiendo el mismo en 5 tetraedros y calculando el volúmen de cada uno, luego se suman.

```
function [faces,points,owner,neighbor,Vceldas,Cceldas,Afaces,Cfaces] = Ejercicio1_FVM(xnode,icone,hexa)

%%
%Example
%icone = [1 2 3 4 5 6 7 8 ;5 6 7 8 9 10 11 12];
%xnode = [xnode = [0 0 0; 1 0 0; 1 1 0; 0 1 0; 0 0 0.1; 1 0 0.1; 1 1 0.1; 0 1 0.1; 0 0 0.2; 1 0 0.2; 1 1 0.2; 0 1 0.2];

points = xnode;

lenicone = length(icone(:,1)); %Cantidad de celdas

%% Calculo de volumenes y centroides de celdas

Vceldas = zeros(lenicone,1); %vector con volumenes de las celdas
Cceldas = zeros(lenicone,3); %vector con los centroides de las celdas

if (hexa)

    for i=1:lenicone
        tets = [points(icone(i,1),:) points(icone(i,2),:) points(icone(i,4),:) points(icone(i,5),:); points(icone(i,2),:) points(icone(i,4),:) points(icone(i,3),:) points(icone(i,7),:); points(icone(i,5),:) points(icone(i,2),:) points(icone(i,8),:) points(icone(i,7),:); points(icone(i,2),:) points(icone(i,5),:) points(icone(i,6),:) points(icone(i,7),:); points(icone(i,2),:) points(icone(i,5),:) points(icone(i,4),:) points(icone(i,7),:)];
        Vceldas(i) = 0;

        for j=1:5
        tet = tets(j,:);
        v1 = [tet(1) tet(2) tet(3)];
        v2 = [tet(4) tet(5) tet(6)];
        v3 = [tet(7) tet(8) tet(9)];
        v4 = [tet(10) tet(11) tet(12)];
        A = v2 - v1;
        B = v3 - v1;
        C = v4 - v1;
        mat = [A;B;C];
        Vceldas(i) = Vceldas(i) + (1/6)*abs(det(mat));
```

```
        end

        %Calcular centroide
        for j=1:8
            Cceldas(i,:) = Cceldas(i,:) + points(icone(i,j),:);
        end
        Cceldas(i,:) = Cceldas(i,:)/8;
    end

else

    for i=1:lenicone
        Vceldas(i) = 0;

        v1 = [points(icone(i,1),:) 1];
        v2 = [points(icone(i,2),:) 1];
        v3 = [points(icone(i,3),:) 1];
        v4 = [points(icone(i,4),:) 1];
        mat = [v1;v2;v3;v4];
        Vceldas(i) = (1/6)*abs(det(mat));

        %Calcular centroide
        for j=1:4
            Cceldas(i,:) = Cceldas(i,:) + points(icone(i,j),:);
        end
        Cceldas(i,:) = Cceldas(i,:)/4;
    end

end
```

Para cada celda, se asignan sus 4 o 6 caras correspondientes en el arreglo *faces* (Siguiendo la regla de la mano derecha, todas las normales apuntando hacia afuera). Luego, en un bucle interno, se asignan todas las caras con su *owner* correspondiente (Inclusive las repetidas, que serán eliminadas luego).

```
%% Calculo de faces y owner

if (hexa)

    faces = zeros(lenicone*6,4); %6 caras por celda, cada una con 4 vertices
    owner = zeros(lenicone*6,1); %6 caras, cada una con 1 propietario

    for i=1:lenicone
```

```
        faces((6*(i-1))+1,:) = [icone(i,5) icone(i,6) icone(i,7) icone(i,8)];
        faces((6*(i-1))+2,:) = [icone(i,1) icone(i,5) icone(i,8) icone(i,4)];
        faces((6*(i-1))+3,:) = [icone(i,2) icone(i,3) icone(i,7) icone(i,6)];
        faces((6*(i-1))+4,:) = [icone(i,1) icone(i,2) icone(i,6) icone(i,5)];
        faces((6*(i-1))+5,:) = [icone(i,4) icone(i,8) icone(i,7) icone(i,3)];
        faces((6*(i-1))+6,:) = [icone(i,1) icone(i,4) icone(i,3) icone(i,2)];

        for j=(6*(i-1) + 1):(6*(i-1) + 6) %Primero ponemos todas, inclusive repetidas
            owner(j) = (i);
        end
    end

else

    faces = zeros(lenicone*4,3); %4 caras por celda, cada una con 3 vertices
    owner = zeros(lenicone*4,1); %4 caras, cada una con 1 propietario

    for i=1:lenicone
        faces((4*(i-1))+1,:) = [icone(i,1) icone(i,3) icone(i,2)];
        faces((4*(i-1))+2,:) = [icone(i,1) icone(i,4) icone(i,3)];
        faces((4*(i-1))+3,:) = [icone(i,3) icone(i,4) icone(i,2)];
        faces((4*(i-1))+4,:) = [icone(i,1) icone(i,2) icone(i,4)];

        for j=(4*(i-1) + 1):(4*(i-1) + 4) %Primero ponemos todas, inclusive repetidas
            owner(j) = (i);
        end
    end

end
```

Una vez inicializados estos arrays, se procede a calcular la longitud del array *faces* con el fin de generar un array del mismo tamaño, pero esta vez con cada cara con la numeración ordenada en forma ascendente. También creamos un array *ownerdup* que será una copia exacta de *owner* que será utilizado más adelante para buscar caras repetidas.

```
lenfaces = length(faces(:,1));
if (hexa)
    facesorted = zeros(lenfaces,4);
else
    facesorted = zeros(lenfaces,3);
end
owner_dup = owner;

for i=1:lenfaces
    facesorted(i,:) = sort(faces(i,:)); %las ordeno para buscar
        repetidas
end
```

Luego inicializar el array *neighbor* en cero, inicio un bucle while en el cual, a partir de una cara del arreglo *facesorted* dada, busco desde allí en adelante alguna otra igual. Si esto se cumple (Condición if de línea 9), se le asigna al array *neighbor* el propietario de esa cara (Aquí se utiliza el array *ownerdup* dado que el *owner* original cambia en cada iteración). Luego esa cara es eliminada de *faces*, *facesorted* y *owner*.

```
1  neighbor = zeros(1,1); %es de long variable asi que lo seteo en
       1 fila
2  neighbor_count = 1; %para ir viendo la fila donde estoy parado
3
4  i=1;
5
6  while (i<=(lenfaces-neighbor_count+1))
7      j = i+1;
8      while (j<=(lenfaces-neighbor_count+1)) %busco de ahi para
           adelante
9          if (facesorted(i,:) == facesorted(j,:)) %si encuentro
               una repetida
10
11             neighbor(neighbor_count,1) = owner_dup(i);
12             faces(j,:) = []; %lo elimino de faces
13             facesorted(j,:) = []; %lo elimino de facesorted
14             owner(j) = []; %lo elimino de propietarios
15             neighbor_count=neighbor_count+1;
16
17         end
18         j=j+1;
```

```
    end
    i=i+1;
end

end
```

Finalmente, para el cálculo de el vector de área normal y centroide de cada cara, se recalcula la longitud del array *faces* y para cada cara se utilizan fórmulas de área con vectores según se traten de hexas o tetras.

```
lenfaces = length(faces(:,1));

Afaces = zeros(lenfaces,3); %vector normal area de las caras
Cfaces = zeros(lenfaces,3); %centroide de las caras

if (hexa)

    for i=1:lenfaces
        v1 = points(faces(i,2),:) - points(faces(i,1),:);
        v2 = points(faces(i,4),:) - points(faces(i,1),:);
        Afaces(i,:)=cross(v1,v2);

        Cfaces(i,:) = (points(faces(i,1),:) + points(faces(i,2),:) + points(faces(i,3),:) + points(faces(i,4),:))/4;
    end

else

    for i=1:lenfaces
        v1 = points(faces(i,2),:) - points(faces(i,1),:);
        v2 = points(faces(i,3),:) - points(faces(i,1),:);
        Afaces(i,:)=0.5*cross(v1,v2);

        Cfaces(i,:) = (points(faces(i,1),:) + points(faces(i,2),:) + points(faces(i,3),:))/3;
    end

end
```

2. Problema 2: Difusion con Fuente en 1D

Para este problema se pide resolver la siguiente ecuación:

$$\int_{\Omega_j} Q^\phi d\Omega + \int_{\Gamma_j} \Gamma^\phi \nabla\phi \cdot d\Gamma_j = 0 \tag{1}$$

En primer lugar es necesario discretizar el término fuente y el término difusivo. Como ya sabemos, el término difusivo se divide en un aporte en la cara derecha y la cara izquierda, esto puede ser visto como una sumatoria para cada celda:

$$Q_c^\phi |\Omega_c| + \sum_{f=li,ld} \Gamma^\phi \nabla\phi \cdot \mathbf{S}_f = 0 \tag{2}$$

Finalmente, el término $\nabla\phi$ puede ser visto como $\frac{\phi_c - \phi_w}{h}$ y el volúmen de la celda $|\Omega_c|$ como Sh, reemplazando:

$$Q_c^\phi Sh + \sum_{f=li,ld} \Gamma^\phi \frac{\phi_e - \phi_c}{h} S - \Gamma^\phi \frac{\phi_c - \phi_w}{h} S = 0 \tag{3}$$

El código que se utilizó para resolver tanto éste problema como el 3 es el mismo, sólo con diferentes parámetros. En cada sub-sección se irán indicando los parámetros elegidos y mostrando los resultados obtenidos. Al final del informe se detallará el código en cuestión.

2.1. Caso 1

Los parámetros utilizados fueron:

- $L = 1.$
- $tiempofinal = 1.$
- $tipotemporal = 3.$
- *deltaT = 1*
- $Q = 10.$
- $Gamma = 1.$
- $v = 0.$
- $rho = 1.$

- *c (del termino reactivo)* = 0.
- *phi0* = 0.
- *phi1* = 1.

Se utilizó el código con estos parámetros y con un h = [0.2, 0.1, 0.05, 0.025]. En la figura 1 observamos una gráfica del error en función del h.

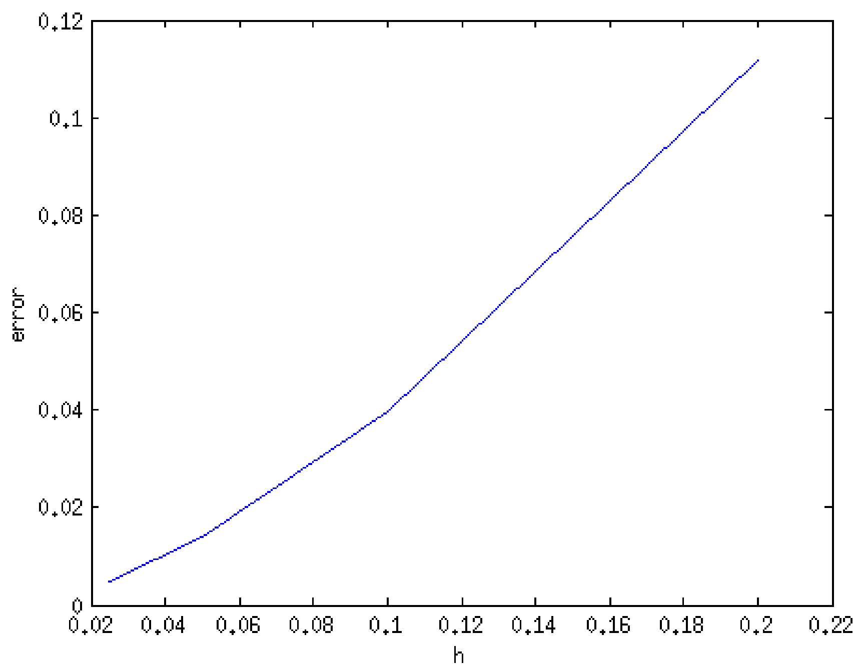

Figura 1: Gráfica del error para el caso 1

2.2. Caso 2

Los parámetros escogidos para este caso fueron los mismos que el caso 1, sólo que se impuso una condición Neumann en el lado izquierdo igual a 1. En este caso el error desciende con h de manera similar al caso 1. Los resultados pueden observarse en la figura 2.

