Ecuaciones en Derivadas Parciales y Análisis Numérico

Prácticas

Capítulo 6. Problemas en varias dimensiones espaciales.

6.1 Solución por elementos finitos de la ecuación del calor en un recinto circular

Vamos a resolver la ecuación del calor usando el método de elementos finitos, dejando que matlab haga todos los cálculos.

El comando básico que resuelve ecuaciones del calor es:

```
u=parabolic(u0,tlist,b,p,e,t,c,a,f,d);
```

Este comando produce una aproximación mediante el método de elementos finitos de la ecuación en derivadas parciales tipo parabólico:

$$d\,u_t - \nabla(c\nabla u) + a\,u = f, \quad \Omega \subset \mathbb{R}^2, \quad t > 0,$$

los argumentos de entrada son:

`u0`	Valor inicial
`tlist`	Lista de tiempos en los que guardamos la solución (independiente de la precisión de la solución)
`b`	Texto que describe las condiciones de frontera
`p,e,t`	Describen el mallado que usaremos para aproximar la solución (lo explicaremos más adelante)
`c,a,f,d`	Coeficientes de la ecuación (serán matrices o números dependiendo de si los coeficientes son constantes)

Consideramos la ecuación:

$$u_t = k\,\Delta u = k(u_{xx} + u_{yy}) \tag{1}$$

en un disco circular D de centro 0 y radio 1, con $k = 0.1$ con datos de frontera:

$$u(x, y, t) = 0 \qquad (x, y) \in \partial D,\, t > 0 \tag{2}$$

y con distribución inicial de temperatura:

$$u(x, y, t) = f(x, y) = \begin{cases} 1 + \cos(2\pi d) & \text{si } d = |(x, y) - (\frac{1}{2}, 0)| < \frac{1}{2} \\ 0 & \text{en otro caso} \end{cases} \tag{3}$$

Vamos a definir los parámetros para ejecutar el comando `parabolic` para nuestra ecuación 1:

```
g='circleg';  % Define el circulo D de centro 0 y radio 1
b='circleb1'; % Condiciones Dirichlet homogéneas
c=.1;
a=0;
f=0;
d=1;
```

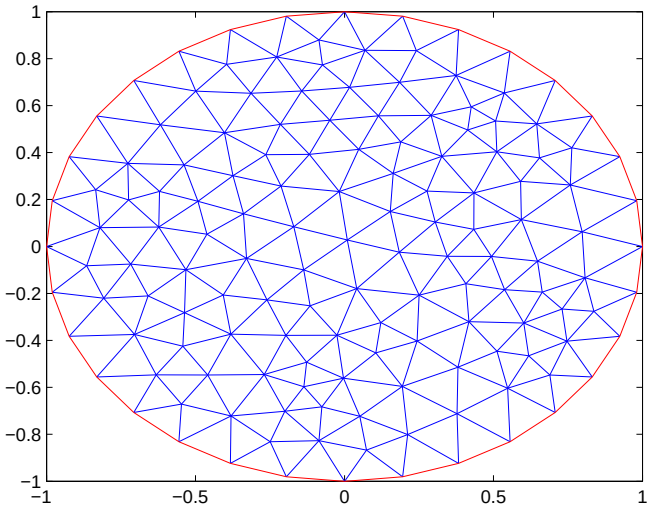
Mallado

El primer paso para usar un método de elementos finitos es descomponer el dominio en partes lo suficientemente pequeñas. Habitualmente, el dominio se descompone en triángulos.

