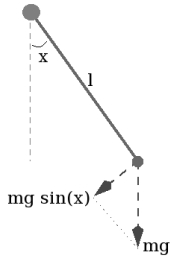
Ecuaciones en Derivadas Parciales y Análisis Numérico

Solución a los problemas de la primera entrega

(17 de Noviembre de 2008)

1 Una ecuación diferencial de segundo orden.

En este problema se os pide resolver numéricamente dos ecuaciones diferenciales que representan el movimiento de un péndulo de masa m sujeto por una varilla rígida de longitud l, con y sin rozamiento. En principio nada impide que el péndulo dé varias vueltas sobre su eje.



Modelizamos el estado del sistema con una única variable: el ángulo x que forma la varilla con la dirección vertical. Para caracterizar el sistema durante un intervalo de tiempo, necesitamos calcular el valor de este ángulo en cada instante x(t). Usando la segunda ley de Newton y la ley de gravedad obtenemos una ecuación diferencial de segundo orden para el péndulo simple:

$$\frac{d^2x}{dt^2} = \frac{-g}{l}\sin(x)$$

1) *Plantea la ecuación como un sistema de primer orden introduciendo la variable velocidad v. Resuelve el sistema de forma aproximada usando el método de matlab* ***ode45*** *en el intervalo [0,10], para x(0)=0, v(0)=1 tomando g=10 y l=1 (para simplificar trabajaremos sin unidades).*

Introducimos la variable velocidad:

$$v = \frac{dx}{dt}$$

Ahora es fácil escribir un sistema de dos ecuaciones diferenciales en las variables *x* y *v*:

$$\frac{dx}{dt} = v$$

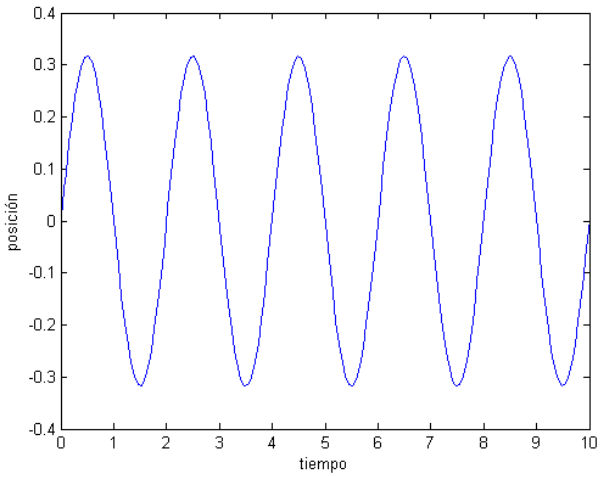
$$\frac{dv}{dt} = \frac{-g}{l}\sin(x)$$

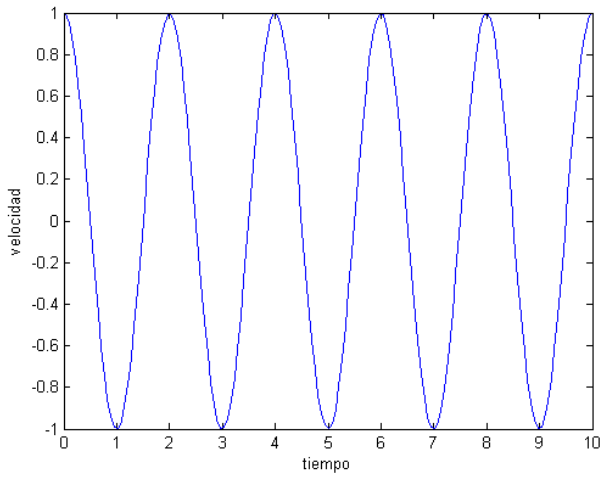
Se trata de un sistema de dos ecuaciones no lineal que podemos intentar resolver con cualquiera de los métodos numéricos para ODES. Para usar el método **ode45** tenemos que crear un archivo, que llamaremos **pendulo.m**, que devuelve las derivadas respecto al tiempo de las variables del sistema x y v, puestas juntas en un vector:

```
def d = pendulo(t,p)
l=1;
g=10;
%El vector p guarda las variables x=p(1) y v=p(2)
%En el vector v ponemos las derivadas dx/dt=d(1) y dv/dt=d(2)
d=[p(2); -(g/l)*sin(p(1))];
```

A continuación invocamos el método ode45 con los parámetros correspondientes a los datos iniciales y el intervalo de tiempo que queremos estudiar:

```
t_0=0;
t_f=10;
p0=[0;1];
[t,p]=ode45('pendulo',[t_0,t_f],p0);
figure;
plot(t,p(:,1))  %dibuja la posición
figure;

plot(t,p(:,2))  %dibuja la velocidad
```





La primera gráfica representa la posición del péndulo en función del tiempo, oscilando entre *-0.3* y *0.3*. La segunda representa la velocidad del péndulo en función del tiempo, oscilando entre *-1* y *1*. Al no haber rozamiento, el movimiento no se detiene. Aunque el movimiento es periódico, estas curvas no son sinusoides. Cuando la posición es pequeña, podemos aproximar *sin(x)* por su polinomio de Taylor de orden 1, que es exactamante *P(x)=x*. Al hacer esta aproximación, obtenemos una ecuación lineal:

$$\frac{d^2x}{dt^2} = \frac{-g}{l}x$$

cuyas soluciones son exactamente sinusoides. Sin embargo, esto es sólo una aproximación, y para valores mayores de la velocidad, obtenemos curvas menos parecidas a curvas sinusoidales.

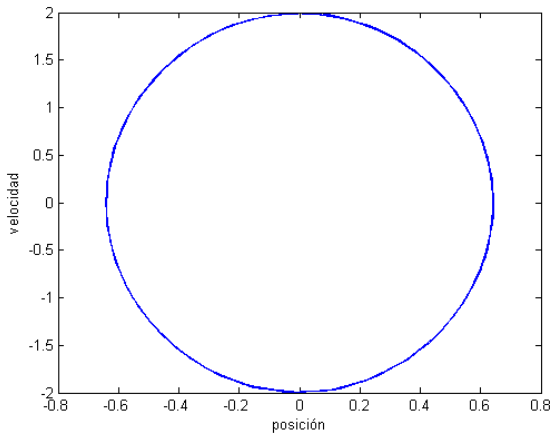
Este sistema de ecuaciones es autónomo, lo que quiere decir que la ecuación no depende explícitamente del tiempo. Por tanto, la trayectoria de un punto depende sólo de la posición y la velocidad inicial, pero no del instante de tiempo inicial. En estos casos es habitual representar las trayectorias en un diagrama que sólo contiene las variables del problema (posición y velocidad) pero no el tiempo (como hicimos en el caso del atractor de Lorentz en el capítulo 2 de las prácticas). Ignorando el hecho de que x es una variable periódica, podemos hacer la representación en el plano x, y. Como verás, tiene sentido permitir que x tome valores arbitrarios, entendiendo que dos puntos cuya coordenada x difiere en más de 2π representan la misma posición del péndulo. Cada vez que el péndulo gira sobre sí mismo pasamos de un intervalo $[k\cdot 2\pi,(k+1)\cdot 2\pi]$ al intervalo siguiente $[(k+1)\cdot 2\pi,(k+2)\cdot 2\pi]$.

2) *Dibuja las trayectorias del punto con condición inicial x(0)=0, v(0)=2 y del punto con condición inicial x(0)=0, v(0)=7. Interpreta el resultado.*

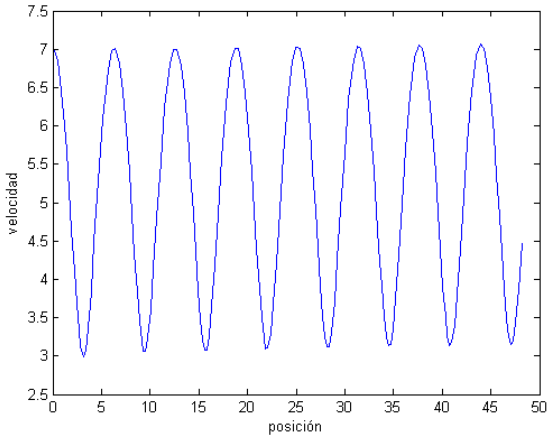
Cambiamos las condiciones iniciales a los nuevos valores, y dibujamos los pares de puntos (x,v), que están guardados como la primera y segunda columnas del vector **p**.

```
t_0=0;
```

```
t_f=10;
%Primera grafica:  x(0)=0, v(0)=2
p0=[0;2];
[t,p]=ode45('pendulo',[t_0,t_f],p0);
figure;
plot(p(:,1),p(:,2))


%Segunda grafica:  x(0)=0, v(0)=7
p0=[0;7];
[t,p]=ode45('pendulo',[t_0,t_f],p0);
figure;
plot(p(:,1),p(:,2))
```



x(0)=0, v(0)=2



x(0)=0, v(0)=7

La primera gráfica muestra la trayectoria del punto x=0, v=2. El péndulo avanza hasta la posición x=0.6, v=0, aproximadamente, y retrocede hasta la posición x=-0.6, v=0, donde de nuevo se invierte el signo de la velocidad, y el péndulo avanza hasta la posición de partida, cerrando la curva que se ve en la gráfica.
En la segunda gráfica, el péndulo tiene suficiente velocidad inicial como para dar vueltas sobre su eje. Aunque vuelve a la misma posición y con la misma velocidad, lo hace después de haber dado varias vueltas completas, por lo que la trayectoria que observamos no vuelve a la misma posición, sino a la posición de partida desplazada en $2n\pi$.