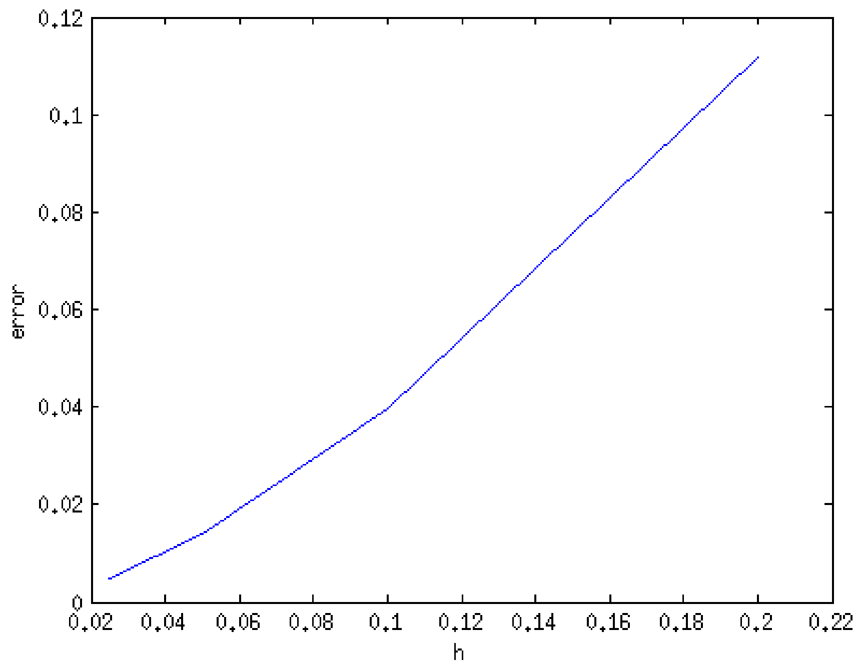

Figura 2: Gráfica del error para el caso 2

2.3. Caso 3

En este problema se tuvo que modificar el código original un poco para que se genere una malla con h variable dependiendo de la siguiente función:

$$h = 1 - 0{,}75e^{\frac{(\xi-0{,}5)^2}{0{,}01}} \tag{4}$$

Con $\xi = 0 : 0{,}01 : 1$

Además se agrega una fuente variable según un intervalo. El código utilizado para agregar esto es el siguiente:

```
Eta = 0:0.001:1;
lenEta = length(Eta);

h = 1 - 0.75*e.^(-((Eta-0.5).^2 ./ 0.01));
h = h/sum(h);

x(1) = 0;
```

```

for i=2:lenEta
    x(i) = x(i-1) + h(i-1);
end

cant_celdas = length(x)-1;

Q = zeros(cant_celdas,1);

for i=1:cant_celdas
    if ((x(i) >= 0.25)&&(x(i) <= 0.75))
      Q(i) = 10;
    end
end
```

El resultado obtenido puedeo observarse en la figura 3.

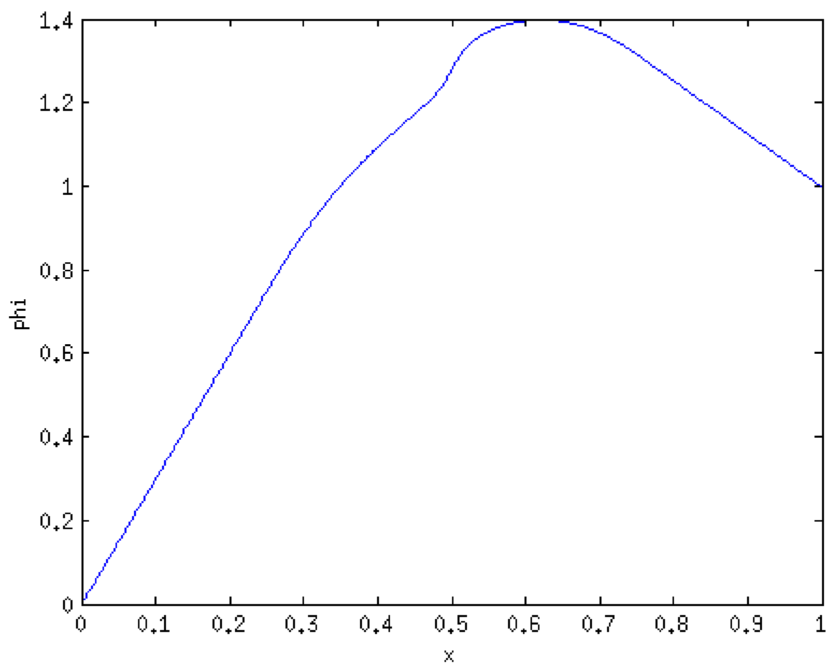

Figura 3: Gráfica de ϕ para el caso 3

2.4. Caso 4

Para este caso los parámetros seteados son similares al caso 1 pero la fuente Q es igual a 0.1 para todas las celdas y el valor de c en el término reactivo es 10. El error en función del h se observa en la figura 4.

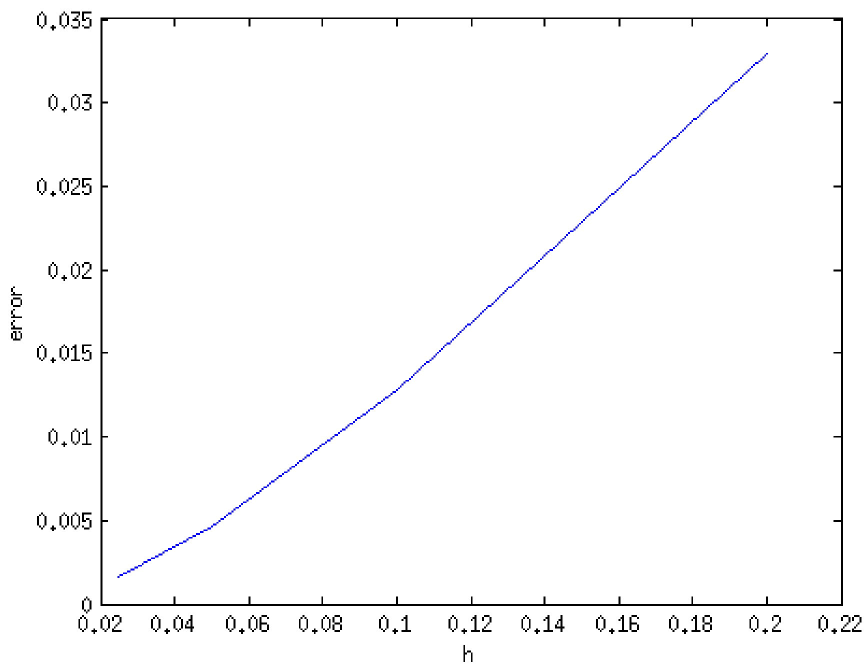

Figura 4: Gráfica del error para el caso 4

2.5. Caso 5

Para este caso se impuso una condición mixta en el lado izquierdo con $\phi_{\text{ínf}} = 2$ y $h_{\text{ínf}} = 10$. El resultado obtenido puede verse en la figura 5.

3. Problema 3: Advección - Difusión con fuente en 1D transiente

Para este problema se pide resolver la siguiente ecuación:

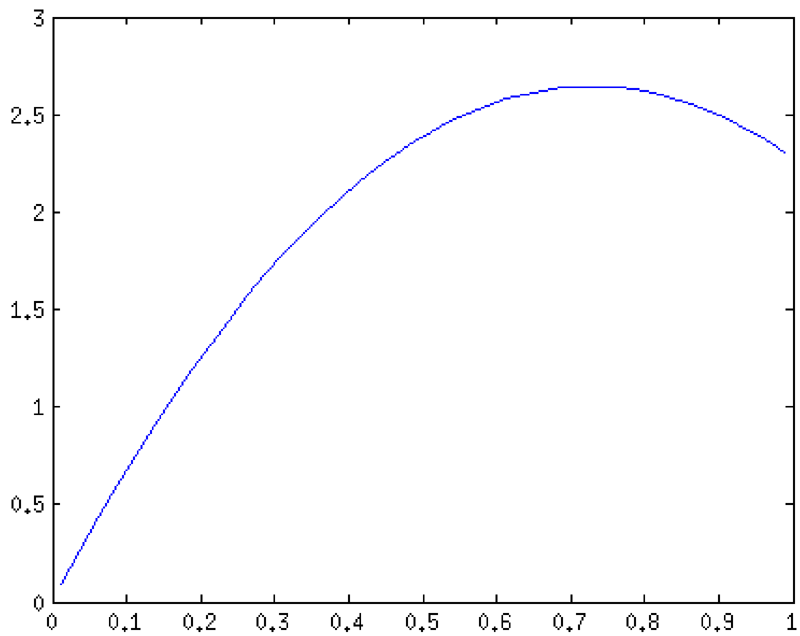

Figura 5: Gráfica de ϕ para el caso 5

$$\int_{\Omega_j} \rho \frac{d\phi}{dt} d\Omega + \int_{\Gamma_j} \rho\phi\mathbf{v} \cdot d\Gamma_j + \int_{\Omega_j} c\phi d\Omega = \int_{\Omega_j} Q^{\phi} d\Omega + \int_{\Gamma_j} \Gamma^{\phi} \nabla\phi \cdot d\Gamma_j \quad (5)$$

Un forma de discretizar la ecuación (5) es la siguiente:

$$\rho \frac{\phi_c^{n+1} - \phi_c^n}{\Delta t} h + \rho v(\phi_{ld}^{n+\theta} - \phi_{li}^{n+\theta}) + c\phi_c^{n+\theta} h = Q_c^{\phi} Sh + \sum_{f=li,ld} \Gamma^{\phi} \frac{\phi_e - \phi_c}{h} S - \Gamma^{\phi} \frac{\phi_c - \phi_w}{h} S \quad (6)$$

3.1. Caso 1

Para este caso, se pedía resolver en estado estacionario un problema con los siguientes parámetros:

- $L = 1$.

- *tiempofinal* = 1.
- *tipotemporal* = 3.
- *deltaT = 1*
- $Q = 0$.
- *Gamma* = 1.
- $v = 1$.
- *rho* = 1.
- *c (del termino reactivo)* = 0.
- *phi0* = 0.
- *phi1* = 1.

La solución analítica a este problema está dada por la siguiente ecuación:

$$y(x) = \frac{(e^x - 1)}{(e - 1)} \tag{7}$$

El error para h = [0.2, 0.1, 0.05, 0.025] con upwind difference puede observarse en la figura 6. El orden de precisión para este método se estima en 1, es decir de primer orden, ya que el error tiende a cero como h.

El error para los mismos h pero esta vez con central difference puede observarse en la figura 7. El orden de precisión para este método se estima en 2, es decir de segundo orden, ya que el error tiende a cero como h^2.

3.2. Caso 2

El mismo problema que el caso anterior, esta vez variando el término *Gamma*, en primer lugar a 0.1. En las figuras 8 y 9 observamos los resultados.

Con un Gamma = 0.01 se tuvieron que probar varios valores más de h para que se notara bien la diferencia entre ámbos métodos, desde h = 0.2 hasta h= 1.5625e-3 dividiendo h/2. Se observa como el comportamiento de central difference en la figura 10 es lineal hasta h=0.1 y luego el error desciende cuadráticamente. En la figura 11 con upwind difference se observa claramente el descenso lineal del error respecto a h.

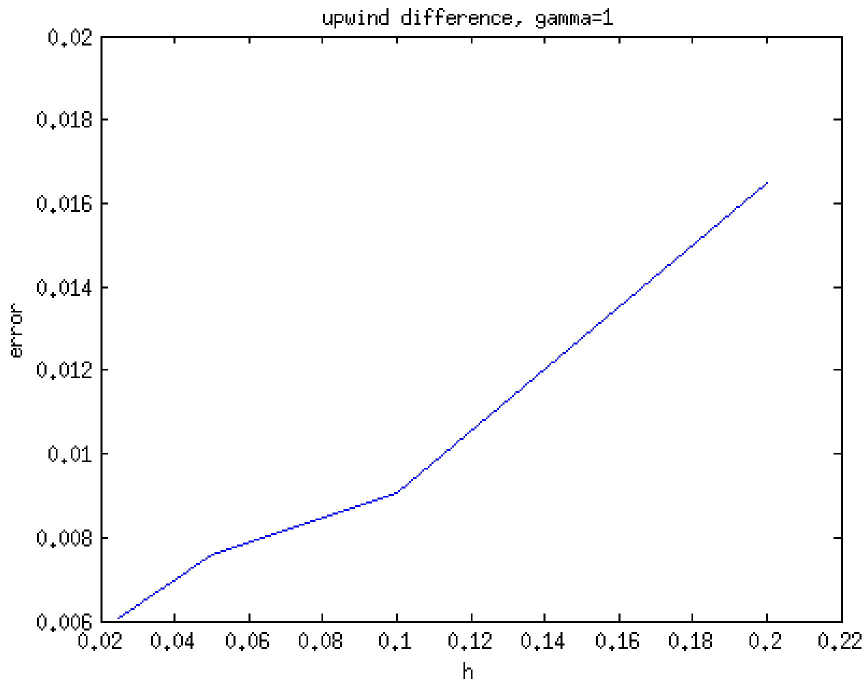

Figura 6: Gráfica del error para el caso 1 con upwind difference

3.3. Caso 3

Para este problema, con 20 celdas no se alcanzaba a discretizar bien el gran salto que tiene la función con Gamma=0.01 entre x=[0.9,1], dando resultados totalmente imprecisos. Esto puede observarse en la figura 12.