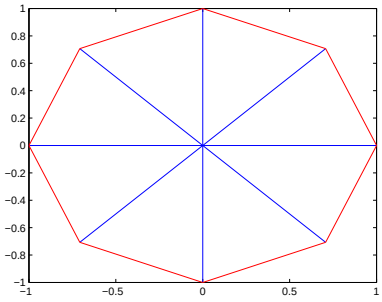
```
% Crea un mallado del cuadrado y lo guarda en las variables p, e y t
[p,e,t]=initmesh(g);
```

Veamos el mallado que ha creado:

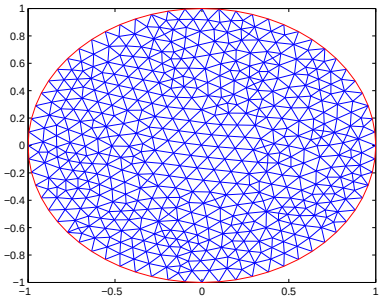
```
pdeplot(p,e,t,'mesh','on')
```



Las variables `p`, `e` y `t` almacenan información sobre los vértices, los lados y los triángulos del mallado, respectivamente. Podemos afinar el tamaño de los triángulos especificando la longitud máxima de los lados de los triángulos.



```
[p,e,t]=initmesh(g,'hmax',1);
```



```
[p,e,t]=initmesh(g,'hmax',.1);
```

Cuanto menores los triángulos, mejor la precisión de la solución, pero aumenta el número de triángulos y, por tanto, el coste computacional. La solución calculada nos devolverá los valores aproximados de la función `u` en los vértices de los triángulos, y para los tiempos que le pedimos en el vector `tlist`:

```
t_f=1;
Nframes=10; % Fija el número de intervalos de tiempo
tlist=linspace(0,t_f,Nframes); % Fija la lista de tiempos: son Nframes tiempos
```

```
% desde 0 hasta t_f
```

Definimos el dato inicial. `p` contiene los vértices de los triángulos del mallado siendo: `p(1,:)` las coordenadas x, y `p(2,:)` las coordenadas y de los vértices de los triángulos. Observa el uso que hacemos del comando `find` para encontrar los índices que corresponden a puntos cuyas distancias al punto $(\frac{1}{2},0)$ sea menor que 0.5.

```
x=p(1,:)';
y=p(2,:)';

%Inicializamos u0 para que tome el valor 0
u0=zeros(size(x));

%Calculamos el vector con las distancias al punto (0.5,0)
distancias=sqrt((x-0.5).^2+(y).^2);

%Buscamos los indices del vector de distancias donde la distancia
% es menor que 0.5
I=find(distancias<0.5);

%Para los indices anteriores, actualizamos el valor de u0 a 1-cos(d)
fun=(1+cos(2*pi*distancias));
u0(I)=fun(I);
```

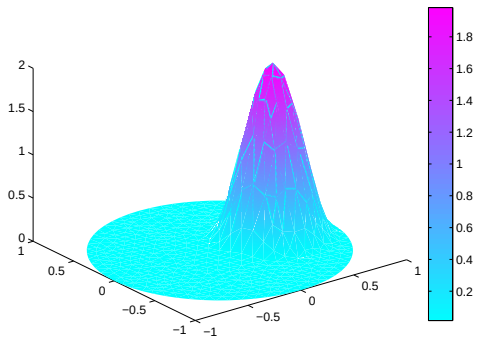
Ya podemos ejecutar el comando que nos da la evolución por elementos finitos:

```
uu=parabolic(u0,tlist,b,p,e,t,c,a,f,d);
```

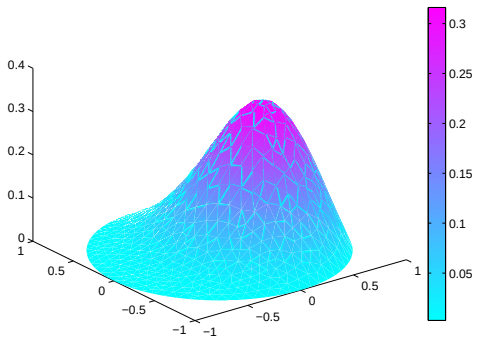
Finalmente, dibujamos la solución para tiempos $t=0$, $t=\frac{t_f}{2}$, $t=t_f$, por ejemplo.

```
figure;
pdeplot(p,e,t,'xydata',uu(:,1),'zdata',uu(:,1),'mesh','off');
figure;
mitad=floor(Nframes/2);
pdeplot(p,e,t,'xydata',uu(:,mitad),'zdata',uu(:,mitad),'mesh','off');
figure;
pdeplot(p,e,t,'xydata',uu(:,Nframes),'zdata',uu(:,Nframes),'mesh','off');
```

y obtenemos gráficas como éstas:



t=0



t=0.5

6.2 Solución exacta de la ecuación

Vamos a intentar resolver de forma exacta la misma ecuación 1 con la condición de frontera 2 y la distribución inicial de temperatura 3.

