

# Programas

Excusa universal: Estos programas se realizaron rápidamente como una ayuda para algunos cálculos y para representar algunas gráficas. No se ha puesto cuidado en que sean “estructurados” ni especialmente comentados. Su finalidad era el uso personal. Desafortunadamente, por razones tipográficas ha sido necesario suprimir los indentados que preceden a la mayoría de las líneas con lo cual no se reconocen a golpe de vista los saltos en los programas FORTRAN. Este lenguaje de programación prácticamente no se emplea fuera de tareas de análisis numérico de alto rendimiento o que requieren altas garantías. En cualquier caso, son tan simples que es muy fácil traducirlos a cualquier lenguaje moderno.

---

**Sección:** 2.1

**Tipo:** MATLAB

**Descripción:** Aproxima la solución de la ecuación del calor en  $[0, 1]$  con extremos a temperatura cero y dato inicial igual a una función triángulo (Problema 2.1.10).

---

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECUACIÓN DEL CALOR EN [0,1]
% CON U(0,T)=U(1,T)=0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% toma 40*2 términos de la serie de Fourier
n=[0:1:40];
x=[0:.01:1];
%"vector Fourier"
f=sin(pi*x'*(2*n+1));
%vector de coeficientes
coef=4/pi^2*cos(pi*n)./(2*n+1).^2;
%Dibujo para t=0.001
t=0.001
expo=exp(-(2*n+1).^2*pi^2*t);
u=f*(coef.*expo)';
plot(x,u);
```

```

hold on
%Dibujo para t=0.01
t=0.01
expo=exp(-(2*n+1).^2*pi^2*t);
u=f*(coef.*expo)';
plot(x,u);
hold on
%Dibujo para t=0.1
t=0.1
expo=exp(-(2*n+1).^2*pi^2*t);
u=f*(coef.*expo)';
plot(x,u);
axis([0 1 0 0.5])

```

---

**Sección:** 2.1

**Tipo:** MATLAB

**Descripción:** Aproxima  $\exp(\cos(2\pi*x))$  por su serie de Fourier hasta  $N = 3$ .

---

```

%%%%%%%%%%%%%%
%% APROXIMACIÓN DE FOURIER
%%%%%%%%%%%%%%
x=[0:.009:1];
subplot(1,3,1)
f=exp(cos(2*pi*x));
plot(x,f,'--')
hold on
f1=1.26066+0.565159*2*cos(2*pi*x);
plot(x,f1);
hold off
subplot(1,3,2)

```

```

f=exp(cos(2*pi*x));
plot(x,f,'--')
hold on
f2=1.26066+0.565159*2*cos(2*pi*x)+0.135749*2*cos(4*pi*x);
plot(x,f2);
hold off
subplot(1,3,3)
f=exp(cos(2*pi*x));
plot(x,f,'--')
hold on
f3=1.26066+0.565159*2*cos(2*pi*x)+ ...
0.135749*2*cos(4*pi*x)+0.022169*2*cos(6*pi*x);
plot(x,f3);
hold off
print -eps fouri.eps

```

---

## Sección: 2.2

**Tipo:** MATLAB

**Descripción:** A partir de la foto `dpto.jpg` crea las fotos filtradas `jp*.eps` en las que se cambian eliminan los coeficientes de Fourier correspondientes a ceros en la matriz `mask`. Con él se han creado los ejemplos de la sección.

---

```

I = imread('dpto.jpg');
I = im2double(I);
T = dctmtx(8);
B = blkproc(I,[8 8], 'P1*x*P2',T,T');
% Señala los bloques y les da el color del promedio
mask = [1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

```

```

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);
imshow(I2)
print -deps jp1.eps
% Usa cinco coeficientes de Fourier
mask = [1 1 1 0 0 0 0 0
1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);
imshow(I2)
print -deps jp2.eps
% Usa 16 coeficientes de Fourier
mask = [1 1 1 1 0 0 0 0
1 1 1 1 0 0 0 0
1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

```

```

0 0 0 0 0 0 0 0];
B2 = blkproc(B,[8 8], 'P1.*x',mask);
I2 = blkproc(B2,[8 8], 'P1*x*P2',T',T);
imshow(I2)
print -deps jp3.eps
% Detecta verticales
mask = [1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];
B2 = blkproc(B,[8 8], 'P1.*x',mask);
I2 = blkproc(B2,[8 8], 'P1*x*P2',T',T);
imshow(I2)
print -deps jp4.eps

```

---

**Sección:** 2.2

**Tipo:** MATLAB

**Descripción:** Ésta es una versión simplificada del programa anterior

---

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% CARGA UNA IMAGEN EN COLOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xg=imread('lenna.jpg');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ESCOGE SÓLO EL BLOQUE mxm DE CADA BLOQUE DE
%% COEFICIENTES DE FOURIER

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
m=2;
```

```
x=masc(xg,m);
```

```
-----
```

```
function q=masc(I,n)
```

```
I = im2double(I);
```

```
T = dctmtx(8);
```

```
B = blkproc(I,[8 8], 'P1*x*P2',T,T');
```

```
% Señala los bloques y les da el color del promedio
```

```
mask=zeros(8);
```

```
mask(1:n,1:n)=1;
```

```
B2 = blkproc(B,[8 8], 'P1.*x',mask);
```

```
I2 = blkproc(B2,[8 8], 'P1*x*P2',T,T);
```

```
imshow(I2)
```

```
q=100*n*n/64
```

---

## Sección: 2.2

**Tipo:** MATLAB

**Descripción:** Resuelve el problema 2.2.6 con diferentes cuantizaciones. Evidentemente estas cuentas se pueden hacer con muchas menos líneas de programa.