Sin embargo, con un h = 0.005 y un deltaT = 0.001 se lograban mejores resultados. Sabemos que la elección de una deltaT adecuado es indispensable para asegurar la convergencia de éste método.($tipotemporal$=1) En la figura 13 se observa la gráfica obtenida con respecto a la solución exacta y en la figura 14 como desciende el error con el paso del tiempo. El número de Courant obtenido en este caso es de 0.2. Como el se cumple que sea menor que la unidad, el sistema explícito es estable.

Para el caso implícito, ($tipotemporal$=2) no modificando ningún parámetro de los anteriores obtenemos el mismo número de Courant. Los resultados son apreciables en las figuras 15 y 16. Aumentando luego del deltaT al doble, el número de Courant aumenta consecuentemente al doble y se obtienen los

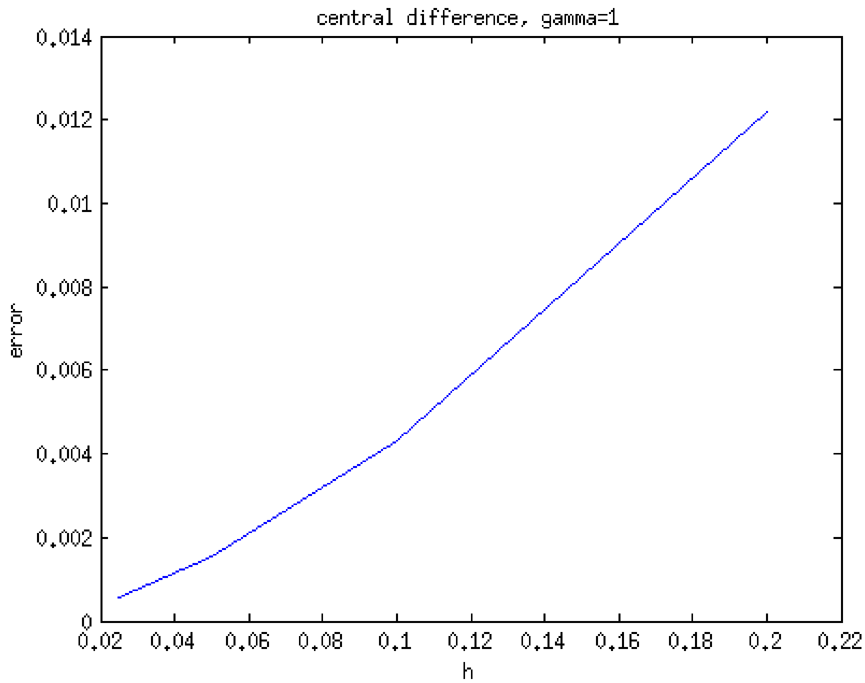

Figura 7: Gráfica del error para el caso 1 con central difference

resultados apreciables en las figuras 17 y 18. Los resultados son similares al caso anterior. El sistema es estable ya que no tiene limitación en el paso del tiempo en el caso lineal.

Para el tercer caso, el método semi-explícito (*tipotemporal*=0) se utiliza nuevamente un deltaT = 0.001 y un h=0.005 obteniendo así un Courant = 0.2, luego se usa un paso igual al doble del anterior. Los resultados son apreciables en las figuras 19,20,21 y 22.

4. Código utilizado

A continuación se presenta el código utilizado para resolver el problema 1 y 3, escrito en el software Matlab.

```
function [err,hvs] = TP2MC_Ej3_nuevo ()

%% Variables iniciales
```

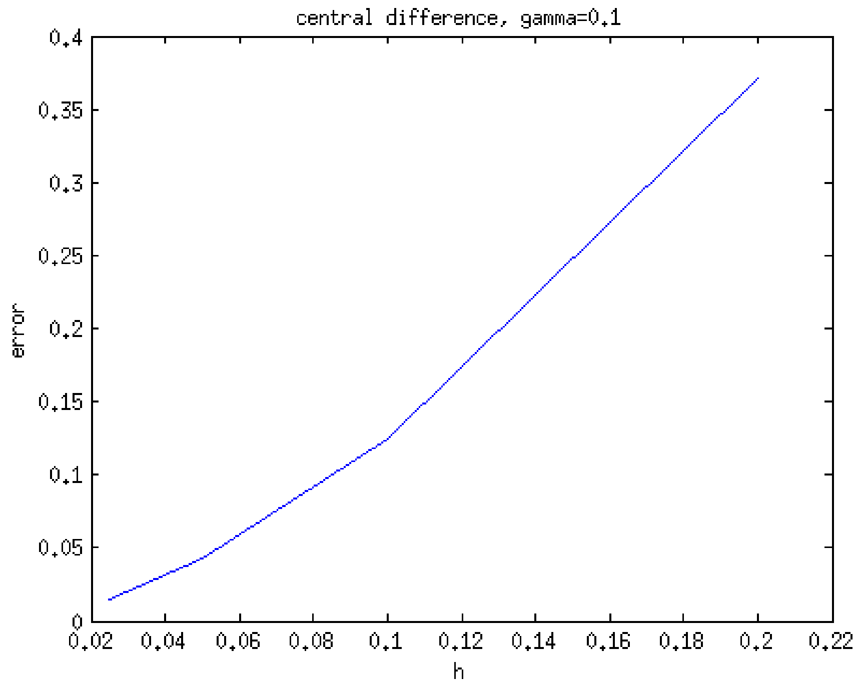

Figura 8: Gráfica del error para el caso 2, Gamma = 0.1 con upwind difference

```
h=0.005;
L = 1;
x = 0:h:L;
cant_celdas = length(x) - 1;

deltaT = 0.002;
tiempofinal = 1.5;
t = 0:deltaT:tiempofinal;
e = exp(1);

cant_tiempos = length(t);
hv = zeros(cant_celdas,1)+h;
Q = zeros(cant_celdas,1)+0;

tipotemporal = 0; %% Tipo de metodo de discretizacion temporal: 0-semi-explicito , 1-explicito , 2-implicito , 3-ninguno (estacionario)

% Termino convectivo
v = 1;
```

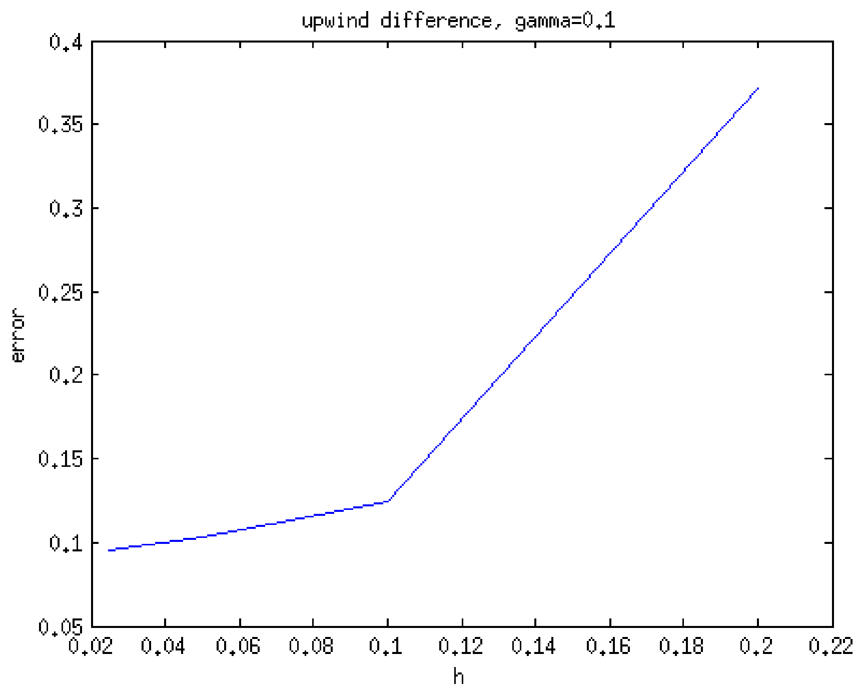

Figura 9: Gráfica del error para el caso 2, Gamma = 0.1 con central difference

```
rho = 1;
upwind = true;
if (upwind)
    if (v>0)
        alfa = 1;
        beta = 0;
    else
        alfa = 0;
        beta = 1;
    end
end

%Termino reactivo
c = 0;

%Termino difusivo
Gamma = 0.01;

%Cond de borde
phi0 = 0; %izquierda
```

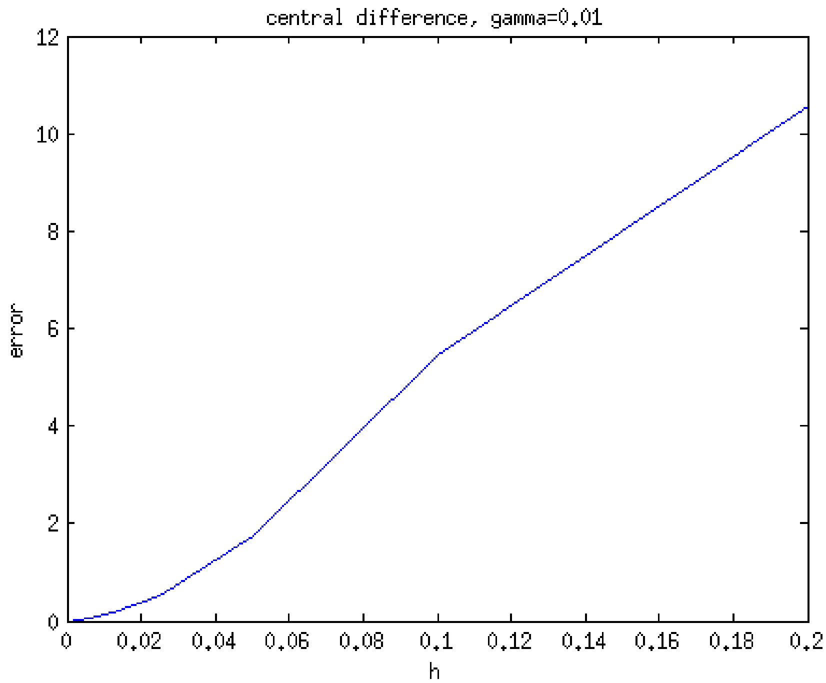

Figura 10: Gráfica del error para el caso 2, Gamma = 0.01 con central difference

```
phi1 = 1; %derecha

gamabi = 1;
gamabd = 1;
hinf = 10;

cond = [0 0]; %0-dirichlet , 1-neumann, 2-mixta

%% Inicializaciones de estructuras

K = zeros(cant_celdas,cant_celdas);
F = zeros(cant_celdas,1);
left = true;

a = zeros(cant_celdas);
a(:,1) = 1;

Courant = ((deltaT*v) / h);
disp(['Numero de Courant: ' num2str(Courant)]);
```

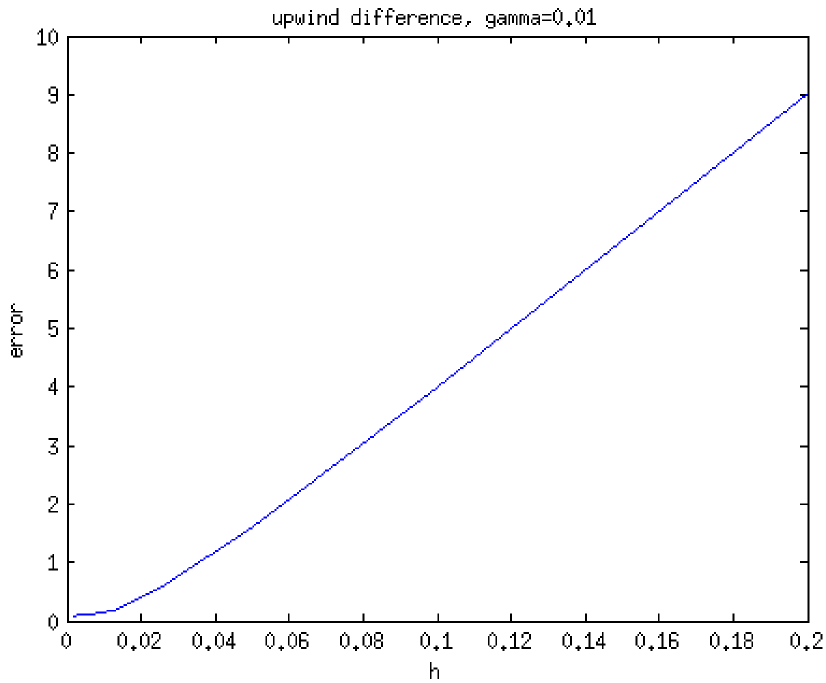

Figura 11: Gráfica del error para el caso 2, Gamma = 0.01 con upwind difference

```
Peclet = ((h*v) / Gamma);
disp(['Numero de Peclet: ' num2str(Peclet)]);

%syms phi_left phi_cent phi_right;

%% Loop principal

for i=1:cant_celdas %cada celda

    for j=i:i+1 %cada cara

        if (j == 1) %primer cara
            %No hago nada
            left = false;

        else
            if (j == cant_celdas+1) %ultima cara
                %No hago nada