En las clases de teoría habéis visto que en un dominio circular la solución exacta de la ecuación del calor en coordenadas polares viene dada por:

$$\begin{aligned} u(r,\theta,t) = & \sum_{m=0}^{\infty}\sum_{n=1}^{\infty} A_{mn} e^{\lambda_{mn} k t} \cos(m\theta) J_m(\sqrt{-\lambda_{mn}}\, r) \\ & + \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} B_{mn} e^{\lambda_{mn} k t} \sin(m\theta) J_m(\sqrt{-\lambda_{mn}}\, r) \end{aligned} \tag{4}$$

El autovalor λ_{mn} se puede escribir en la forma $\lambda_{mn} = -(\xi_{mn})^2$, donde ξ_{mn} es el cero n-ésimo de la función de Bessel $J_m(z)$ (la condición necesaria para que se verifique la condición de frontera $r = 1 \Rightarrow u(r,\, \theta,\, t) = 0$ es $J_m(\sqrt{-\lambda_{mn}}) = J_m(\xi_{mn}) = 0$). Los coeficientes A_{mn} y B_{mn} se pueden calcular del siguiente modo:

$$A_{mn} = \frac{\int\int u_0(r,\theta)\cos(m\theta) J_m(\xi_{mn} r) r\, dr\, d\theta}{\int\int (\cos(m\theta) J_m(\xi_{mn} r))^2 r\, dr\, d\theta} \tag{5}$$

$$B_{mn} = \frac{\int\int u_0(r,\theta)\sin(m\theta) J_m(\xi_{mn} r) r\, dr\, d\theta}{\int\int (\sin(m\theta) J_m(\xi_{mn} r))^2 r\, dr\, d\theta} \tag{6}$$

Comenzamos por definir un fichero `efe.m` que calcula el dato inicial en un conjunto de puntos en coordenadas polares:

```
function Z=efe(r,theta)
%function Z=efe(r,theta)
%Acepta un vector de N valores de r como primer argumento
%y un vector de M valores de theta como segundo argumento
%Devuelve una matriz NxM con los valores de efe
%en los puntos (r(n),theta(m))

[R,THETA]=meshgrid(r,theta);
X=R.*cos(THETA);
Y=R.*sin(THETA);
D=sqrt((X-0.5).^2+Y.^2);
I=find(D<0.5);
Fun=1+cos(2*pi*D);
Z=zeros(size(R));
Z(I)=Fun(I);
```

Toma nota de las siguientes observaciones:

- `r` es un vector fila de longitud N que contiene valores de la coordenada r.
- `theta` es también un vector fila, de longitud M, que contiene valores de la coordenada θ.
- Para representar una aproximación a una función de las dos variables r y θ, necesitamos una matriz `Z` de dimensiones NxM, donde `Z(n,m)` aproxima el valor de la función en el punto de coordenadas `r(n)` y `theta(m)`.

- `R` y `THETA` son matrices NxM, que representan las funciones $(r,\theta) \to r$ y $(r,\theta) \to \theta$ respectivamente. De este modo, `R(n,m)=r(n)` y `THETA(n,m)=theta(m)`.
- `X`,`Y`,`D`,`Fun` y `Z` representan todas funciones de las dos variables r y θ.