Introducimos ahora en el modelo una fuerza de rozamiento, con lo que la ecuación se queda en:

$$\frac{d^2x}{dt^2}=\frac{-g}{l}\sin(x)-a\frac{dx}{dt}$$

3) *Toma un valor de a igual a 0,5. Dibuja las trayectorias de algunos puntos con distintas condiciones iniciales e interpreta el resultado. Encuentra un valor para la velocidad tal que si le imprimimos esa velocidad al péndulo cuando está en la posición de reposo x=0, el péndulo dé cinco vueltas sobre su eje.*

El cambio en la ecuación de segundo grado se traduce en el siguiente cambio en el sistema de ecuaciones de primer orden con las variables *x* y *v*.

$$\frac{dx}{dt}=v$$

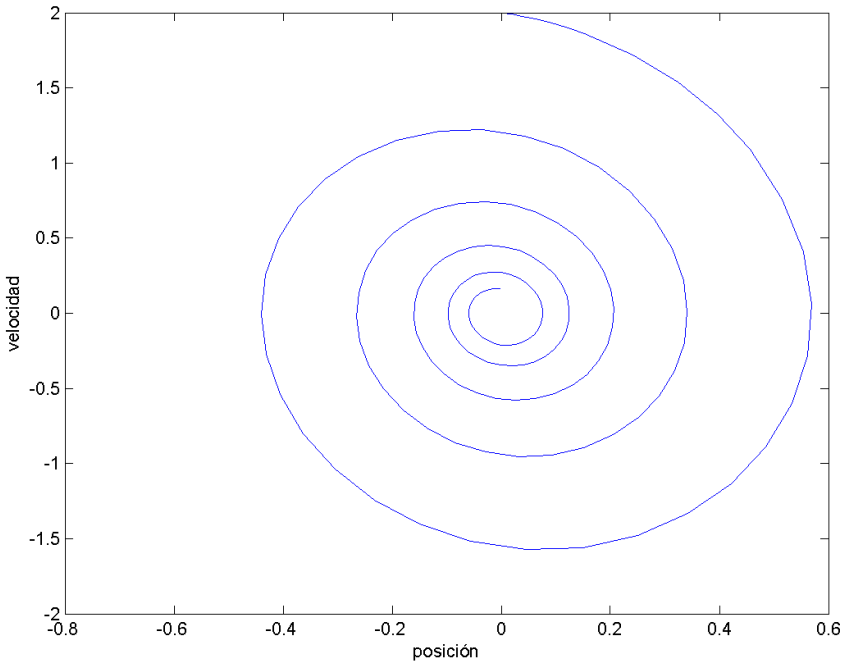
$$\frac{dv}{dt}=\frac{-g}{l}\sin(x)-a\,v$$

Para poder resolver este sistema de ecuaciones de forma aproximada con el método **ode45**, tenemos que crear un nuevo fichero **penduloR.m**, con el código siguiente:

```
def d = penduloR(t,p)
l=1;
g=10;
a=0.5;
%El vector p guarda las variables x=p(1) y v=p(2)
%En el vector v ponemos las derivadas dx/dt=d(1) y dv/dt=d(2)
d=[p(2); -(g/l)*sin(p(1))-a*p(2)];
```

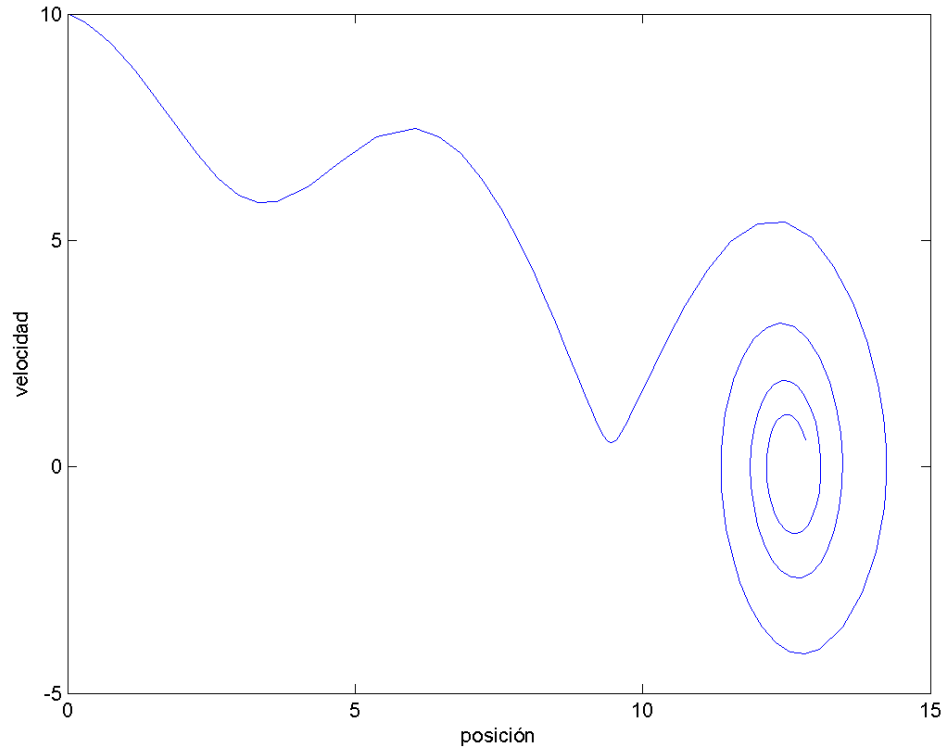
A continuación, llamamos al método **ode45** con distintos valores iniciales, e interpretamos el resultado:

```
t_0=0;
t_f=10;
%Primera grafica:  x(0)=0, v(0)=2
p0=[0;2];
[t,p]=ode45('penduloR',[t_0,t_f],p0);
figure;
plot(p(:,1),p(:,2))
```



Partiendo de esta posición inicial, el péndulo oscila, perdiendo energía mecánica debido al rozamiento. Según aumenta el tiempo, disminuye la amplitud de las oscilaciones y se acerca más a la posición de reposo *x=0*, *v=0*.

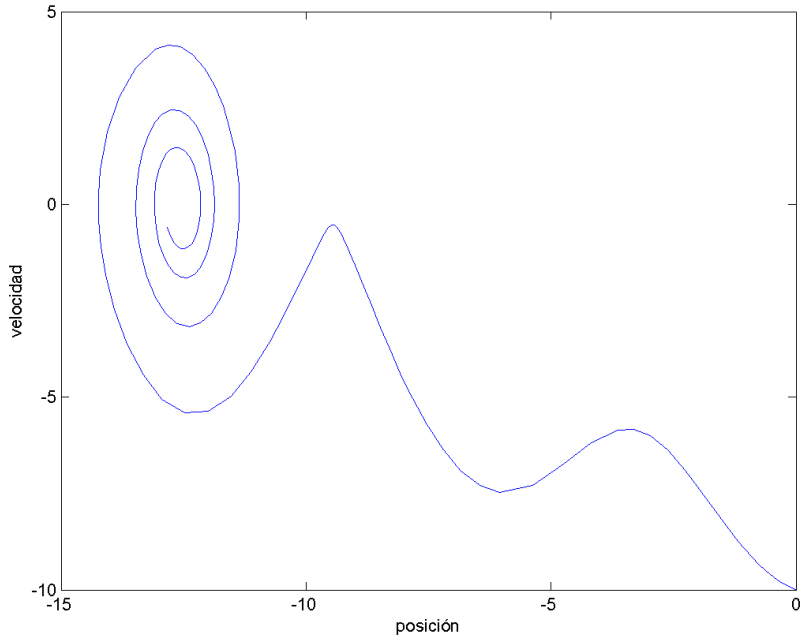
```
%Segunda grafica:  x(0)=0, v(0)=10
p0=[0;10];
[t,p]=ode45('penduloR',[t_0,t_f],p0);
figure;
plot(p(:,1),p(:,2))
```



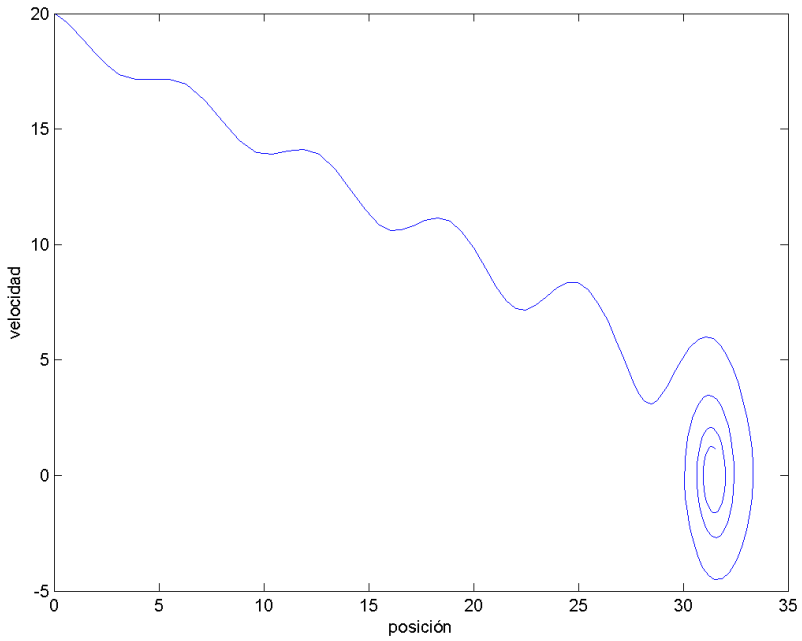
En este caso, el péndulo sobrepasa la posición vertical en dos ocasiones (cuando $x=\pi$ y cuando $x=3\pi$), completando dos vueltas completas alrededor de su eje. Al hacerlo, su energía mecánica disminuye, y deja de dar vueltas, para pasar a oscilar en torno a la posición de reposo, mientras la amplitud de las oscilaciones disminuye y se acerca más a la posición de reposo $x=3\pi, v=0$.

```
%Tercera grafica:  x(0)=0, v(0)=-10
p0=[0;-10];
[t,p]=ode45('penduloR',[t_0,t_f],p0);
figure;
plot(p(:,1),p(:,2))
```

Al comenzar con velocidad negativa, la variable x disminuye. En realidad, esta gráfica es simétrica a la anterior, ya que el sistema es simétrico.



```
%Cuarta grafica:  x(0)=0, v(0)=20
p0=[0;20];
[t,p]=ode45('penduloR',[t_0,t_f],p0);
figure;
plot(p(:,1),p(:,2))
```



Finalmente, probamos distintas velocidades hasta encontrar una (*v(0)=20*) para la que el péndulo da cinco vueltas sobre sí mismo, tal y como nos pedían. Probando todas las condiciones iniciales en el entorno de v(0)=20, encontramos que el péndulo da cinco vueltas para cualquier velocidad inicial entre 19 y 21.9.