```

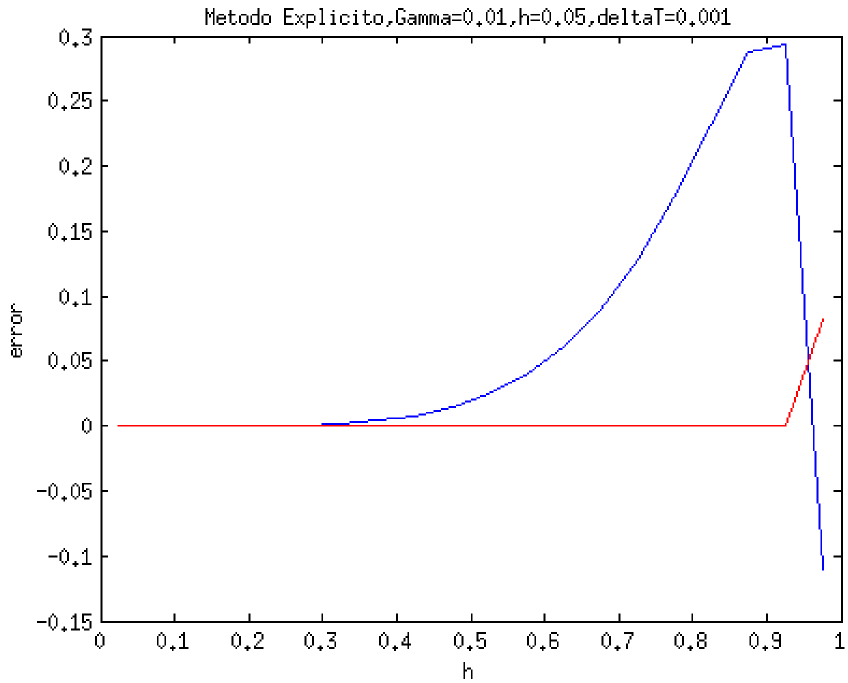

Figura 12: Método explícito, h=0.05

```
            else
                if (left)
                    %Eq = Eq + (-1 * Gamma * (1/h) * (phi_left -
                        phi_cent));
                    gf = hv(i)/(x(i+1) - x(i-1));
                    Gamma1 = Gamma*(1-gf) + Gamma*gf;
                    phili = rho*v;

                    if (upwind)
                        K(i,j-1) = K(i,j-1) - alfa*phili;
                        K(i,j) = K(i,j) - beta*phili;
                    else
                        K(i,j) = K(i,j) - (1-gf)*rho*v;
                        K(i,j-1) = K(i,j-1) - (gf*rho*v);
                    end

                    K(i,j) = K(i,j) + (Gamma1 * (1/hv(i)));
                    K(i,j-1) = K(i,j-1) - (Gamma1 * (1/hv(i)));

```

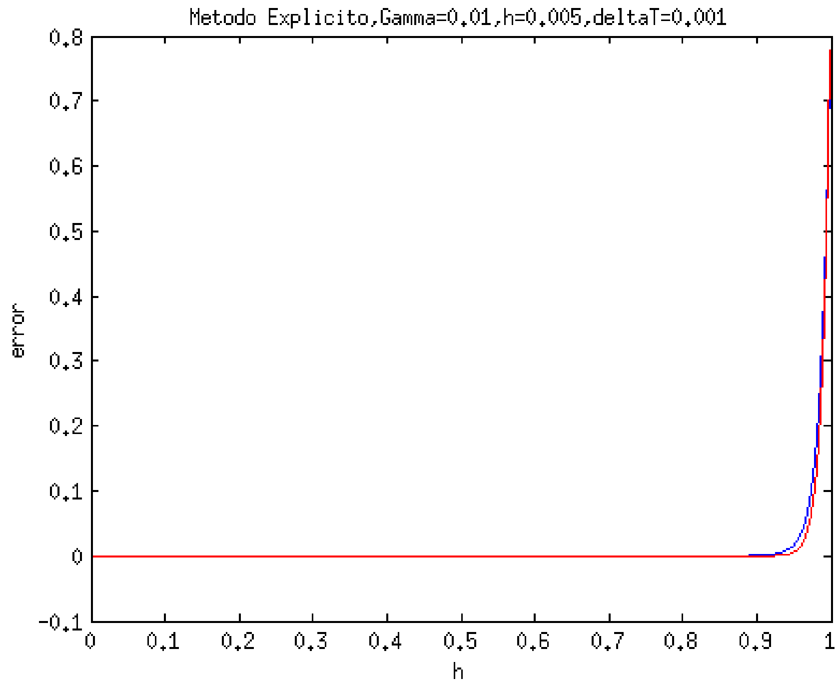

Figura 13: Método explícito, h=0.005, deltaT = 0.001

```
                left = false;
            else
                if (~left)
                    %Eq = Eq + (Gamma * (1/h) * (phi_right -
                        phi_cent));
                    gf = hv(i)/(x(i+2) - x(i));
                    Gamma1 = Gamma*(1-gf) + Gamma*gf;
                    phild = rho*v;

                    if (upwind)
                        K(i,j) = K(i,j) + beta*phild;
                        K(i,j-1) = K(i,j-1) + alfa*phild;
                    else
                        K(i,j) = K(i,j) + (gf*rho*v);
                        K(i,j-1) = K(i,j-1) + (1-gf)*rho*v;
                    end

                    K(i,j) = K(i,j) - Gamma1 * (1/hv(i));
                    K(i,j-1) = K(i,j-1) + (Gamma1 * (1/hv(i)
                        ));
```

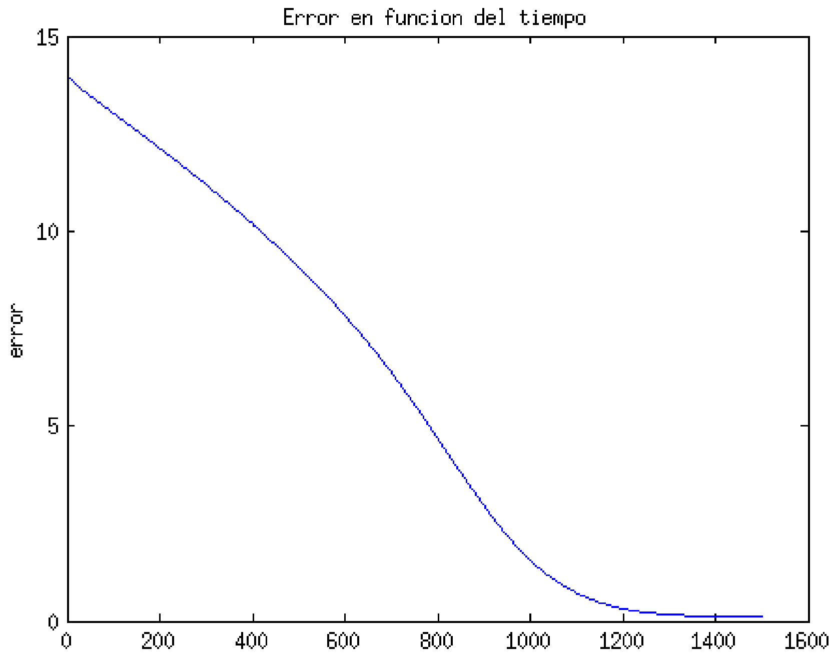

Figura 14: Método explícito, h=0.005, deltaT = 0.001, error en función del tiempo

```

                                left = true;
                            end
                        end
                    end
                end

            end
        K(i,i)= K(i,i) + c*hv(i);
        F(i) = F(i) + Q(i)*hv(i);

end

%% Cond de contorno

h1 = hv(1)/2;  %h de la primer celda
hf = hv(cant_celdas)/2;  %h de la ultima celda

%Dirichlet
```

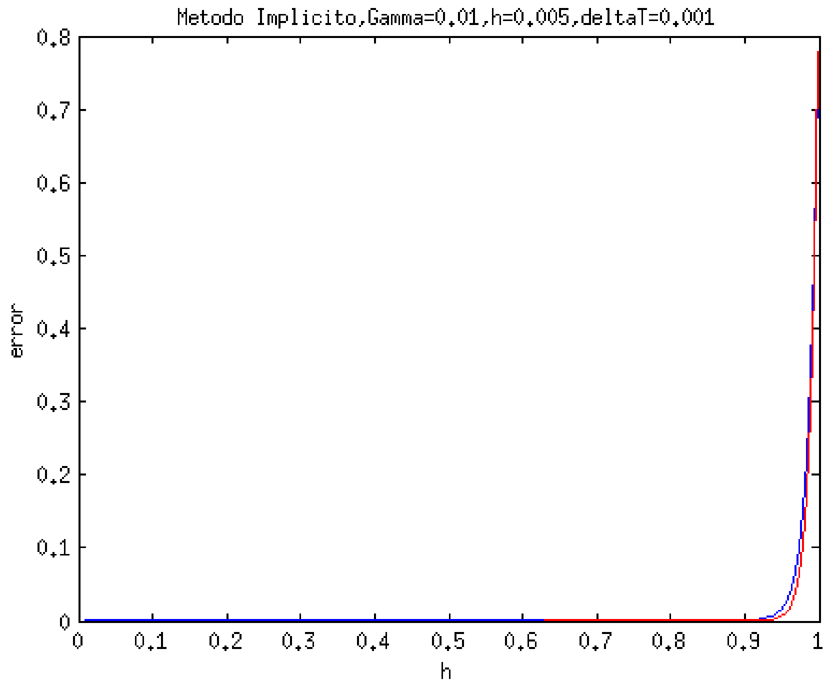

Figura 15: Método implícito, h=0.005, deltaT = 0.001

```
if (cond(1) == 0)
    %termino difusivo
    K(1,1) = K(1,1) + Gamma/h1;
    F(1) = F(1) + (Gamma*phi0)/h1;
    %termino convectivo
    F(1) = F(1) + rho*v*phi0;
end
if (cond(2) == 0)
    %termino difusivo
    K(cant_celdas,cant_celdas) = K(cant_celdas,cant_celdas) +
        Gamma/hf;
    F(cant_celdas) = F(cant_celdas) + (Gamma*phi1)/hf;
    %termino convectivo
    F(cant_celdas) = F(cant_celdas) - rho*v*phi1;
end

%Neumann
if (cond(1) == 1)
    %termino difusivo
    F(1) = F(1) - phi0;
```

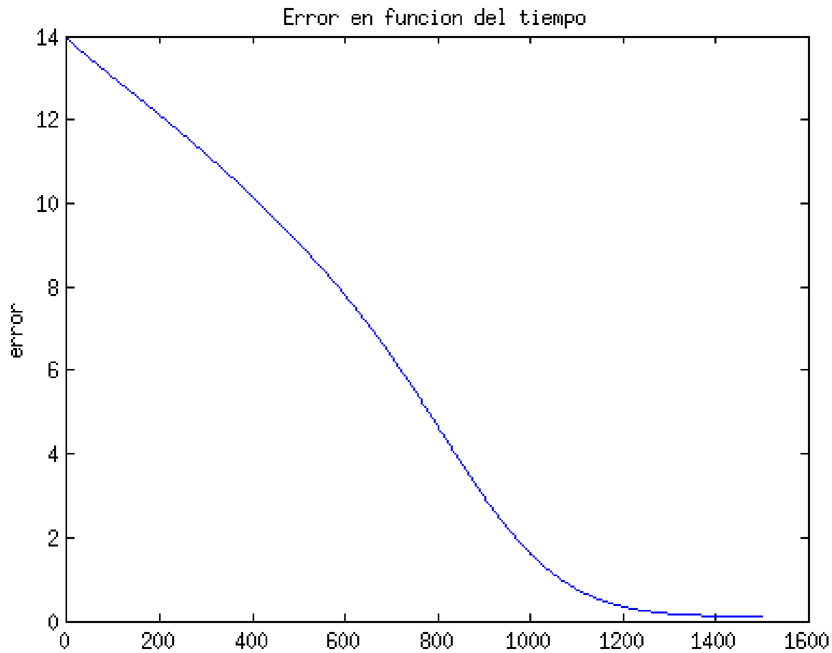

Figura 16: Método implícito, h=0.005,, deltaT = 0.001, error en función del tiempo

```
    %termino convectivo
    K(1,1) = K(1,1) - rho*v;
    F(1) = F(1) + rho*v*phi0*h1;
end
if (cond(2) == 1)
    %termino difusivo
    F(cant_celdas) = F(cant_celdas) + phi1;
    %termino convectivo
    K(cant_celdas,cant_celdas) = K(cant_celdas,cant_celdas) +
        rho*v;
    F(cant_celdas) = F(cant_celdas) - rho*v*phi1*hf;
end

%Mixtas

if (cond(1) == 2)
    aux = (hinf*(gamabi/h1))/(hinf+(gamabi/h1));
    convecinf = (hinf*h1)/(gamabi+h1 * hinf);
    convecC = gamabi / (gamabi+h1 * hinf);
```