Para calcular todas las integrales 5 y 6 de forma aproximada, podéis usar la función `coeficientes.m`, que podéis descargar de la página web de las prácticas de esta asignatura. No vamos a entrar en comentar ese código, que usa un método numérico para aproximar las integrales. Descarga también el archivo `cerosbessel.m`, que simplemente crea una matriz con unos cuantos ceros de las funciones de Bessel. Desgraciadamente, matlab no dispone de ninguna función que calcule estos valores. Ejecutando el script desde la línea de comandos cargamos en memoria estos valores:

```
cerosbessel;
```

preparamos el mallado inicial, en las coordenadas r y θ:

```
Nr=60;      %Numero de divisiones del intervalo [0,1]
Ntheta=60;  %Numero de divisiones del intervalo [0,2*pi]
r=linspace(0,1,Nr);            %Descomposicion de [0,1] en Nr-1 partes iguales
theta=linspace(0,2*pi,Ntheta); %Descomposicion de [0,2*pi] en
                               %Ntheta-1 partes iguales
[R,THETA]=meshgrid(r,theta);
```

En este punto ya podemos dibujar la distribución inicial de temperatura

```
X=R.*cos(THETA);
Y=R.*sin(THETA);
Z=efe(r,theta);
mesh(X,Y,Z);
```

Calculamos los coeficientes de la función definida en `efe.m` usando el script `coeficientes.m` (recuerda que tiene que estar en el directorio de trabajo, al igual que `efe.m`)

```
[A0,A,B]=coeficientes();
```

Finalmente, podemos aplicar la fórmula 4 para calcular la distribución de temperatura al cabo de un tiempo t_f que tomamos igual a 2. Inicializamos un vector `Zt` a ceros y le sumamos todos los términos correspondientes. Observa que usamos las funciones `besselj`, donde `bessel(m,r)` calcula $J_m(r)$ y, si `r` es un vector, devuelve otro vector resultado de evaluar J_m en cada punto del vector `r`.

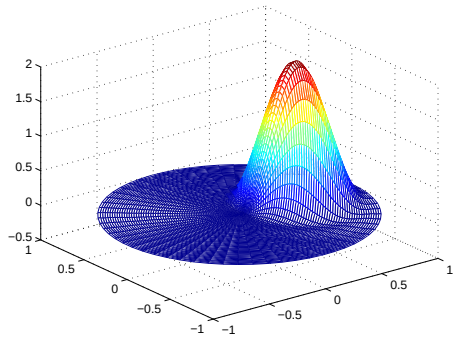
```
t_f=2;
k=0.1;
Zt=zeros(size(Z));

%Sumamos los terminos de la serie con m=0 y n entre 1 y 10
Cosenos=cos(0*THETA); %en realidad se puede poner Cosenos=ones(size(Z))
                      %porque cos(0)=1
for n=1:10
    psi0n=besselC0n(n);
    landa0n=-psi0n^2;
    expo=exp(landa0n*k*t_f);
    Bessel = besselj(0,psi0n*R);
    Zt=Zt+A0(n)*expo*Cosenos.*Bessel;
end
```

```
%Sumamos los terminos de la serie con m y n entre 1 y 10
for m=1:10
    Cosenos=cos(m*THETA);
    Senos=sin(m*THETA);
    for n=1:10
        psimn=besselCmn(m,n);
        landamn=-psimn^2;
        expo=exp(landamn*k*t_f);
        Bessel= besselj(m,psimn*R);
        Zt=Zt+A(m,n)*expo*Cosenos.*Bessel+ B(m,n)*expo*Senos.*Bessel;
    end
end
```

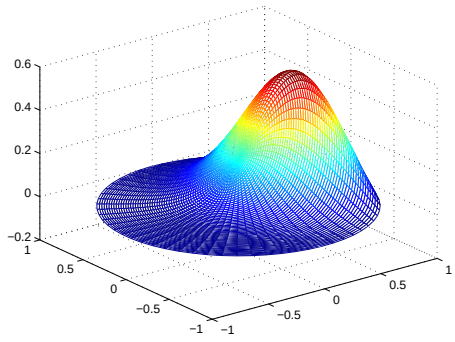
Ahora podemos dibujar la distribución de temperatura al cabo de un cierto tiempo de modo similar al que usábamos antes:

```
mesh(X,Y,Zt);
```