4) *Dibuja en el mismo diagrama las trayectorias de varios puntos con la misma posición inicial x(0)=0 pero distinta velocidad inicial. Este tipo de diagrama, al que podríamos llamar un diagrama de fases del sistema dinámico del péndulo, permite observar todo el comportamiento del sistema de un vistazo, y especialmente los atractores del sistema, que son los puntos del diagrama de fases a los que convergen las trayectorias o, en otras palabras, los posibles estados de equilibrio estable del péndulo, en los que una pequeña perturbación devolverá el sistema al mismo estado al cabo de un tiempo. En este ejemplo, los estados de equilibrio estable corresponden a x=0, v=0, que es la posición de reposo.*

Para dibujar el diagrama puedes usar dos métodos:

1. *Los comandos* ***hold on*** *y* ***hold off*** *de matlab permiten dibujar varias gráficas en la misma figura. Por ejemplo, para dibujar varias funciones seno con distintos periodos:*

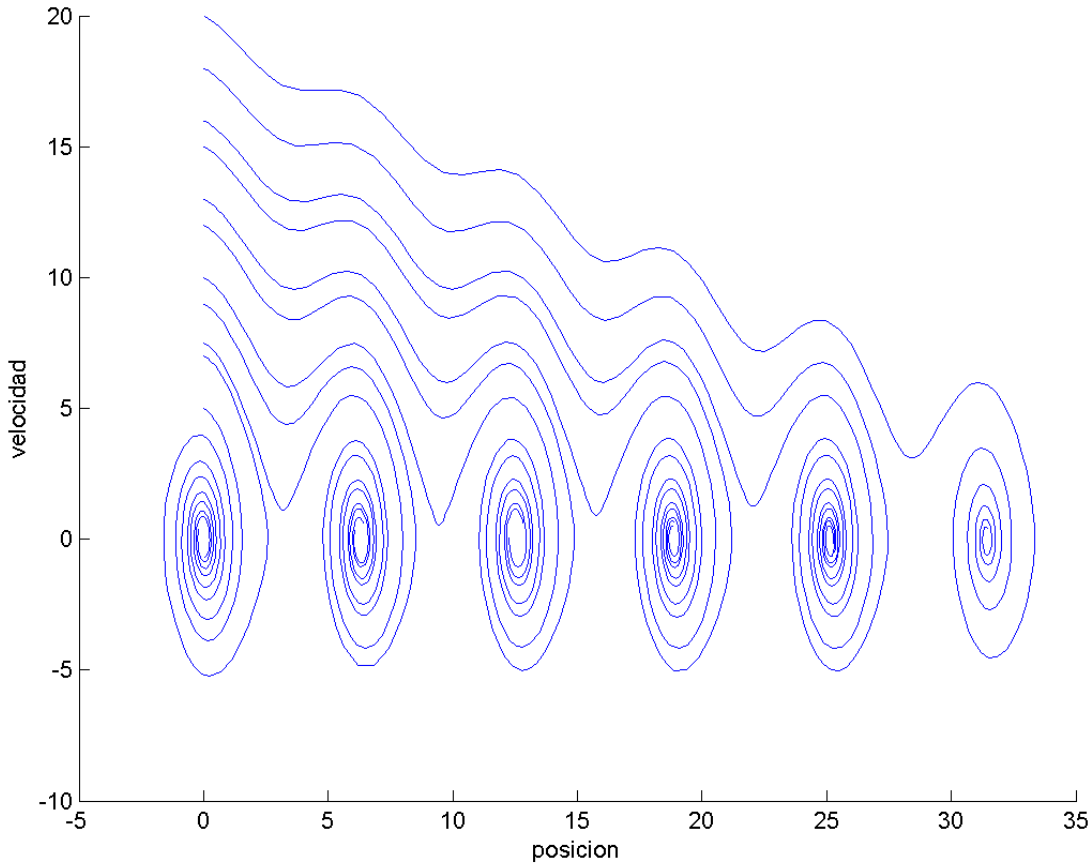
```
t=0:0.1:pi;
hold on
for k=0:3
    plot(t,sin(t*k),'Color',[k/3,1-k/3,0])
end
hold off
```

2. *Si pasas al comando* ***plot*** *dos matrices, dibujará varias gráficas en el mismo diagrama, dibujando las columnas de la primera matriz contra las de la segunda. El siguiente código es equivalente al anterior (salvo por los colores, que ahora los elige automáticamente):*

```
t=0:0.1:3;
plot([t',t',t',t'],[sin(0*t)',sin(1*t)',sin(2*t)',sin(3*t)'])
```

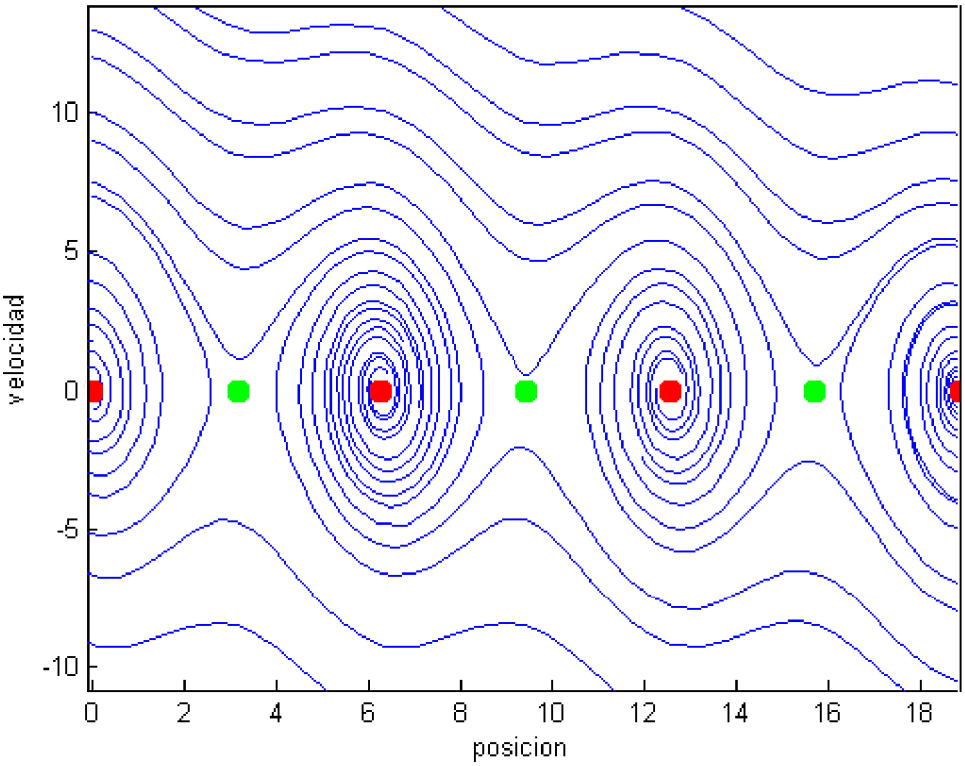
Se trata simplemente de dibujar varias trayectorias en la misma gráfica. Escogemos algunas de las más significativas y modificamos los tiempos para que el dibujo no quede demasiado abigarrado. El único objetivo es que el diagrama muestre los casos más representativos:

```
figure;
hold on;
t_0=0;
tfs=[8, 10, 10, 10, 10, 10, 12, 13, 13, 14, 14];
v0s=[ 5, 7,7.5,  9, 10, 12, 13, 15, 16, 18, 20];
x0s=zeros(length(tfs) );
for i = 1:length(tfs)
    [t,p]=ode45('penduloR',[t_0,tfs(i)],[x0s(i),v0s(i)]);
    plot(p(:,1),p(:,2))
end
hold off;
xlabel('posicion')
ylabel('velocidad')
saveas(gcf,'diagrama1','png')
```