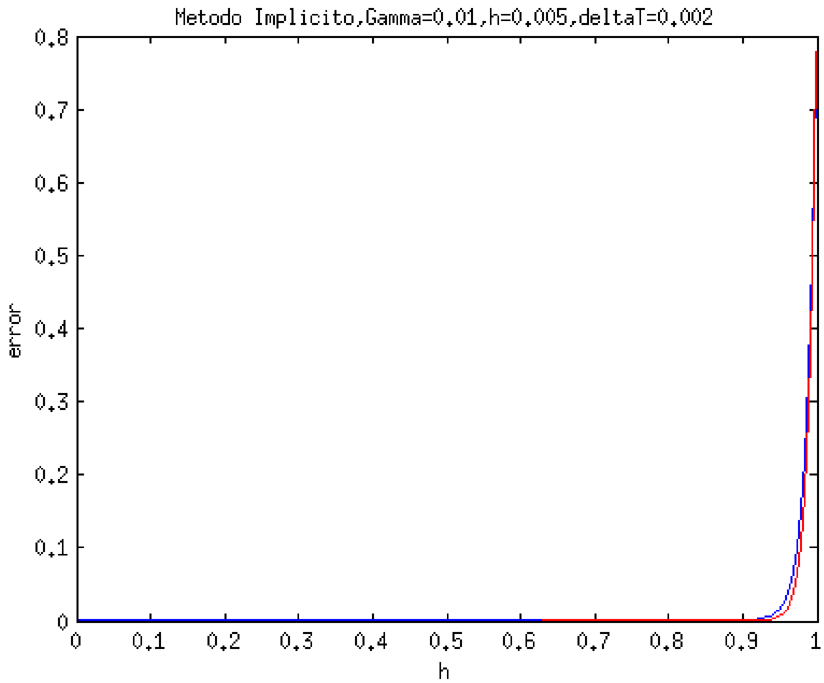

Figura 17: Método implícito, h=0.005, deltaT = 0.002

```
    %termino difusivo
    F(1) = F(1) + aux*phi0;
    K(1,1) = K(1,1) + aux;
    %termino convectivo
    K(1,1) = K(1,1) - convecC*rho*v;
    F(1) = F(1) - (convecinf*phi0)*rho*v;
end

if (cond(2) == 2)
    aux = (hinf*(gamabd/hf))/(hinf+(gamabd/hf));
    convecinf = (hinf*hf)/(gamabd+hf * hinf);
    convecC = gamabd / (gamabd+hf * hinf);
    %termino difusivo
    F(cant_celdas) = F(cant_celdas) + aux*phi1;
    K(cant_celdas,cant_celdas) = K(cant_celdas,cant_celdas) +
        aux;
    %termino convectivo
    K(cant_celdas,cant_celdas) = K(cant_celdas,cant_celdas) +
        convecC*rho*v;
    F(cant_celdas) = F(cant_celdas) + (convecinf*phi1)*rho*v;
```

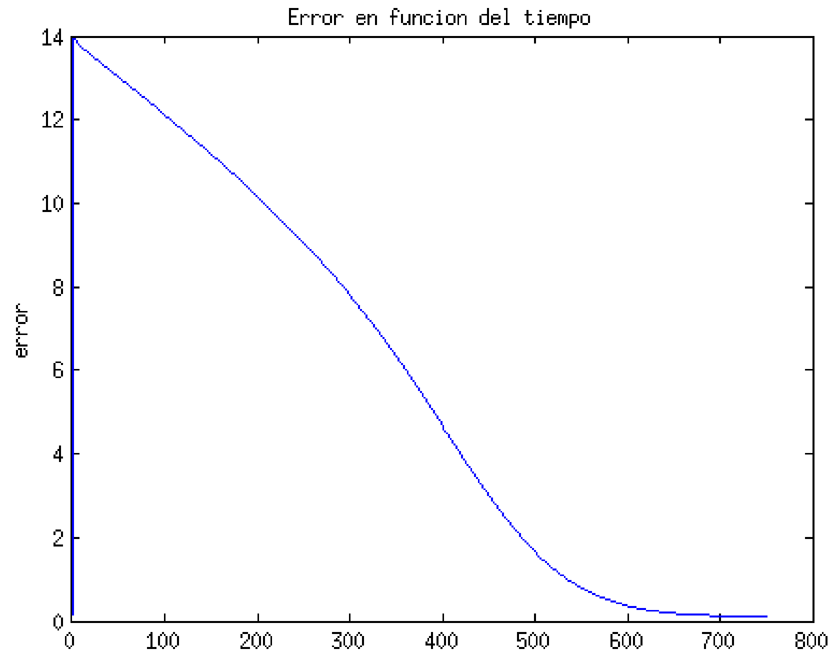

Figura 18: Método implícito, h=0.005, deltaT = 0.002, error en función del tiempo

```
end

%% Calculo de los centroides

for i=1:cant_celdas
    xmid(i)= x(i)+(x(i+1)-x(i))/2;
end

%% Metodo temporal
if (tipotemporal == 0) %semi explicito

terminotemporal = (rho.*hv)./deltaT;
K0(:,:) = K(:,:)./2;
Kn(:,:) = K0(:,:);

for i=1:cant_celdas
    Kn(i,i) = Kn(i,i) + terminotemporal(i);
    K0(i,i) = K0(i,i) - terminotemporal(i);
end
```

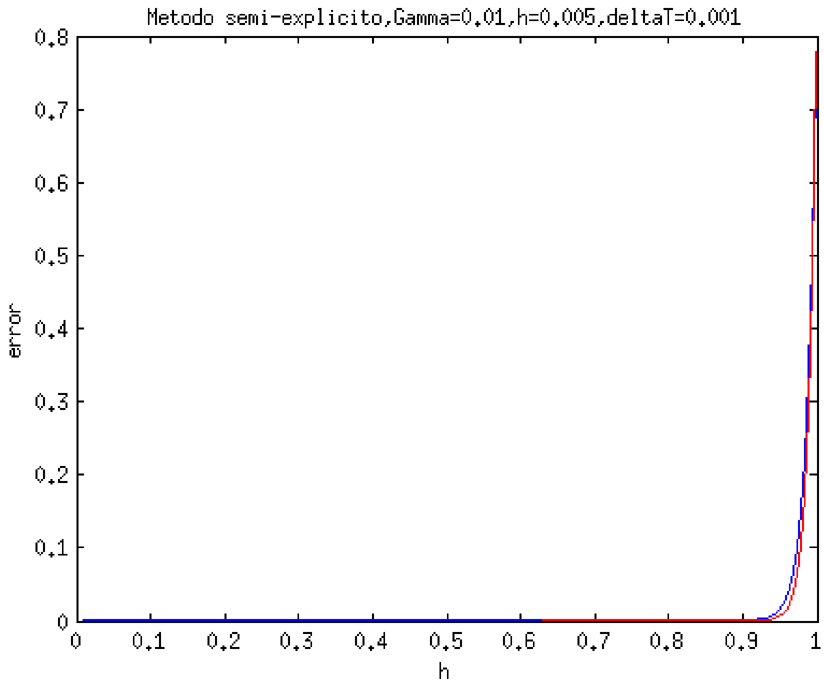

Figura 19: Método semi-explícito, h=0.005, deltaT = 0.001

```

for time=2:cant_tiempos
    a(:,time) = Kn(:,:)\(F(:) - (K0(:,:)*a(:,time-1)));

    err(time) = norm((a(:,time) - exact'));
end

else if (tipotemporal == 1) %explicito
        Faux = zeros(cant_celdas,1);
        for i = 1:cant_celdas
            Kaux(i,:) = -(deltaT/(rho*hv(i)))*K(i,:);
            Faux(i) = (deltaT/(rho*hv(i)))*F(i);
            Kaux(i,i) = Kaux(i,i)+1;
        end

        for time=2:cant_tiempos
            a(:,time) = Kaux*a(:,time-1) + Faux;

            err(time) = norm((a(:,time) - exact'));
        end
```

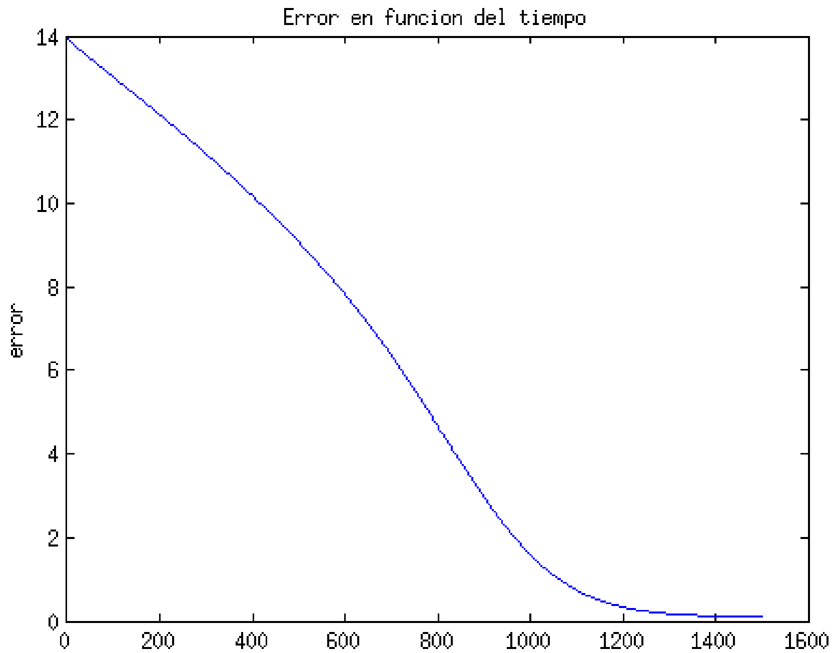

Figura 20: Método semi-explícito, h=0.005,, deltaT = 0.001, error en función del tiempo

```

    else if (tipotemporal == 2) %implicito
            terminotemporal = (rho.*hv)./deltaT;
            Kaux = K;
            for i = 1:cant_celdas
                Kaux(i,i) = K(i,i) + terminotemporal(i);
            end
            Faux = F;

            for time=2:cant_tiempos
                Faux = F + a(:,time-1).*terminotemporal;

                a(:,time) = Kaux\Faux;

                err(time) = norm((a(:,time) - exact'));
            end

        else
            a(:,1) = K\F;
```

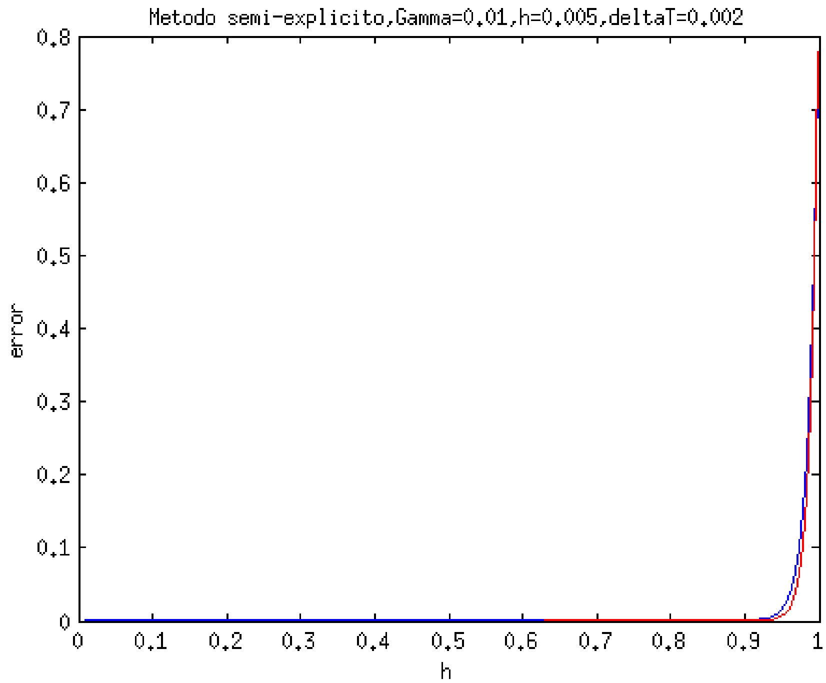

Figura 21: Método semi-explícito, h=0.005, deltaT = 0.002

```
                err(1) = norm((a(:,1) - exact'));
            end
        end
end

%end

figure;
for i=1:cant_tiempos-1
    title(i*deltaT);
    pause(0.001);
    plot(xmid,a(:,i));
end
hold on;
plot(xmid,exact,'r');
```