Por ejemplo, poniendo `t_f=0`, comprobamos qué tal es nuestra aproximación a la distribución inicial, y poniendo `t_f=0.5`, podemos comparar el resultado con el obtenido en el apartado anterior:



$t=0$



$t=0.5$

6.3 Ecuación de ondas en un cuadrado por diferencias finitas

Ahora vamos a resolver la ecuación de ondas en un recinto cuadrado utilizando un método de diferencias finitas. Consideramos la ecuación de ondas:

$$u_{tt} = c^2 \Delta u = c^2 (\, u_{xx} + u_{yy}) \tag{7}$$

en una dimensión espacial, en el cuadrado $[0,1] \times [0,1]$, para tiempos entre 0 y t_f, con condiciones de contorno:

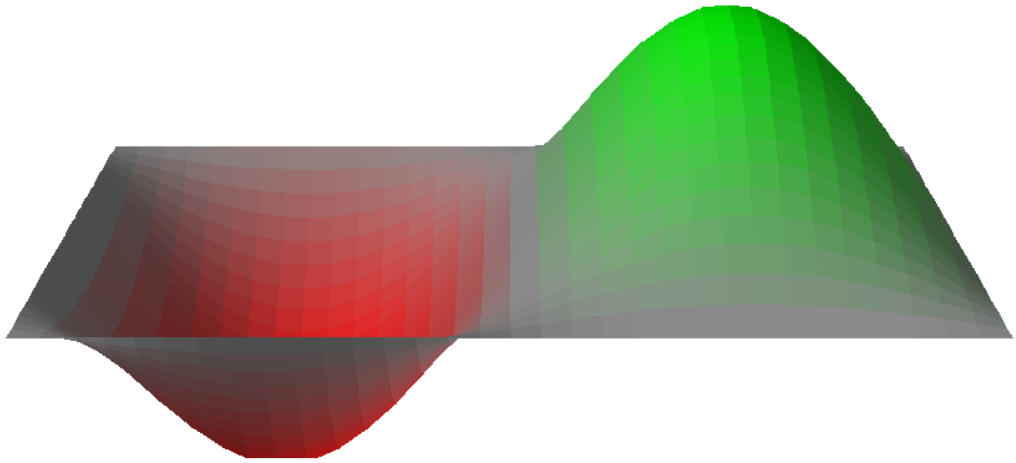
$$u(x,y,t) = 0 \quad \text{si}\,(x,y) \in \partial([0,1] \times [0,1]\ ) \tag{8}$$

y con condiciónes iniciales:

$$u(x,y,0) = \quad f(x,y) \tag{9}$$

$$u_t(x,y,0) = \quad g(x,y) \tag{10}$$

Podemos pensar que estas ecuaciones modelizan el comportamiento de una membrana cuadrada con los bordes fijos.



Como hemos hecho otras veces, interpretamos la ecuación como un sistema de ecuaciones diferenciales ordinarias (EDO) en las que un valor inicial u (ahora una función de dos variables) que contiene los desplazamientos verticales de las distintas posiciones de la membrana evoluciona según una EDO de segundo orden.

De nuevo, es necesario utilizar una versión *discreta* del valor inicial para poder representarlo en el ordenador mediante un vector. Descomponemos el cuadrado $[0,\,1] \times [0,\,1]$ en M^2 partes iguales, obteniendo un mallado uniforme $(x_m,\, y_l)$ para $m = 1, ..., M+1$ y $l = 1, ..., M+1$, con $x_{m+1} = x_1 + m/M$ e $y_{l+1} = y_1 + l/M$, y consideramos los valores de u sólo en esos puntos.