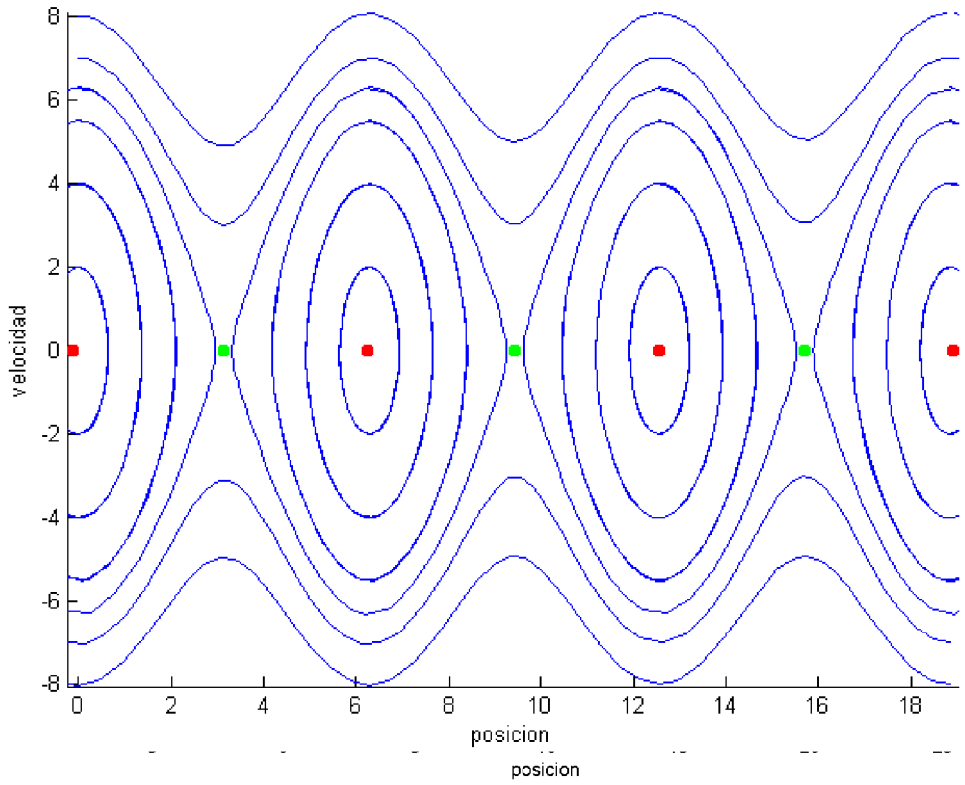
Observamos que algunas trayectorias muy próximas divergen al acercarse a los puntos de la forma $x=(2n+1)\pi, v=0$. Todos estos puntos corresponden al mismo estado del péndulo, puesto de pie a velocidad cero. Este punto es un equilibrio, pues el péndulo puede permanecer en esa posición indefinidamente, pero es un equilibrio inestable, porque una perturbación muy pequeña llevará al péndulo a un lado o al otro.

Podemos añadir algunas trayectorias con velocidad negativa para redondear el diagrama, y ya de paso resaltar los puntos de equilibrio La gráfica siguiente se ha obtenido dibujando algunas trayectorias con velocidad negativa, haciendo zoom en la gráfica y añadiendo los puntos de equilibrio estable en rojo y los de equilibrio inestable en verde.



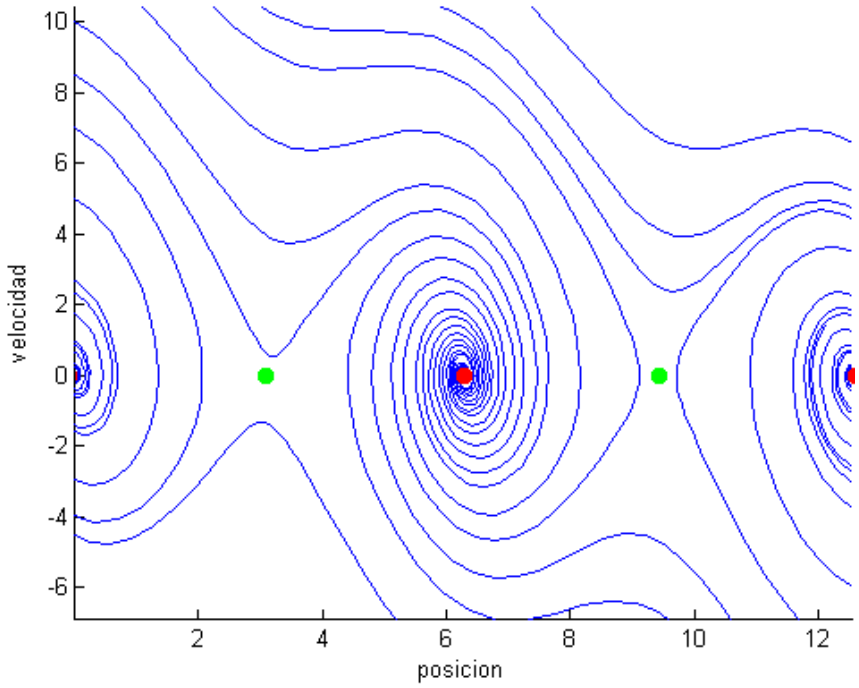
Como extra, comparamos el diagrama de fases del péndulo con rozamiento con un diagrama similar para el péndulo sin rozamiento. Usando un código similar al anterior y retocando un poco la imagen obtenemos un diagrama de fases para el péndulo sin rozamiento.

```
figure;
hold on;
t_0=0;
tfs=[ 5, 5,   5,   5,    5,    5,    5,    5,    5,    5,    5,    5,    5,    5,    5, 4.5,  4.5,  3.5,  3.5];
x0s=[ 0, 0,   0,   0, 2*pi, 2*pi, 2*pi, 2*pi, 4*pi, 4*pi, 4*pi, 4*pi, 6*pi, 6*pi, 6*pi, 6*pi,    0, 6*pi,    0, 6*pi];
v0s=[ 2, 4, 5.5, 6.3,    2,    4,  5.5,  6.3,    2,    4,  5.5,  6.3,    2,    4,  5.5,  6.3,   7,   -7,    8,   -8];
for i = 1:length(tfs)
    [t,p]=ode45('pendulo',[t_0,tfs(i)],[x0s(i),v0s(i)]);
    plot(p(:,1),p(:,2))
end
hold off;
xlabel('posicion')
ylabel('velocidad')
saveas(gcf,'diagrama2','png')
```



Observamos que cuando la velocidad inicial (desde la posición de reposo), es menor que 6.32 aproximadamente, las trayectorias con cerradas. Esta es la velocidad que hace que la energía mecánica inicial sea igual a la energía potencial en el punto más alto del péndulo. Cuando la energía mecánica es mayor, el sistema también vuelve periódicamente a la misma posición, pero la trayectoria que vemos no es cerrada porque el desplazamiento sobre el eje x muestra el número de vueltas acumuladas.

Finalmente, lo comparamos con el diagrama de un péndulo con más rozamiento ($a=1$). Observamos que en este caso, las trayectorias pierden velocidad mucho antes, y se acercan a los estados de equilibrio del péndulo mucho antes.



2 Diferencias finitas con un método de Euler implícito para la ecuación del calor.

En este problema se os pide realizar dos modificaciones al código del capítulo 3.1 de los apuntes:

1. *Cambia las condiciones de frontera a condiciones no homogéneas:*

$$u(x_0)=a\,,\,u(x_{M+1})=b$$

Prueba el código generado para distintos valores de a y de b y para distintas condiciones iniciales. Observa la evolución de la solución durante un periodo de tiempo suficientemente largo y conjetura cuál será la distribución de temperatura en la barra en el límite para tiempos grandes.

Para poder tener en cuenta las condiciones de frontera tenemos que repasar nuestra derivación del método de diferencias finitas. Para aproximar la derivada u_{xx} en los puntos de frontera es necesario utilizar las condiciones de frontera. Para el extremo izquierdo:

$$u_{xx}(x_2)\approx\frac{1}{h^2}(u(x_1-h)-2u(x_1)+u(x_1+h))\approx\frac{1}{h^2}((\hat{u})_1-2(\hat{u})_2+(\hat{u})_3)=\frac{1}{h^2}(-2(\hat{u})_2+(\hat{u})_3)+\frac{1}{h^2}a$$

y para el extremo derecho:

$$u_{xx}(x_M)\approx\frac{1}{h^2}(u(x_{M-1}-h)-2u(x_M)+u(x_{M+1}+h))$$

$$...\approx\frac{1}{h^2}((\hat{u})_{M-1}-2(\hat{u})_M+(\hat{u})_{M+1})=\frac{1}{h^2}((\hat{u})_{M-1}-2(\hat{u})_M)+\frac{1}{h^2}b$$

De modo que la ecuación diferencial para $\hat{u}$ se escribe en forma matricial:

$$\frac{d\hat{u}}{dt}=k(D\hat{u}+c)$$

donde la matriz D es:

$$D=\frac{1}{h^2}\begin{pmatrix}-2 & 1 & ... & ... & . \\ 1 & -2 & 1 & ... & . \\ ... & ... & ... & ... & ... \\ . & ... & ... & 1 & -2\end{pmatrix}$$

y el vector *c* es:

$$c=\frac{1}{h^2}\begin{pmatrix}a\\0\\\vdots\\0\\b\end{pmatrix}$$

Por lo tanto, para implementar un método de Euler explícito, basta hacer:

$$\hat{u}^{n+1}=\hat{u}^n+\Delta t\left(\frac{d\hat{u}}{dt}\right)^n=\hat{u}^n+\Delta t\;k(D\hat{u}^n+c)=(I+\Delta t\;kD)\hat{u}^n+\Delta t\;kc$$

Modificamos el código del capítulo 3.1:

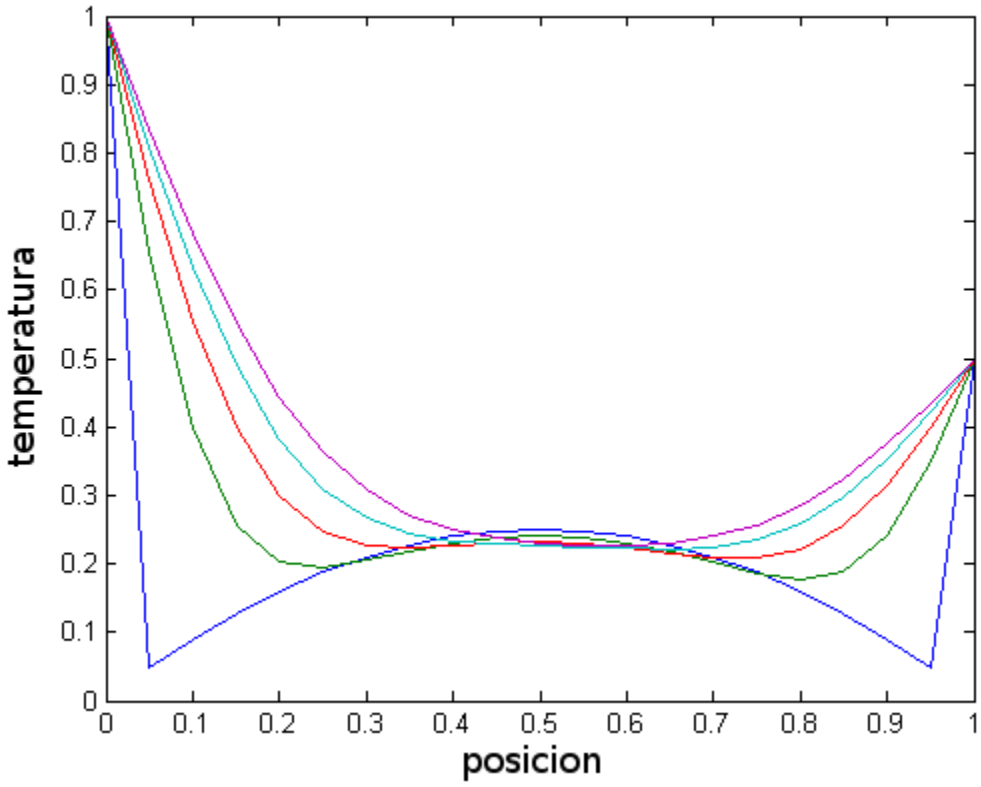
```
function [u_save,x,t_save]=calorFrontera(k, M, t_f, tsteps)
%function [u_save,x,t_save]=calorFrontera(k, M, t_f, tsteps)
%
%resuelve la ecuacion del calor u_t = k u_xx en [0,1]
%con condiciones de contorno tipo Dirichlet u(0,t)=a; u(1,t)=b
%
%Datos de entrada:
% k: coeficiente k de la ecuacion
% M: numero de partes iguales en que se descompone [0,1]
% t_f: instante de tiempo hasta el que calculamos la solucion
% tsteps: numero de partes iguales en que se descompone [0,t_f]

%Mallado
dt=t_f/tsteps;
tiempos=0:dt:t_f;
h=1/M;
x=0:h:1;

%Condiciones de contorno:
a=1;
b=0.5;

%Datos iniciales
xred=x(2:M); %quitamos los extremos del vector x
u=(xred.*(1-xred))'; %trasponemos para obtener un vector columna

%El operador D
%Usamos el comando diag(vector,k) para crear una matriz tridiagonal
Tridiag=diag(-2*ones(M-1,1))+diag(ones(M-2,1),1)+diag(ones(M-2,1),-1);
D=(1/h^2)*Tridiag;

%El vector c (vector columna de longitud M-1)
c=zeros(M-1,1);
c(1)=(1/h^2)*a;
c(M-1)=(1/h^2)*b;

%Datos de salida: en vez de guardar todos los datos intermedios,
guardamos el
%vector u solo Nframes veces.
Nframes=5;
marca=floor(tsteps/(Nframes-1));
u_save=zeros(M+1,Nframes);
t_save=zeros(1,Nframes);
%Le ponemos las condiciones de frontera
u_save(1,:)=a*ones(1,Nframes);
u_save(M+1,:)=b*ones(1,Nframes);
%guardamos la posicion de partida
u_save(2:M,1)=u;
t_save(1)=0;

%Bucle principal
I=eye(M-1);
A=(I+dt*k*D);
v=dt*k*c;
for n=1:tsteps
```

```
    u=A*u+v;
    %Guardamos los valores de u para algunos tiempos
    if mod(n,marca)==0
       indice=1+n/marca;
      u_save(2:M,indice)=u;
      t_save(indice)=tiempos(n);
    end
end
```

Llamamos a la función anterior y dibujamos el resultado:

```
[u,x,t]==calorFrontera(1, 20, 0.02, 1000)
plot(x,u)
```

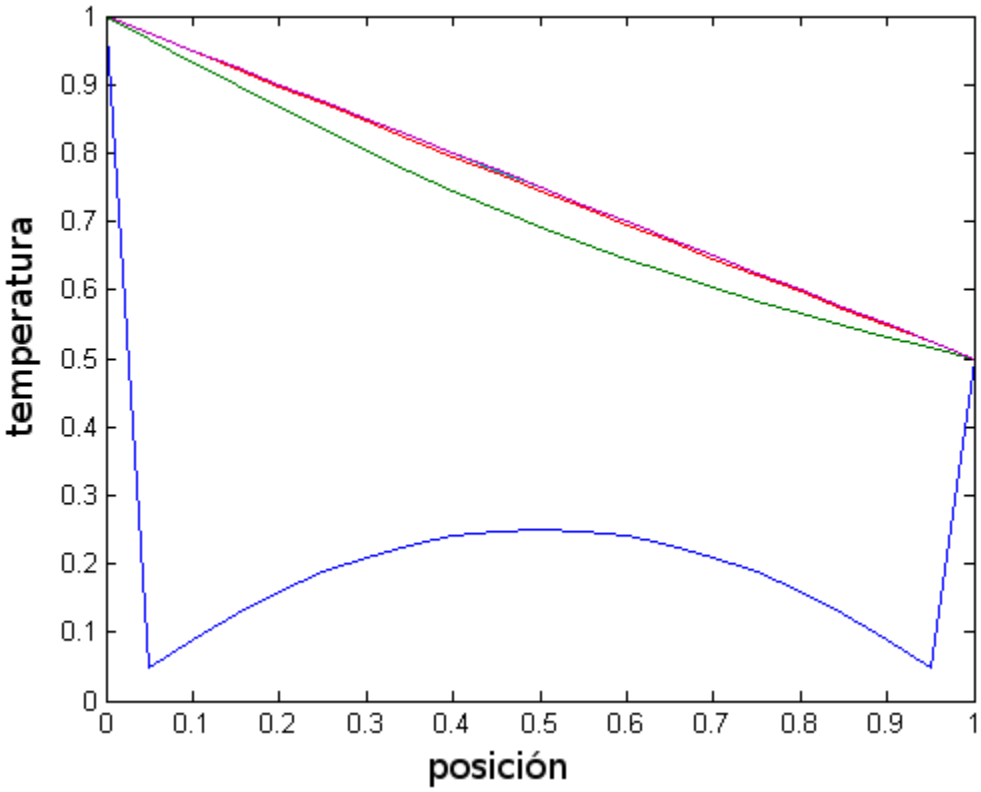
y obtenemos la gráfica:



Observamos que el dato inicial no verificaba las condiciones de frontera. A pesar de ello, el método nos devuelve aproximaciones razonables a la solución de la ecuación del calor. Observamos que la temperatura en un punto tiende siempre a equilibrarse con la de sus vecinos. Aumentando el intervalo de tiempo:

```
[u,x,t]==calorFrontera(1, 20, 1, 1000)
plot(x,u)
```

obtenemos la gráfica:



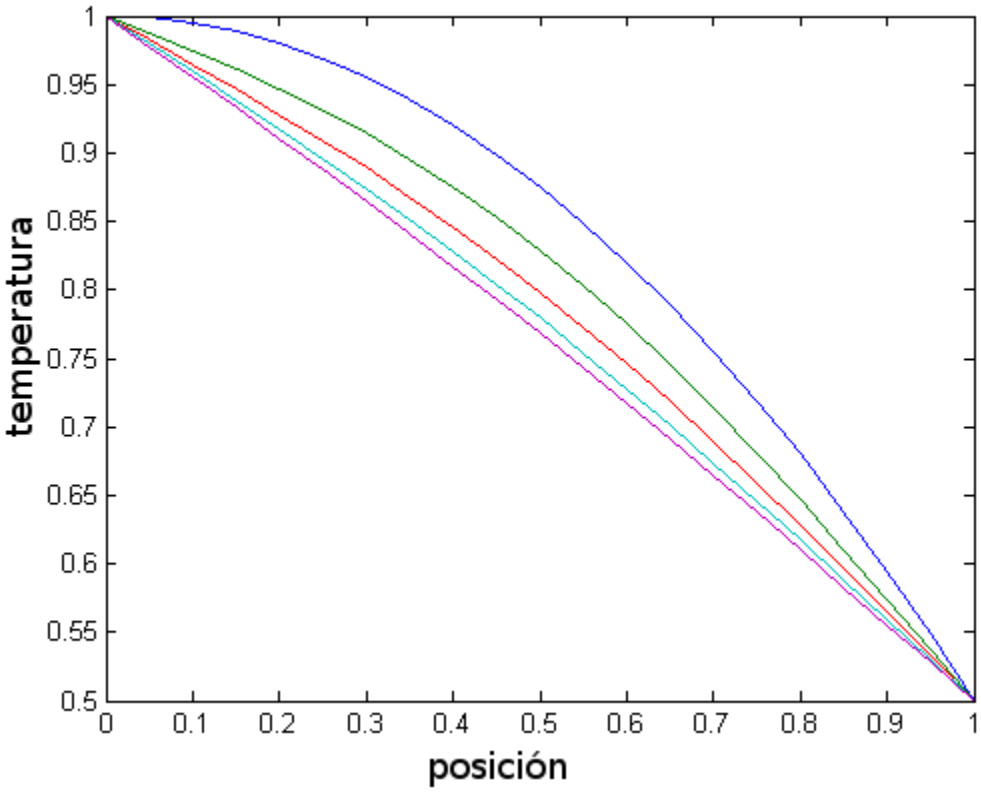
en la que se aprecia que para tiempos largos la solución tiende a estabilizarse formando la línea recta que une los dos valores de la temperatura en la frontera. En efecto, este estado final es un equilibrio porque la temperatura en un punto es igual al promedio de la temperatura de sus vecinos. Sin embargo, al haber un gradiente de temperatura, hay un flujo de calor desde el extremo izquierdo, más caliente, al extremo derecho, más frío.