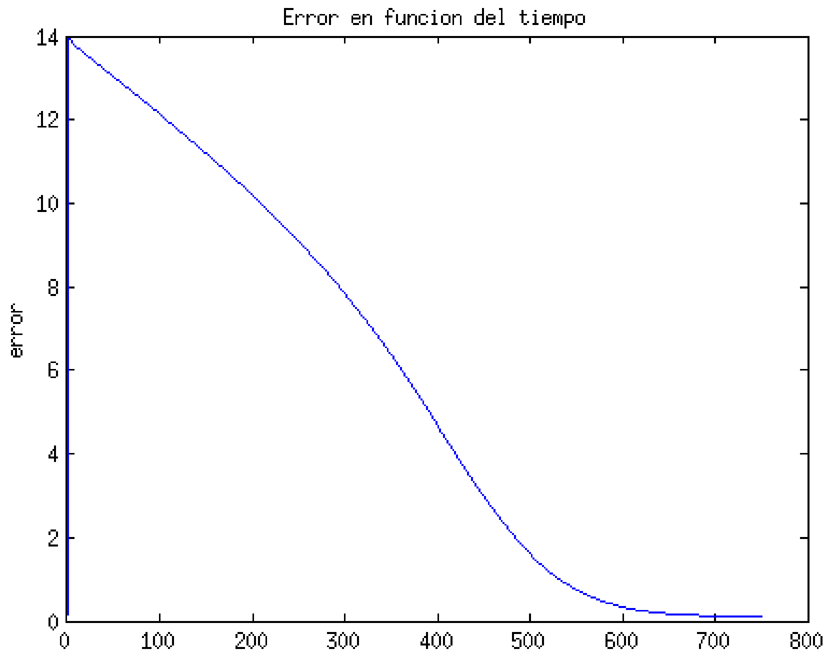

Figura 22: Método semi-explícito, h=0.005, deltaT = 0.002, error en función del tiempo

5. Difusión con fuente en 1D estacionaria en OpenFOAM

Para resolver este problema, primero generaremos la malla modificando el archivo *blockMeshDict* del tutorial *PitzDaily* de *scalarTransportFoam* de la siguiente manera:

```
vertices
(
    (0 -0.1 -0.1)
    (1 -0.1 -0.1)
    (1 0.1 -0.1)
    (0 0.1 -0.1)
    (0 -0.1 0.1)
    (1 -0.1 0.1)
    (1 0.1 0.1)
    (0 0.1 0.1)
```

```
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (50 1 1) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
    side1
    {
        type patch;
        faces
        (
                (0 4 7 3)
        );
    }
        side2
    {
        type patch;
        faces
        (
                        (1 2 6 5)
        );
    }
    empty
    {
        type empty;
        faces
        (
            (0 1 5 4)
            (5 6 7 4)
            (3 7 6 2)
            (0 3 2 1)
        );
    }
);
```

Para setear el Gamma en 0.1, el archivo *transportPropierties* es modificado. Para un h=0.02 y deltaT=0.001 con upwind-difference se obtiene algo como se observa en las figuras 23 y 24. Para central-difference los resultados se observan en la figura 25.

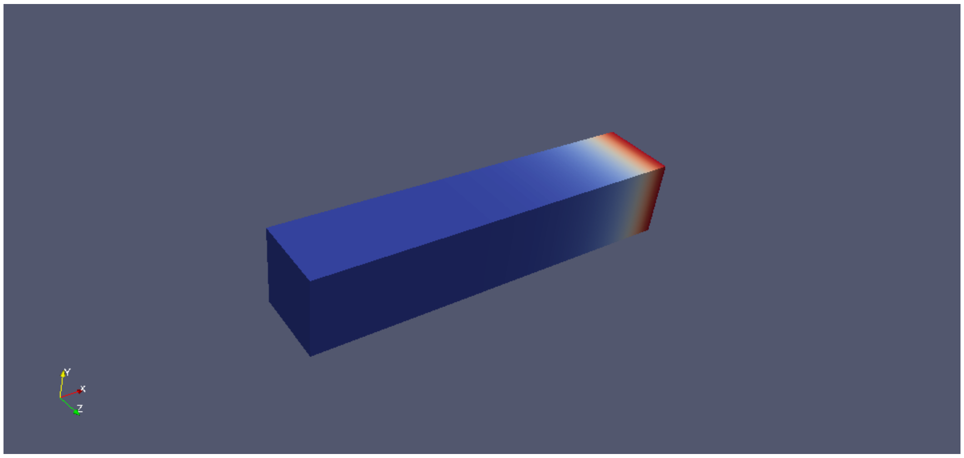

Figura 23: Screenshot del problema con upwind-difference y Gamma 0.1

De la misma manera, para un Gamma=0.01 se siguen los mismos pasos y se obtiene con upwind-difference los resultados de las figuras 26 y 27 y para CD los resultados de la imagen 28.

Si refinamos la malla hacia el lado de x=1 usando Grading, modificamos la línea correspondiente en *blockMeshDict* para hacer un grading de 50 en el eje x:

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (50 1 1) simpleGrading (50 1 1)
);
```

Los resultados se observan en las figuras 29,30,31,32 y 33. En estas gráficas se observa como al estar mejor discretizada la última porción de la malla donde se distribuye la temperatura, el salto de temperatura que se produce en casos con el Gamma=0.01 se distribuye mejor en la gráfica y no se ve tan abrupto.

6. Adveccion-Difusión en 2D transiente con OpenFOAM con velocidad constante

Para generar el dominio se modificó el archivo *blockMeshDict* de la siguiente forma:

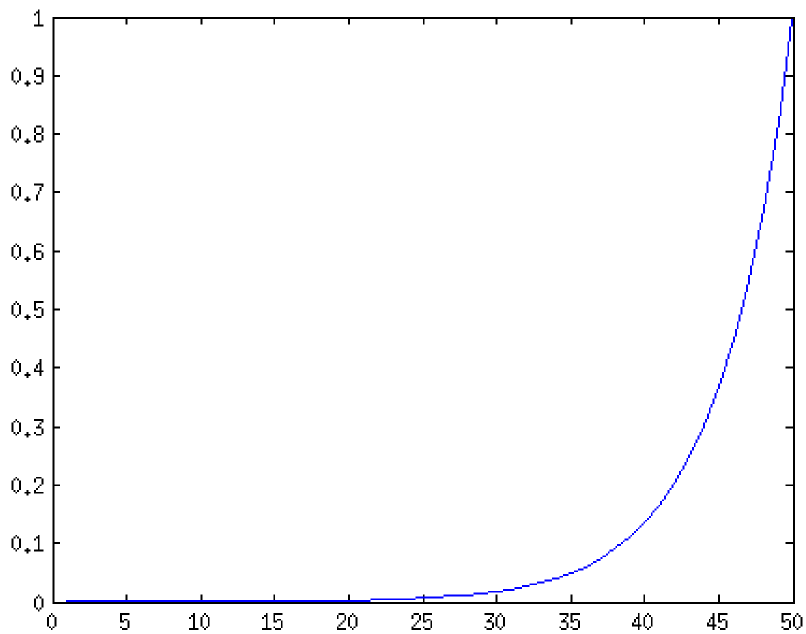

Figura 24: Gráfica de la temperatura final con upwind-difference y Gamma 0.1

```
vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (40 40 1) simpleGrading (1 1 1)
);

edges
```

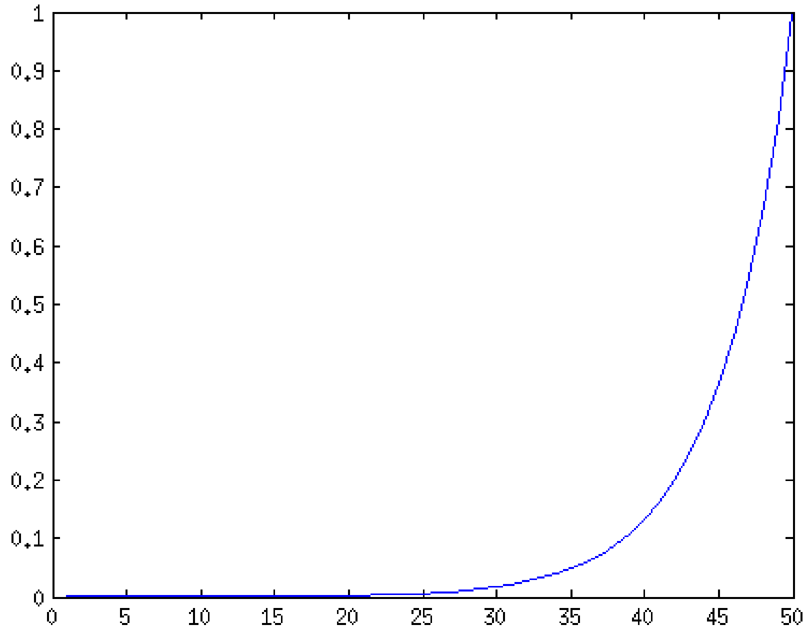

Figura 25: Gráfica de la temperatura final con central-difference y Gamma 0.1

```
(
);

boundary
(
    caraarriba
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
        caraizq
    {
        type wall;
        faces
        (
            (0 4 7 3)
```

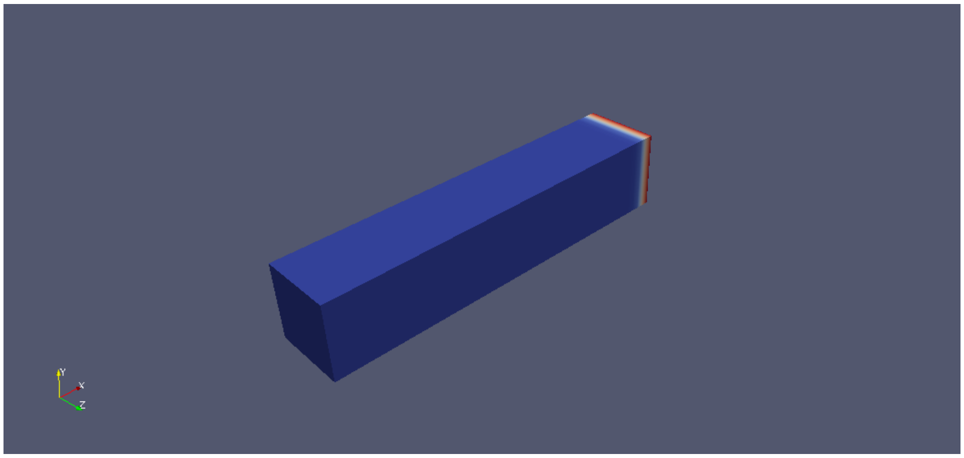

Figura 26: Screenshot del problema con upwind-difference y Gamma 0.01

```
        );
    }
    carader
    {
        type wall;
        faces
        (
            (2 6 5 1)
        );
    }
        caraabajo
    {
        type wall;
        faces
        (
            (1 5 4 0)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```

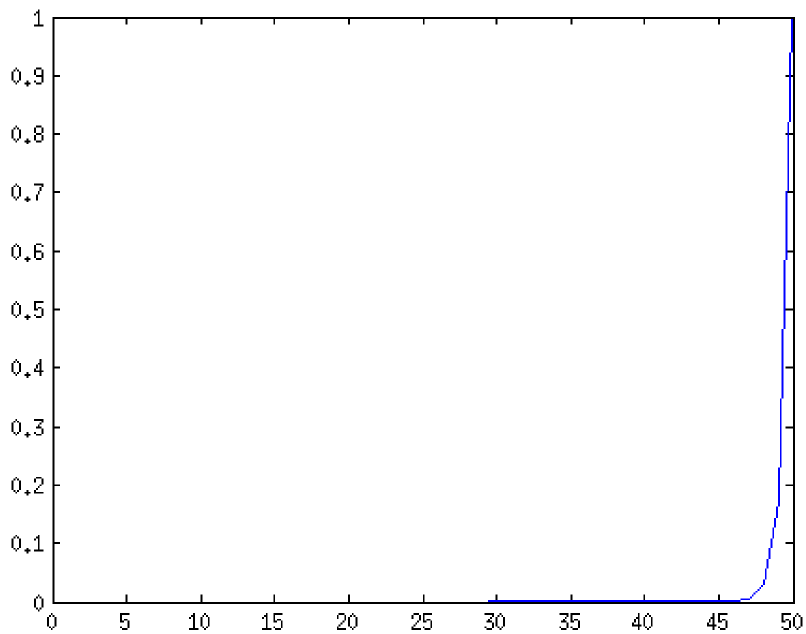

Figura 27: Gráfica de la temperatura final con upwind-difference y Gamma 0.01

Luego de setear las temperaturas y la velocidades correspondientes en la carpeta 0 y Γ en la carpeta *Constant* los resultados obtenidos fueron los siguientes (Figuras 34,35,36 y 37). Al ser Gamma tan pequeño, la temperatura no llega a distribuirse de manera completa en todo el dominio. Si aumentamos Gamma a 1, observamos un transporte de temperatura mas distribuido como se observa en la figura 38.