Dependiendo de lo que vayamos a hacer, puede ser mejor almacenar una función definida en todo el cuadrado como una matriz de dimensiones $(M+1) \times (M+1)$ o como un vector de longitud $(M+1)^2$, donde las columnas de la matriz se disponen una detrás de otra. Llamaremos a este vector la versión *aplanada* de la matriz.

$$\begin{pmatrix} u_{11} & u_{12} & \dots & u_{1(M+1)} \\ u_{21} & u_{22} & \dots & u_{2(M+1)} \\ \vdots & & & \vdots \\ u_{(M+1)1} & u_{(M+1)2} & \dots & u_{(M+1)(M+1)} \end{pmatrix} \qquad \begin{pmatrix} u_{11} \\ \vdots \\ u_{(M+1)1} \\ u_{12} \\ \vdots \\ u_{(M+1)2} \\ \vdots \\ u_{(M+1)(M+1)} \end{pmatrix}$$

Matriz Matriz aplanada

Para cambiar de una forma a otra, utilizaremos el comando `reshape` para cambiar la forma de una matriz sin alterar sus elementos.

```
A=[1,2;3,4]
reshape(A,4,1)  %Devuelve el vector columna [1;2;3;4]
```

Como ya conocemos los valores de u en la frontera del cuadrado para todos los tiempos, a la hora de hacer los cálculos trabajaremos con la versión reducida de la matriz de arriba, donde los índices van desde 2 hasta $M-1$. Para mantener la distinción entre las distintas variables que representan funciones definidas en el cuadrado, usaremos los nombres siguientes:

`U, F, G`	matrices bidimensionales que incluyen los datos en la frontera
`U_red, F_red, G_red`	matrices bidimensionales sin los datos de la frontera
`u, f, g`	matrices aplanadas (es decir, vectores) que no incluyen los datos de la frontera

Dicho ésto, nos disponemos a buscar una versión discreta de la ecuación en derivadas parciales 7. Para comenzar, tomamos funciones del tiempo $\hat{u}_{m,l}(t)$, que representan la altura de la membrana sobre el punto $(x_m,\ y_l)$ en el instante de tiempo t. Podemos disponer todos los $\hat{u}_{m,l}(t)$ en un vector $\hat{u}(t)$ (observa que no incluimos los valores en la frontera):

$$\hat{u}(t) = \begin{pmatrix} u_{22}(t) \\ \vdots \\ u_{M2}(t) \\ u_{23}(t) \\ \vdots \\ u_{M3}(t) \\ \vdots \\ u_{\mathrm{M\,M}}(t) \end{pmatrix}$$

Podemos encontrar una ecuación diferencial ordinaria para este vector $\hat{u}(t)$ si sustituimos las derivadas parciales $(u_{xx} + u_{yy})$ por un operador que actúe sobre el vector $\hat{u}$. Utilizamos las diferencias finitas centradas habituales para aproximar derivadas de segundo orden:

$$u_{xx}(x_m, y_l) \approx \tfrac{1}{h^2}(\hat{u}_{m-1,l} - 2\hat{u}_{m,l} + \hat{u}_{m+1,l})$$
$$u_{yy}(x_m, y_l) \approx \tfrac{1}{h^2}(\hat{u}_{m,l-1} - 2\hat{u}_{m,l} + \hat{u}_{m,l+1})$$

Obtenemos la siguiente aproximación para el laplaciano:

$$\Delta u(x_m, y_l) \approx \tfrac{1}{h^2}(-4\hat{u}_{m,l} + \hat{u}_{m-1,l} + \hat{u}_{m+1,l} + \hat{u}_{m,l-1} + \hat{u}_{m,l+1})$$