Modificamos el código del programa **calorFrontera.m** para poner una condición inicial $u(0,x)=1-0.5\,x^2$ y datos de frontera *a=1* y *b=0.5*.

```
%Condiciones de contorno:
a=1;
b=0.5;

%Datos iniciales
xred=x(2:M); %quitamos los extremos del vector x
u=(1-xred.^2)'; %trasponemos para obtener un vector columna
```

Al llamar al nuevo código obtenemos:

```
[u,x,t]==calorFrontera(1, 20, 0.2, 1000)
plot(x,u)
```



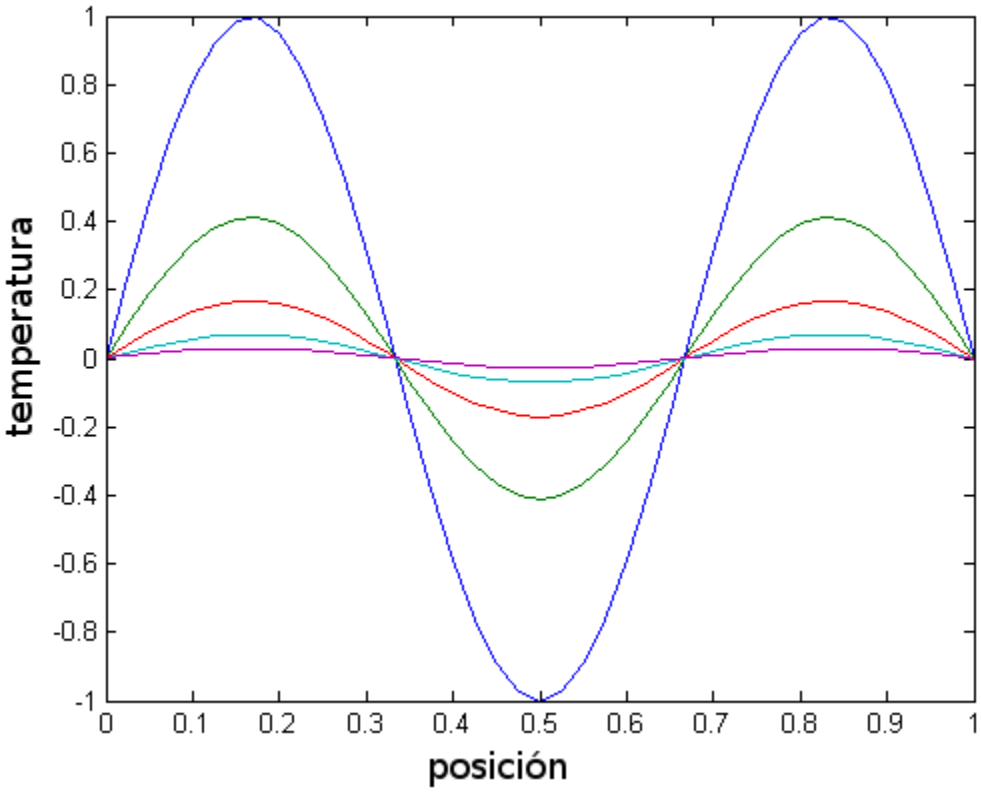
De nuevo observamos que la distribución de temperatura para tiempos muy grandes es muy próxima a una línea recta.

Como último ejemplo, tomamos una condición inicial $u(0,x)=\sin(3\pi x)$ y datos de frontera $a=0$ y $b=0$.

```
%Condiciones de contorno:
a=0;
b=0;

%Datos iniciales
xred=x(2:M); %quitamos los extremos del vector x
u=sin(3*pi*xred)'; %trasponemos para obtener un vector columna
```

Llamemos al nuevo código. Hemos aumentado el número de pasos espaciales porque la solución no tenía suficiente precisión.

```
[u,x,t]==calorFrontera(1, 40, 0.4, 1000)
plot(x,u)
```



Una vez más observamos que la distribución inicial de temperatura, en azul, se suaviza hasta quedar prácticamente una línea recta entre los valores de frontera (curva en violeta).

2. *Modifica el código para que implemente el método de Euler implícito en vez del método de Euler explícito para resolver la EDO resultante. Comprueba que el método no desarrolla inestabilidad aunque dt no sea del orden de h^2.*

Para utilizar el método implícito sólo tenemos que avanzar en cada paso con la derivada evaluada en el punto siguiente. Si las condiciones de frontera son homogéneas:

$$\hat{u}^{n+1} = \hat{u}^n + \Delta t \left(\frac{d\hat{u}}{dt}\right)^{n+1} = \hat{u}^n + \Delta t \; k \; D\hat{u}^{n+1}$$

y despejando el vector $\hat{u}^{n+1}$:

$$(I - \Delta t \; k D)\hat{u}^{n+1} = \hat{u}^n$$

$$\hat{u}^{n+1} = (I - \Delta t \; k D)^{-1}(\hat{u}^n)$$

Y si no son homogéneas:

$$\hat{u}^{n+1} = \hat{u}^n + \Delta t \left(\frac{d\hat{u}}{dt}\right)^{n+1} = \hat{u}^n + \Delta t \; k(D\hat{u}^{n+1} + c) = \hat{u}^n + \Delta t \; k D\hat{u}^{n+1} + \Delta t \; k c$$

y despejando el vector $\hat{u}^{n+1}$:

$$(I - \Delta t \; k D)\hat{u}^{n+1} = \hat{u}^n + \Delta t \; k c$$

$$\hat{u}^{n+1}=(I-\Delta t\ k\,D)^{-1}(\hat{u}^{n}+\Delta t\ k\,c)$$

Escribimos el código que implementa el método implícito con condiciones no homogéneas *a=1*, *b=0*, y con dato inicial $u(0,x)=\cos(\frac{x\,\pi}{2})$:

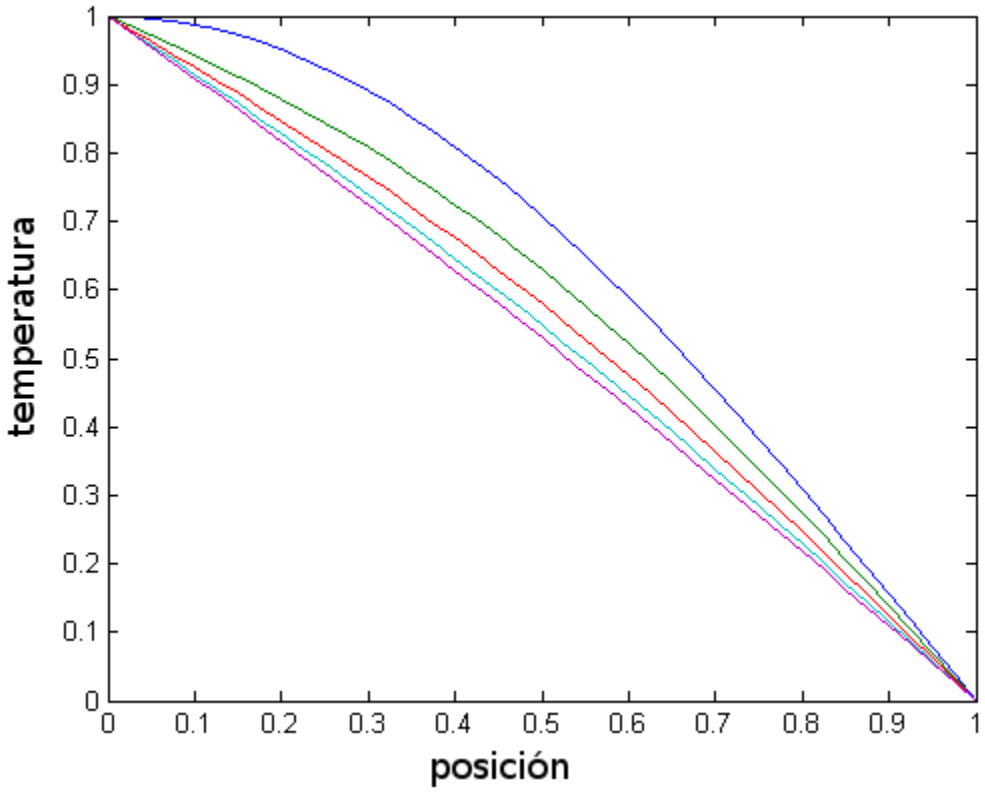
```
function [u_save,x,t_save]=calorImplicito(k, M, t_f, tsteps)
%function [u_save,x,t_save]=calorImplicito(k, M, t_f, tsteps)
%
%resuelve la ecuacion del calor u_t = k u_xx en [0,1]
%con dato inicial cos(x*pi/2)
%y condiciones de contorno tipo Dirichlet u(0,t)=a; u(1,t)=b
%usando el metodo de Euler implicito
%
%Datos de entrada:
% k: coeficiente k de la ecuacion
% M: numero de partes iguales en que se descompone [0,1]
% t_f: instante de tiempo hasta el que calculamos la solucion
% tsteps: numero de partes iguales en que se descompone [0,t_f]

%Condiciones de contorno:
a=1;
b=0;

%Mallado
dt=t_f/tsteps;
tiempos=0:dt:t_f;
h=1/M;
x=0:h:1;

%Datos iniciales
xred=x(2:M); %quitamos los extremos del vector x
u=cos(x*pi/2)'; %trasponemos para obtener un vector columna

%El operador D
%Usamos el comando diag(vector,k) para crear una matriz tridiagonal
Tridiag=diag(-2*ones(M-1,1))+diag(ones(M-2,1),1)+diag(ones(M-2,1),-1);
D=(1/h^2)*Tridiag;

%El vector c (vector columna de longitud M-1)
c=zeros(M-1,1);
c(1)=(1/h^2)*a;
c(M-1)=(1/h^2)*b;

%Datos de salida: en vez de guardar todos los datos intermedios,
guardamos el
%vector u solo Nframes veces.
Nframes=5;
marca=floor(tsteps/(Nframes-1));
u_save=zeros(M+1,Nframes);
t_save=zeros(1,Nframes);
%Le ponemos las condiciones de frontera
u_save(1,:)=a*ones(1,Nframes);
u_save(M+1,:)=b*ones(1,Nframes);
%guardamos la posicion de partida
```

```
u_save(2:M,1)=u;
t_save(1)=0;

%Bucle principal
I=eye(M-1);
A=inv(I-dt*k*D);
v=dt*k*c;
for n=1:tsteps
    u=A*(u+v);
    %Guardamos los valores de u para algunos tiempos
    if mod(n,marca)==0
       indice=1+n/marca;
      u_save(2:M,indice)=u;
      t_save(indice)=tiempos(n);
    end
end
```