7. Adveccion-Difusión en 2D transiente con OpenFOAM con velocidad variable en el espacio

En primer lugar, para lograr el número de Raynolds pedido la viscosidad cinemática debe ser seteada en 0.0025, en el archivo *transportProperties*. Una vez convergido el campo de velocidades, se obtiene lo que se observa en la figura 39.

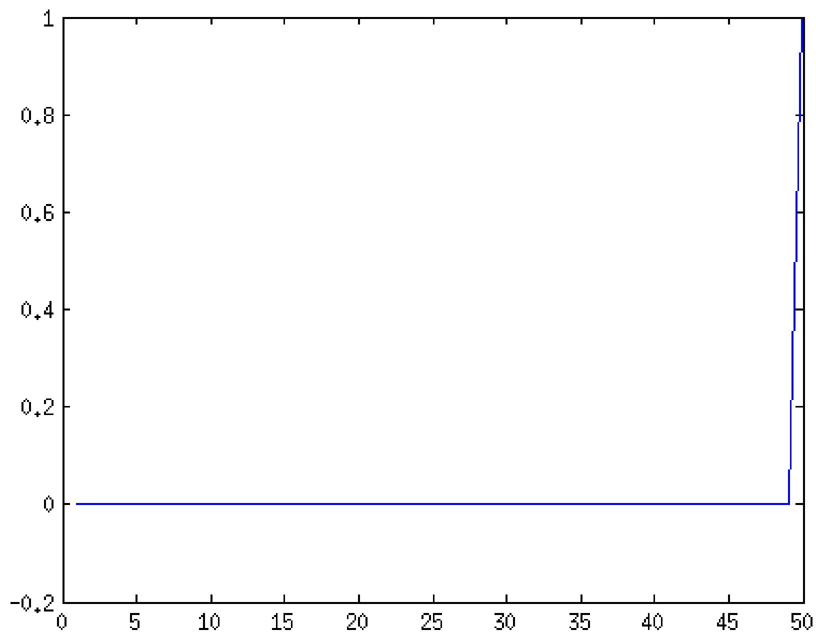

Figura 28: Gráfica de la temperatura final con central-difference y Gamma 0.01

Con este campo de velocidades, debemos armar un caso que resuelva el transporte de la temperatura usando el campo de velocidades hallado. Para ello, en nuestro nuevo caso debemos modificar el archivo U de la carpeta 0, y setear nuestro campo de velocidades encontrado como elcampo interno de velocidades del nuevo caso. Para setear las temperaturas se modificó el archivo T en la carpeta 0 de la siguiente forma:

```
internalField    uniform 0;

boundaryField
{
        carasuperior
    {
        type            fixedGradient;
                gradient                uniform 100;
    }
        lateralizquierda
    {
```

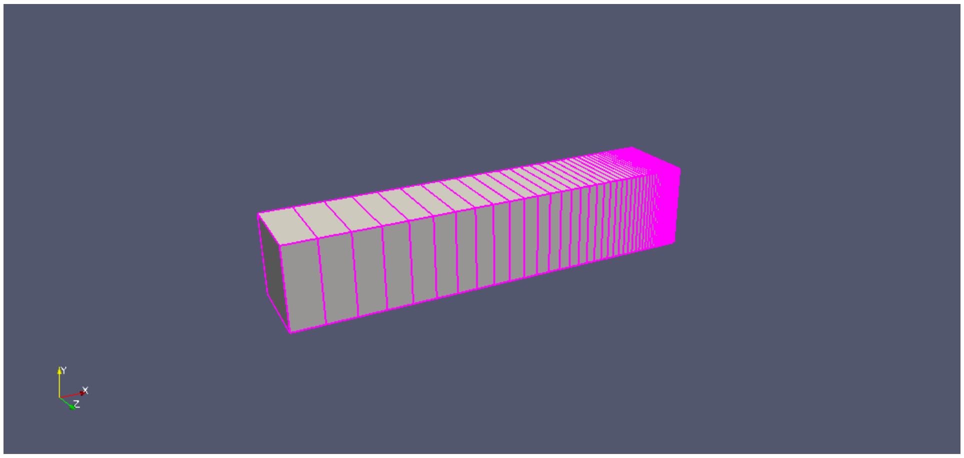

Figura 29: Malla refinada hacia el extremo x=L.

```
        type              fixedValue;
                value                         uniform 100;
    }
    lateralderecha
    {
        type              fixedValue;
                value                         uniform 200;
    }
        carainferior
    {
        type              zeroGradient;
    }
    frontAndBack
    {
        type empty;
    }
}
```

El campo de temperatura obtenido puede observarse en la figura 40. Distintas isotermas del mismo pueden verse en la figura 41. Aquí observamos como las isotermas son rectas en la pared inferior adiabática y la temperatura superior se halla en el extremo superior derecho, dado que el campo de velocidades la transporta hacia ese extremo.

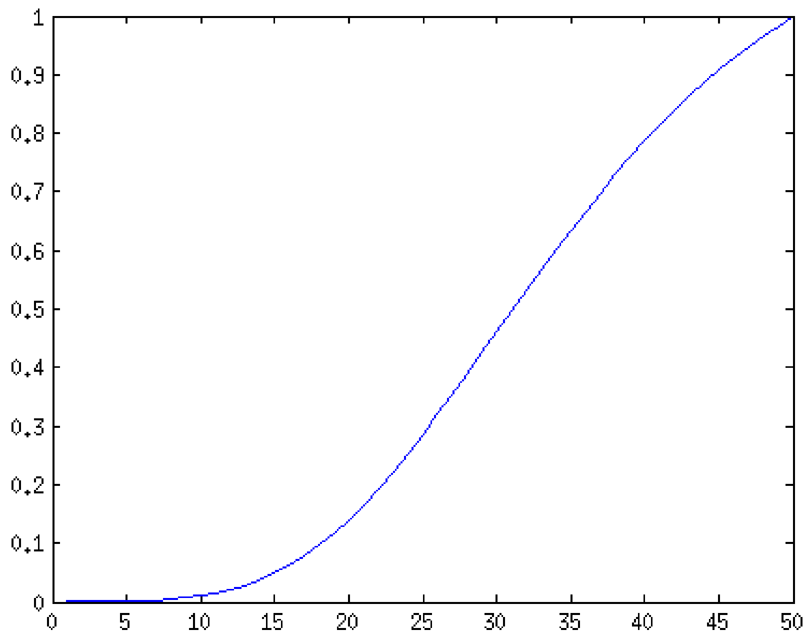

Figura 30: Malla refinada hacia el extremo x=L, gráfica de la temperatura final con upwind-difference y Gamma 0.1

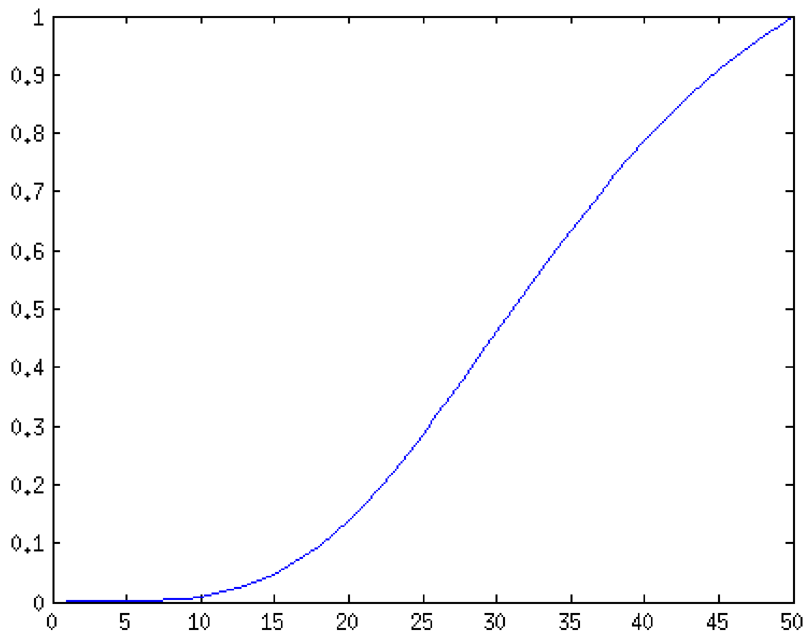

Figura 31: Malla refinada hacia el extremo x=L, gráfica de la temperatura final con central-difference y Gamma 0.1

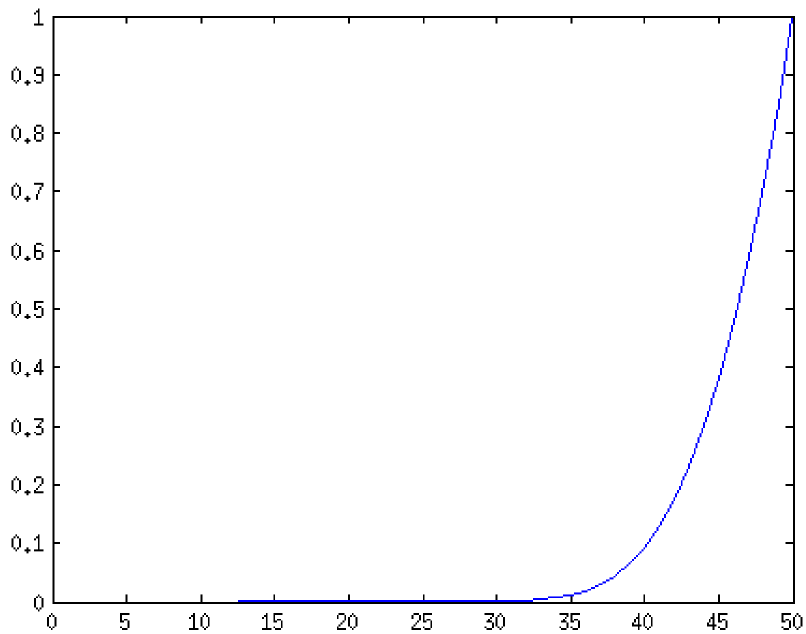

Figura 32: Malla refinada hacia el extremo x=L, gráfica de la temperatura final con upwind-difference y Gamma 0.01

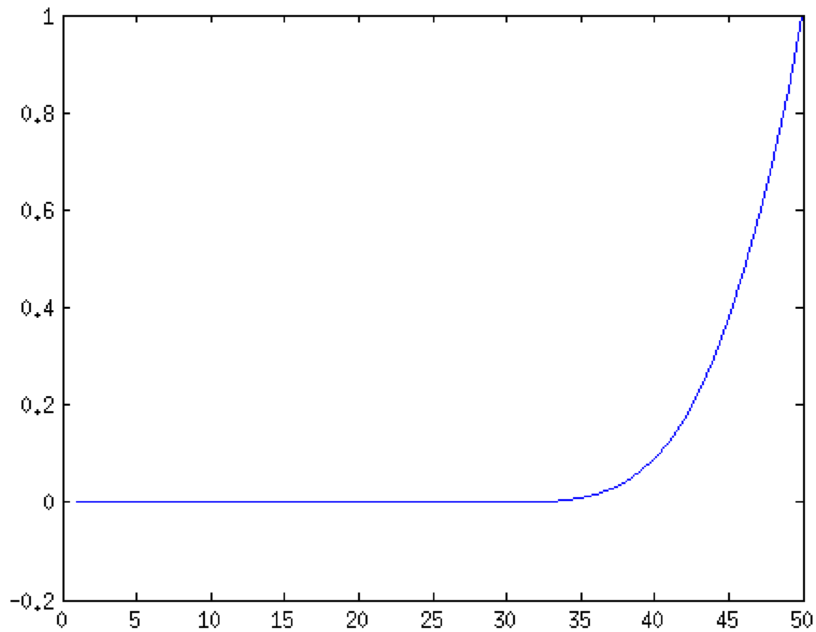

Figura 33: Malla refinada hacia el extremo x=L, gráfica de la temperatura final con central-difference y Gamma 0.01