Y si ponemos todas las aproximaciones juntas, para el caso $M = 5$:

$$\begin{pmatrix} \Delta u_{22} \\ \Delta u_{23} \\ \Delta u_{24} \\ \Delta u_{32} \\ \Delta u_{33} \\ \Delta u_{34} \\ \Delta u_{42} \\ \Delta u_{43} \\ \Delta u_{44} \end{pmatrix} \approx \frac{1}{h^2} \begin{pmatrix} -4\hat{u}_{22} + \hat{u}_{23} + \hat{u}_{32} \\ -4\hat{u}_{23} + \hat{u}_{22} + \hat{u}_{33} + \hat{u}_{24} \\ -4\hat{u}_{24} + \hat{u}_{34} + \hat{u}_{23} \\ -4\hat{u}_{32} + \hat{u}_{22} + \hat{u}_{42} + \hat{u}_{33} \\ -4\hat{u}_{33} + \hat{u}_{23} + \hat{u}_{43} + \hat{u}_{22} + \hat{u}_{24} \\ -4\hat{u}_{34} + \hat{u}_{24} + \hat{u}_{44} + \hat{u}_{33} \\ -4\hat{u}_{42} + \hat{u}_{32} + \hat{u}_{43} \\ -4\hat{u}_{43} + \hat{u}_{33} + \hat{u}_{42} + \hat{u}_{44} \\ -4\hat{u}_{44} + \hat{u}_{34} + \hat{u}_{43} \end{pmatrix}$$

$$= \frac{1}{h^2} \begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix} \begin{pmatrix} u_{22} \\ u_{23} \\ u_{24} \\ u_{32} \\ u_{33} \\ u_{34} \\ u_{42} \\ u_{43} \\ u_{44} \end{pmatrix} = D \begin{pmatrix} u_{22} \\ u_{23} \\ u_{24} \\ u_{32} \\ u_{33} \\ u_{34} \\ u_{42} \\ u_{43} \\ u_{44} \end{pmatrix}$$

Observa cómo hemos utilizado las condiciones de frontera tipo Dirichlet. En general, vemos que podemos aproximar Δu con una matriz similar a la que utilizábamos en el caso de una única dimensión espacial. Nuestra ecuación diferencial de segundo orden para $\hat{u}(t)$ se escribe entonces:

$$\frac{d^2\hat{u}}{dt^2} = F(\hat{u}) = c^2 D\hat{u}$$

Finalmente, aplicamos la aproximación habitual para $\frac{d^2\hat{u}}{dt^2}$:

$$\frac{d^2\hat{u}}{dt^2} \approx \frac{1}{(\Delta t)^2}(\hat{u}(t_{n-1}) - 2\hat{u}(t_n) + \hat{u}(t_{n+1}))$$

Utilizando esta aproximación obtenemos el esquema numérico:

$$\hat{u}^{(n+1)} = 2\hat{u}^{(n)} - \hat{u}^{(n-1)} + (\Delta t)^2 \cdot F(\hat{u})$$

y para el primer paso temporal obtenemos la misma fórmula que en el capítulo 5, en función de los datos iniciales:

$$u^{(1)} = \hat{f} + \Delta t\ \hat{g} + \frac{1}{2}(\Delta t)^2 \cdot F(\hat{f})$$

donde escribimos $\hat{f}$ y $\hat{g}$ para representar los vectores $(f(x_m,\, y_l))$ y $(g(x_m,\, y_l))$ para todos los pares (m, l).

Finalmente, éste es nuestro código para la ecuación de ondas en el cuadrado $[0,1]\times[0,1]$, con dato inicial:

$$\begin{aligned} f(x,y) &= \min(x, y, 1-x, 1-y) \\ g(x,y) &= 0 \end{aligned}$$

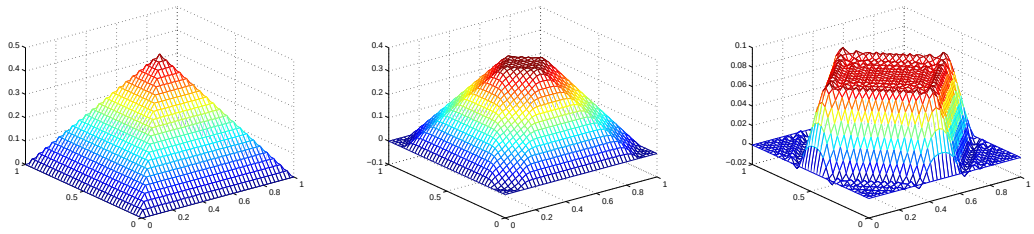
```
function [U_save,x,y,t_save]= ondas2D(c,t_f,M,tsteps);
%function [U_save,x,y,t_save]= ondas2D(c,t_f,M,tsteps);
%
%Resuelve utt - c2 (uxx +uyy) = 0 mediante un esquema explicito de segundo orden
%
%Datos de entrada:
% c: coeficiente c de la ecuacion
% M: numero de partes iguales en que se descompone cada intervalo [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;
L=1;%Cuadrado [0,1]
h=L/M;
x=0:h:L;
y=0:h:L;
[X,Y]=meshgrid(x,y);

%%Condiciones iniciales
F=min(min(X,Y),min(1-X,1-Y));%La posicion inicial
G=zeros(size(X)); %La velocidad inicial
%Condiciones iniciales, sin los bordes:
F_red=F(2:M,2:M);
G_red=G(2:M,2:M);
%Versiones aplanadas
f=reshape(F_red,(M-1)^2,1);
g=reshape(G_red,(M-1)^2,1);

%%Datos de salida: en vez de guardar todos los datos intermedios,
%guardamos el vector u solo Nframes veces.
Nframes=30; marca=floor(tsteps/(Nframes-1));
%Utilizamos la matriz U para trabajar comodamente con el primer indice
%para la coordenada 'x' y el segundo para la coordenada 'y'
U=zeros(M+1,M+1);
U_save=zeros(M+1,M+1,Nframes);
t_save=zeros(Nframes,1);
%Le ponemos las condiciones de frontera
%lado izquierdo
U_save(1,:,:)=zeros(1,M+1,Nframes);
%lado derecho
U_save(M+1,:,:)=zeros(1,M+1,Nframes);
%lado inferior
U_save(:,1,:)=zeros(M+1,1,Nframes);
%lado superior
U_save(:,M+1,:)=zeros(M+1,1,Nframes);
%guardamos la posicion de partida
U_save(:,:,1)=F;
```

```
t_save(1)=0;

%%El operador D
%Usamos el comando diag(vector,k) para crear una matriz tridiagonal
%Y el comando kron para poner los unos en los pares de indices
%que representan puntos vecinos del mallado
Vecinos = diag(-4*ones((M-1)^2,1))+...
   kron(eye(M-1),diag(ones(M-2,1),1))+...
   kron(eye(M-1),diag(ones(M-2,1),-1))+...
   kron(diag(ones(M-2,1),1),eye(M-1))+...
   kron(diag(ones(M-2,1),-1),eye(M-1));
D=(1/h^2)*Vecinos;

% Calculo del primer paso temporal
u_ant=f;
u = f + dt*g + 0.5*(dt^2)*(c^2)*D*f;
% Iteracion para el resto de pasos temporales
%Bucle principal
for n=1:tsteps
    temp=u;
    u = 2*u - u_ant + (dt^2)*(c^2)*D*u;
    u_ant=temp;
    %Guardamos los valores de u para algunos tiempos
    if mod(n,marca)==0
        indice = 1 + n/marca;
        U(2:M,2:M)=reshape(u,M-1,M-1);
        U_save(:,:,indice)=U;
        t_save(indice)=tiempos(n);
    end
end
```

Llamamos al código anterior (observa que somos mucho más modestos con el número de divisiones del intervalo):

```
[U,x,y,t]=ondas2D(1,.3,30,200);
[X,Y]=meshgrid(x,y);
figure;
mesh(X,Y,U(:,:,end))
```

obtenemos gráficas como éstas:



$t = 0$ $t = 0.1$ $t = 0.3$

6.4 Applets de la ecuación de ondas

En las páginas web:

http://www.falstad.com/membrane/

http://www.falstad.com/circosc/

puedes ver dos applets que muestran el comportamiento de una membrana cuadrada y circular respectivamente, que obedecen a la ecuación de ondas.