Llamamos a la función anterior y dibujamos el resultado:

```
[u,x,t]=calorImplicito(0.2, 40, 1, 1000)
plot(x,u)
```

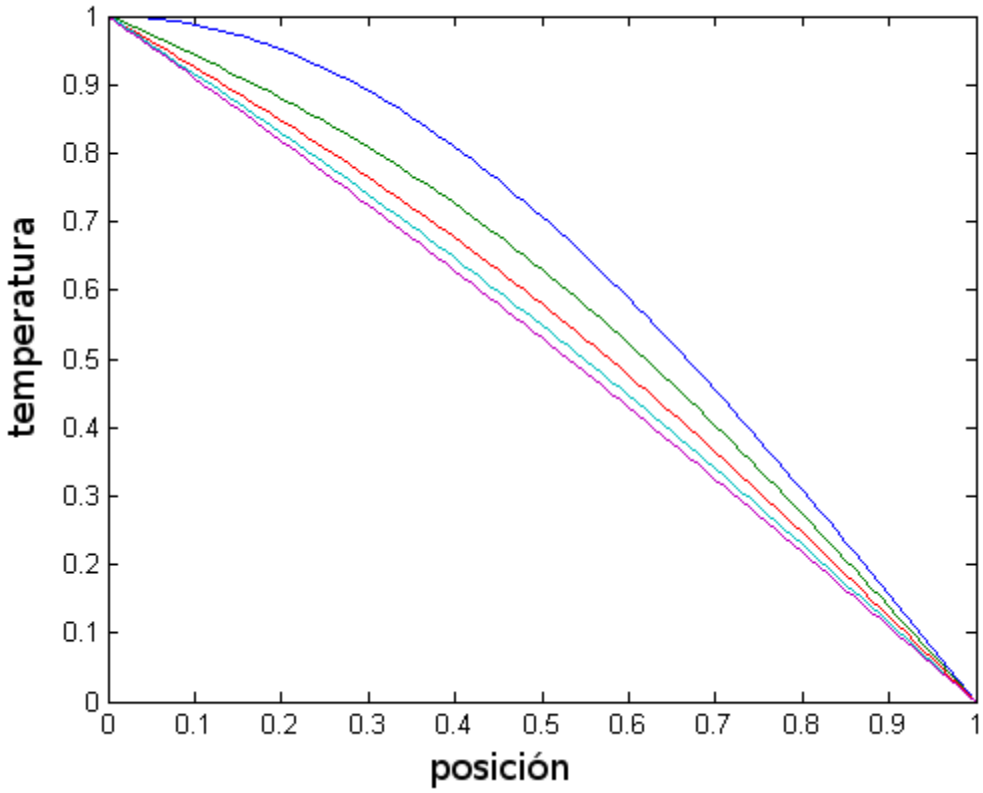
y obtenemos la gráfica:



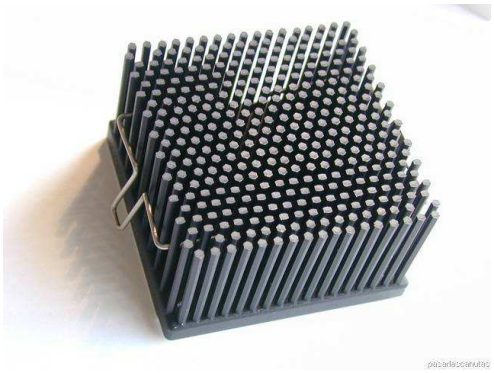
En el caso anterior se verificaba la condición de estabilidad para el método explícito: $s = k\frac{dt}{h^2} < \frac{1}{2}$. Sin embargo, esta condición ya no es necesaria. Aunque llamemos al método con un paso temporal del orden del paso espacial, el método no muestra inestabilidad. Por ejemplo, con los siguientes datos, tenemos k=0.2, dt=0.01, h=0.01.

```
[u,x,t]=calorImplicito(0.2, 100, 1, 100)
plot(x,u)
```

y comprobamos que el resultado es estable (aunque seguramente aproxima la solución con menor precisión):



3 Simulación del disipador de un microprocesador (opcional).



Vamos a intentar modelizar la temperatura de un disipador para microprocesador, compuesto de una serie de barras paralelas sujetas por un extremo al micro. Aunque se trata de un problema tridimensional, haremos algunas simplificaciones para poder tratarlo como un problema unidimensional. En primer lugar, asumimos que la base común a todas las barras de metal se calienta de modo uniforme. Además, suponemos que el disipador está acoplado a un ventilador que mantiene constante la temperatura del aire cercano a las barras, y asumimos que esta temperatura es la misma en cada barra. De este modo, para estudiar la disipación de calor podemos limitarnos a estudiar una barra aislada, para después aproximar el comportamiento del disipador multiplicando por el número de barras.

Continuamos analizando el modelo, haciendo suposiciones sobre el comportamiento de cada barra que nos permitan escribir una ecuación diferencial. Cada barra pierde calor a lo largo de toda su longitud, traspasando este calor al aire. La magnitud de calor que se disipa en un instante de tiempo es proporcional a la diferencia de temperatura entre la barra y el aire. Ignoramos el calor que se disipa por la cara expuesta de la barra. La actividad del micro genera calor que llega a la base común a las barras del disipador. Este calor se reparte a partes iguales entre todas las barras.

Con estas hipótesis ya es posible plantear un modelo. Representamos la barra mediante un intervalo unidimensional [0,L], donde L es la longitud de la barra. El extremo 0 está unido al micro, y el otro extremo queda libre. En el cuerpo de la barra la temperatura responde a la ley de Fourier y a la ley de enfriamiento de Newton:

$$\rho c u_t = k u_{xx} - r(u - u_{Ext})$$

En la fórmula anterior, el coeficiente r depende de la capacidad de transferencia térmica entre el aire y el material elegido para las barras y u_{Ext} es la temperatura exterior.

En el extremo libre no hay flujo de calor, mientras que por el extremo unido al micro entra una cantidad de calor igual a (1/B)f(t), donde B es el número de barras y f(t) es la cantidad de calor generada por el micro en el instante de

tiempo t. Esta cantidad de calor se toma como dato externo del problema.

$$\frac{du}{dx}(x_0, t) = \frac{-1}{B} f(t), \frac{du}{dx}(x_f, t) = 0$$

Cuando el micro no está trabajando, f(t) es 0, de modo que para estudiar el comportamiento del disipador con el micro apagado obtenemos la ecuación:

$$\rho c u_t = k u_{xx} - r(u - u_{Ext})$$
$$\frac{du}{dx}(x_0, t) = 0, \frac{du}{dx}(x_f, t) = 0$$

Sin embargo, cuando el procesador opera a pleno rendimiento, f(t) es una cantidad constante de calor que llamamos F, de modo que para estudiar el comportamiento del disipador cuando el micro opera a pleno rendimiento obtenemos la ecuación:

$$\rho c u_t = k u_{xx} - r(u - u_{Ext})$$
$$\frac{du}{dx}(x_0, t) = \frac{-F}{B}, \frac{du}{dx}(x_f, t) = 0$$

En un ejemplo concreto aceptamos los siguientes valores de los parámetros (todos en unidades del Sistema Internacional):

$$u_{Ext} = 20\,^oC,\ L = 0.03\,m,\ k = 400\left(\frac{W}{m\,^oK}\right), c = 0.38\left(\frac{Julios}{g\,^oK}\right),$$
$$B = 300,\ F = 3 \cdot 10^5 (^oK/m),\ \rho = 9 \cdot 10^6 \left(g/m^3\right), r = 120000\left(\frac{Julios}{s\,m^3\,^oK}\right)$$

1) Después de un tiempo trabajando, la alarma de temperatura salta y se detiene el procesador. La temperatura se distribuye según la función:

$$u_0(x) = 20 + 120\cosh(17x) - 57\sinh(17x)$$

Dejamos reposar el procesador durante 10 segundos. Dibuja la distribución de temperatura en una de las barras al cabo de ese tiempo.

Comenzamos por encontrar un método numérico que aproxime esta ecuación. Tras hacer la discretización espacial (al igual que en el capítulo 3.4), tenemos una ecuación del tipo:

$$\rho c \frac{d\hat{u}}{dt} = F(\hat{u})$$

donde $F(\hat{u})$ aproxima al término $k u_{xx} - r(u - u_{Ext})$. Al tener las mismas condiciones de frontera que en el capítulo 3.4, podemos aproximar u_{xx} con la misma matriz D que usábamos entonces:

$$D = \frac{1}{h^2}\begin{pmatrix} -2 & 2 & \ldots & \ldots & . \\ 1 & -2 & 1 & \ldots & . \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ . & \ldots & \ldots & 2 & -2 \end{pmatrix}$$

Respecto al segundo término, lo podemos aproximar por $r(\hat{u} - \overrightarrow{u_{Ext}})$, donde $\overrightarrow{u_{Ext}}$ es un vector de tamaño *(M+1)x1* con todos sus elementos iguales a u_{Ext}. De este modo, tenemos una ecuación del tipo:

$$\rho c \frac{d\hat{u}}{dt} = k D \hat{u} - r(\hat{u} - \overrightarrow{u_{Ext}})$$