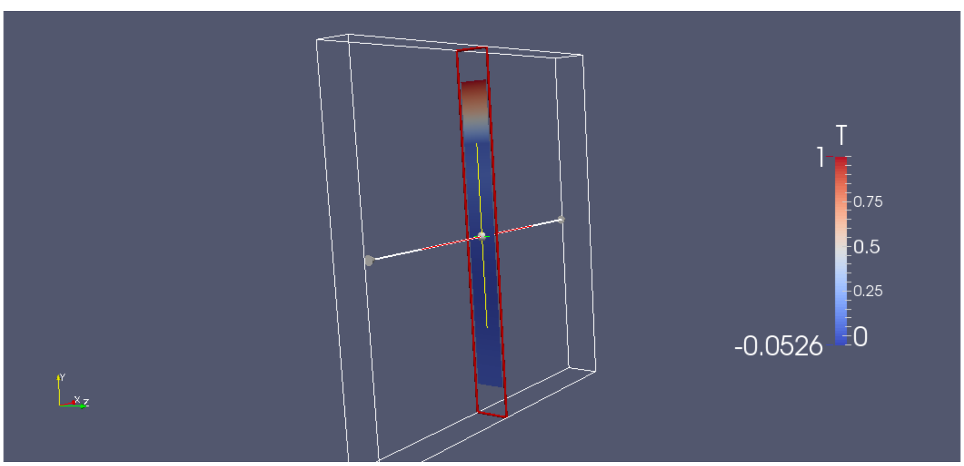

Figura 34: Corte de la solución en x = 1/2, N=40

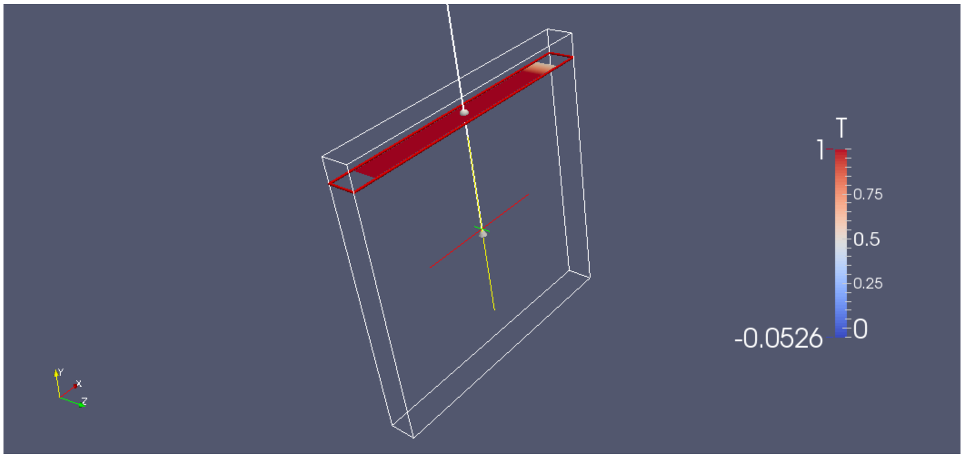

Figura 35: Corte de la solución en y = 1, N=40

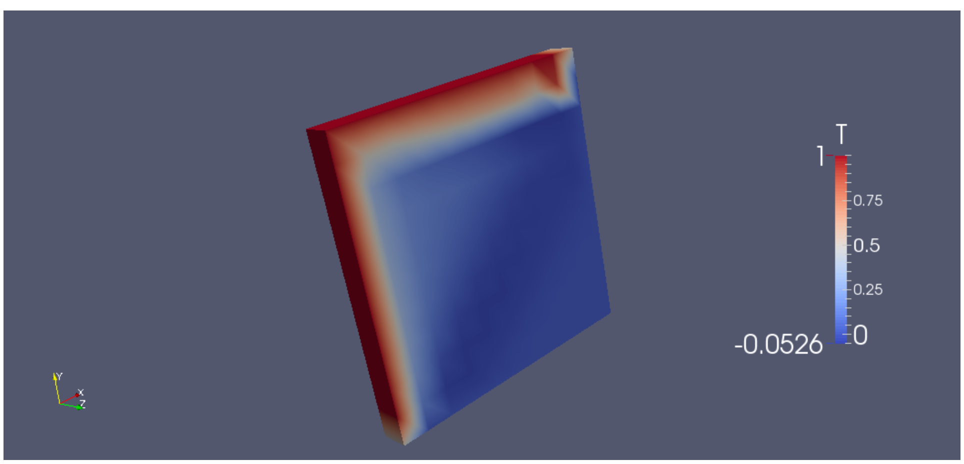

Figura 36: Función ϕ en todo el dominio, N=40

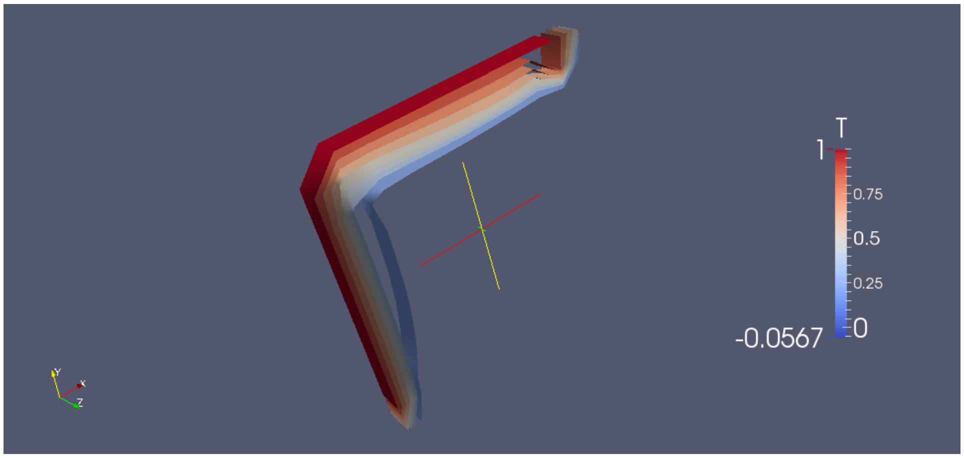

Figura 37: Distintas isotermas de la función ϕ, N=40

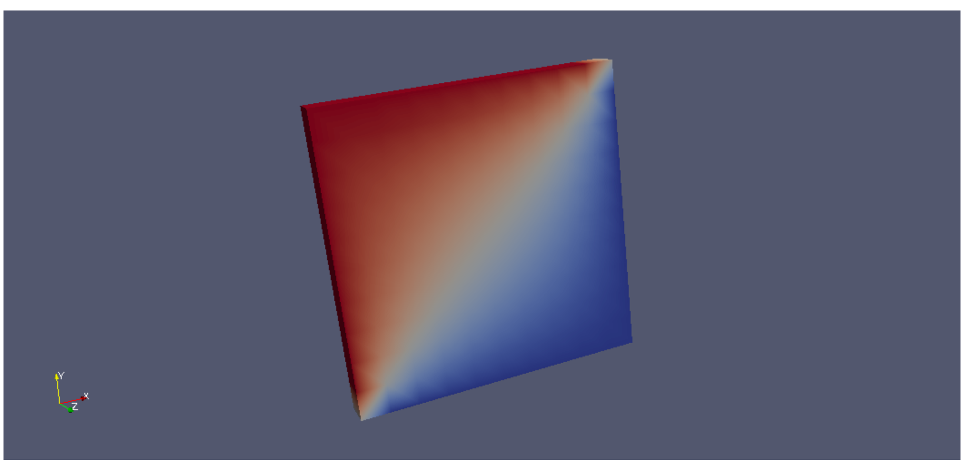

Figura 38: Función ϕ en todo el dominio con Gamma=1, N=40

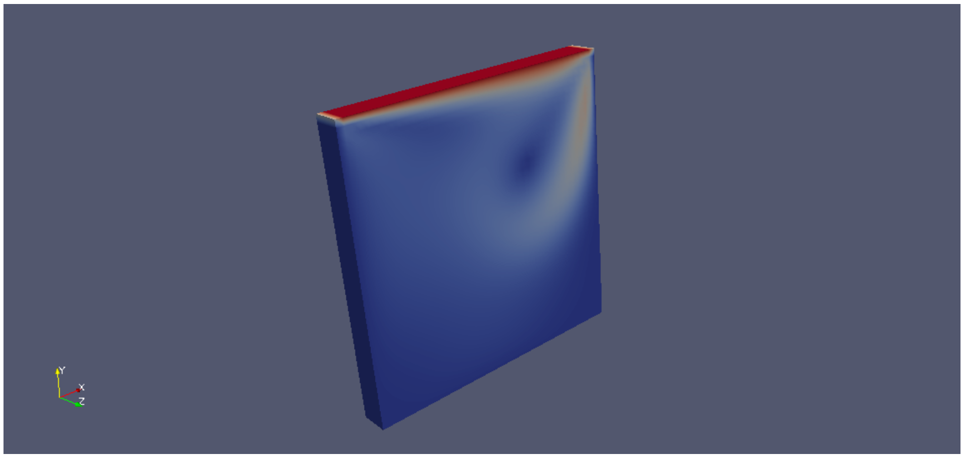

Figura 39: Campo de velocidades obtenido con la solución convergida

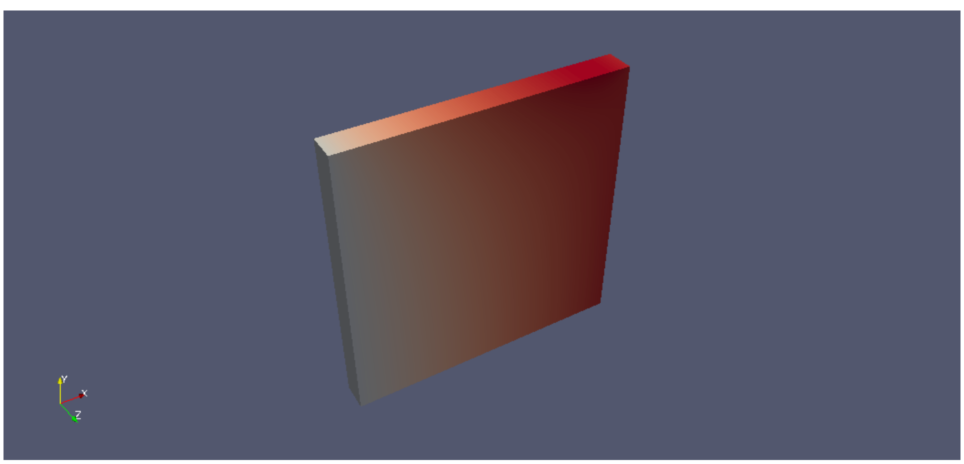

Figura 40: Screenshot del campo de temperaturas distribuido obtenido

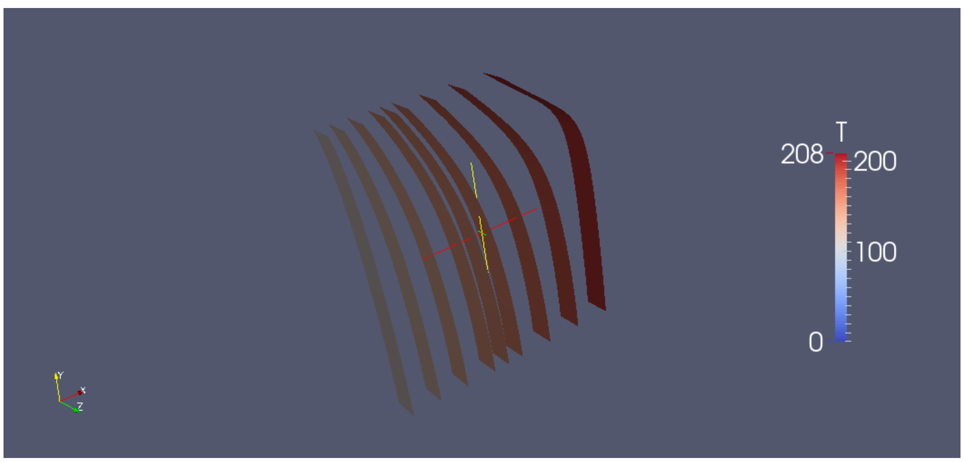

Figura 41: Diferentes isotermas para el campo de temperaturas obtenido