Por lo tanto, para implementar un método de Euler explícito, basta hacer:

$$\hat{u}^{n+1} = \hat{u}^n + \Delta t \left(\frac{d\hat{u}}{dt}\right)^n = \hat{u}^n + \frac{\Delta t}{\rho c}\left(k(D\hat{u}^n) - r(\hat{u} - \overrightarrow{u_{Ext}})\right) = \left(I + \frac{\Delta t\ k}{\rho c} D - \frac{\Delta t\ r}{\rho c} I\right)\hat{u}^n + \frac{\Delta t\ r}{\rho c}\overrightarrow{u_{Ext}}$$

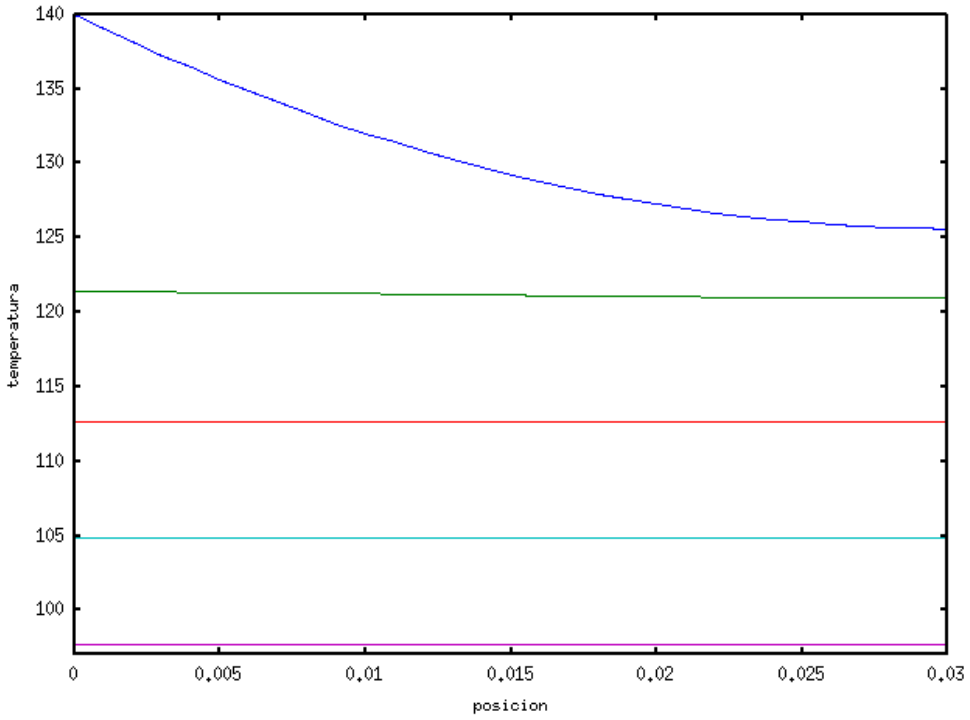
El siguiente código implementa el esquema:

```
function [u_save,x,t_save]=disipador(M, t_f, tsteps)
%function [u_save,x,t_save]=disipador(M, t_f, tsteps)
%
%simula la evolucion de la temperatura en las barras de un disipador
%
%Datos de entrada:
% M: numero de partes iguales en que se descompone [0,1]
% t_f: instante de tiempo hasta el que calculamos la solucion
% tsteps: numero de partes iguales en que se descompone [0,t_f]

%Constantes
k=400;
rho=9e6;
c=0.38;
r=120000;
u_ext=20;
L=0.03;
B=300;
F=3e5;

%Mallado
dt=t_f/tsteps;
tiempos=0:dt:t_f;
h=L/M;
x=0:h:L;

%Datos iniciales
u=(20+120*cosh(17*x)-57*sinh(17*x))';

%El operador D
%Usamos el comando diag(vector,k) para crear una matriz tridiagonal
Tridiag=diag(-2*ones(M+1,1))+diag(ones(M,1),1)+diag(ones(M,1),-1);
Tridiag(1,2)=2;
Tridiag(M+1,M)=2;
D=(1/h^2)*Tridiag;

%Datos de salida: en vez de guardar todos los datos intermedios,
guardamos el
%vector u solo Nframes veces.
Nframes=5;
marca=floor(tsteps/(Nframes-1));
u_save=zeros(M+1,Nframes);
t_save=zeros(1,Nframes);
%guardamos la posicion de partida
u_save(:,1)=u;
```

```
t_save(1)=0;

%Bucle principal
I=eye(M+1);
A = I + (dt*k/(rho*c))*D - (dt*r/(rho*c))*I;
b = (dt*r/(rho*c))*u_ext;
for n=1:tsteps
        u=A*u+b;
        %Guardamos los valores de u para algunos tiempos
        if mod(n,marca)==0
            indice=1+n/marca;
            u_save(:,indice)=u;
            t_save(indice)=tiempos(n);
        end
end
```

Lo llamamos y dibujamos la temperatura en la barra al cabo de diez segundos:

```
[u,x,t]=disipador(30, 10, 5000)
plot(x,u)
```



Observamos en la distribución final de temperatura (en color violeta) que, debido al intercambio de temperatura con el medio, la temperatura ha descendido hasta menos de 100 grados y, gracias a la difusión de calor, la temperatura es casi uniforme en toda la barra. Si dejamos que las barras del disipador se enfríen durante más tiempo, la temperatura se acercará más a la temperatura ambiente.

2) Partiendo de la situación de reposo en la que la temperatura del disipador es la temperatura ambiente $u(x,0)=u_{Ext}$ *, se conecta el procesador que comienza a operar a pleno rendimiento durante un minuto. ¿Cuál es la temperatura en el punto más caliente de la barra al cabo de ese tiempo?*

Ahora tenemos que modificar el código para introducir el efecto del calor que genera el microprocesador. Repasamos la derivación de la aproximación a u_{xx} en el extremo izquierdo del intervalo:

$$u_{xx}(x_1) \approx \frac{1}{h^2}(u(x_1-h)-2\mathrm{u}(x_1)+u(x_1+h))$$

Y utilizando la aproximación a la condición de frontera:

$$-F/B = \frac{\partial u}{\partial x}(x_1) \approx \frac{1}{2\mathrm{h}}(u(x_1+h)-u(x_1-h))$$

$$u(x_1-h) \approx u(x_1+h)+2\mathrm{h}\,\frac{F}{B}$$

obtenemos:

$$u_{xx}(x_1) \approx \frac{1}{h^2}(-2\mathrm{u}(x_1)+2\mathrm{u}(x_1+h))+\frac{2}{h}\frac{F}{B}$$

Por tanto definimos un vector:

$$v = \frac{1}{2\mathrm{h}}\begin{pmatrix} F/B \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

y tenemos

$$\rho c \frac{d\hat{u}}{dt} = k D\hat{u} + k v - r(\hat{u} - \overrightarrow{u_{Ext}})$$

que nos conduce al esquema numérico:

$$\hat{u}^{n+1} = \hat{u}^n + \Delta t \left(\frac{d\hat{u}}{dt}\right)^n = \hat{u}^n + \frac{\Delta t}{\rho c}\left(k(D\hat{u}^n + v) - r(\hat{u} - \overrightarrow{u_{Ext}})\right)$$

$$\hat{u}^{n+1} = \left(I + \frac{\Delta t\ k}{\rho c}D - \frac{\Delta t\ r}{\rho c}I\right)\hat{u}^n + \frac{\Delta t\ r}{\rho c}\overrightarrow{u_{Ext}} + \frac{\Delta t\ k}{\rho c}v$$

El siguiente código responde a ese esquema numérico:

```
function [u_save,x,t_save]=disipador2(M, t_f, tsteps)
%function [u_save,x,t_save]=disipador2(M, t_f, tsteps)
%
%simula la evolucion de la temperatura en las barras de un disipador
%
%Datos de entrada:
% k: coeficiente k de la ecuacion
% M: numero de partes iguales en que se descompone [0,1]
% t_f: instante de tiempo hasta el que calculamos la solucion
% tsteps: numero de partes iguales en que se descompone [0,t_f]

%%Constantes
k=400;
rho=9e6;
```

```
c=0.38;
r=120000;
u_ext=20;
L=0.03;
B=300;
F=3e5;

%Mallado
dt=t_f/tsteps;
tiempos=0:dt:t_f;
h=L/M;
x=0:h:L;
%Datos iniciales
u=zeros(M+1,1);

%El operador D
%Usamos el comando diag(vector,k) para crear una matriz tridiagonal
Tridiag=diag(-2*ones(M+1,1))+diag(ones(M,1),1)+diag(ones(M,1),-1);
Tridiag(1,2)=2;
Tridiag(M+1,M)=2;
D=(1/h^2)*Tridiag;
%vector con datos de frontera
v=zeros(M+1,1);
v(1)=(2/h)*(F/B);

%Datos de salida: en vez de guardar todos los datos intermedios,
guardamos el
%vector u solo Nframes veces.
Nframes=5;
marca=floor(tsteps/(Nframes-1));
u_save=zeros(M+1,Nframes);
t_save=zeros(1,Nframes);
%guardamos la posicion de partida
u_save(:,1)=u;
t_save(1)=0;

%Bucle principal
I=eye(M+1);
A = I + (dt*k/(rho*c))*D - (dt*r/(rho*c))*I;
b = (dt*r/(rho*c))*u_ext+(dt*k/(rho*c))*v;
for n=1:tsteps
        u=A*u+b;
        %Guardamos los valores de u para algunos tiempos
        if mod(n,marca)==0
            indice=1+n/marca;
            u_save(:,indice)=u;
            t_save(indice)=tiempos(n);
        end
end
```

Llamamos al código y dibujamos la distribución de temperatura al cabo de 60 segundos. La temperatura en el punto más caliente, que es el extremo izquierdo, y en el tiempo final viene dada por **`u_save(1,end)`**

```
[u_save,x,t_save]=disipador4(20,60,10000);
printf('La temperatura en el punto mas caliente es %f',u_save(1,end))
```

```
figure;
plot(x,u_save);
xlabel('posicion')
ylabel('temperatura')
```

La temperatura en el punto mas caliente es 124.960694