---

```
%% PARÁMETROS
```

```
I=ones(64, 64);
```

```
t=pi/6;
```

```
v0=[1 cos(t*1) cos(2*t*1)];
```

```
v1=[1 cos(t*3) cos(2*t*3)];
```

```
v2=[1 cos(t*5) cos(2*t*5)];
```

```
%% MATRIZ DE PRODUCTOS ESCALARES
```

```
L=[-512 -221.7025 128
```

```
-221.7025 -96 55.4256
```

```

128 55.4256 -32];
delta=[1/3 2/3 2/3]’*[1/3 2/3 2/3];
%% MATRIZ DE LOS lambda kl
L=L.*delta;
%% CUANTIZACIÓN TRIVIAL
Cuant=[1 1 1; 1 1 1; 1 1 1]
round(L./Cuant)
Lc=Cuant.*round(L./Cuant)
%% RECONSTRUCCIÓN
fr=round([v0*Lc*v0’ v0*Lc*v1’ v0*Lc*v2’;...
v1*Lc*v0’ v1*Lc*v1’ v1*Lc*v2’; v2*Lc*v0’ v2*Lc*v1’ v2*Lc*v2’])
Imag=[fr(3,1)*I fr(3,2)*I fr(3,3)*I;...
fr(2,1)*I fr(2,2)*I fr(2,3)*I;...
fr(1,1)*I fr(1,2)*I fr(1,3)*I];
figure(1)
imshow(Imag, [-128 127])
%% CUANTIZACIÓN DEL PROBLEMA
Cuant=[20 40 60; 40 60 80; 60 80 100]
round(L./Cuant)
Lc=Cuant.*round(L./Cuant)
%% RECONSTRUCCIÓN
fr=round([v0*Lc*v0’ v0*Lc*v1’ v0*Lc*v2’;...
v1*Lc*v0’ v1*Lc*v1’ v1*Lc*v2’; v2*Lc*v0’ v2*Lc*v1’ v2*Lc*v2’])
Imag=[fr(3,1)*I fr(3,2)*I fr(3,3)*I;...
fr(2,1)*I fr(2,2)*I fr(2,3)*I;...
fr(1,1)*I fr(1,2)*I fr(1,3)*I];
figure(2)
imshow(Imag, [-128 127])
%%CUANTIZACIÓN COMO LA DEL PRIMER BLOQUE 3X3 DEL JPEG

```

```

Cuant=[16 11 10; 12 12 14; 14 13 16]
round(L./Cuant)
Lc=Cuant.*round(L./Cuant)
%%RECONSTRUCCIÓN
fr=round([v0*Lc*v0' v0*Lc*v1' v0*Lc*v2';...
v1*Lc*v0' v1*Lc*v1' v1*Lc*v2'; v2*Lc*v0' v2*Lc*v1' v2*Lc*v2'])
Imag=[fr(3,1)*I fr(3,2)*I fr(3,3)*I;...
fr(2,1)*I fr(2,2)*I fr(2,3)*I;...
fr(1,1)*I fr(1,2)*I fr(1,3)*I];
figure(3)
imshow(Imag, [-128 127])
%%CUANTIZACIÓN COMO LA DEL BLOQUE 0,3,6x0,3,6 DEL JPEG
Cuant=[16 16 51; 14 29 80; 49 87 120]
round(L./Cuant)
Lc=Cuant.*round(L./Cuant)
%%RECONSTRUCCIÓN
fr=round([v0*Lc*v0' v0*Lc*v1' v0*Lc*v2';...
v1*Lc*v0' v1*Lc*v1' v1*Lc*v2'; v2*Lc*v0' v2*Lc*v1' v2*Lc*v2'])
Imag=[fr(3,1)*I fr(3,2)*I fr(3,3)*I;...
fr(2,1)*I fr(2,2)*I fr(2,3)*I;...
fr(1,1)*I fr(1,2)*I fr(1,3)*I];
figure(4)
imshow(Imag, [-128 127])
%%CUANTIZACIÓN COMO LA MITAD DE LA DEL PROBLEMA
Cuant=[10 20 30; 20 30 40; 30 40 50]
round(L./Cuant)
Lc=Cuant.*round(L./Cuant)
%%RECONSTRUCCIÓN
fr=round([v0*Lc*v0' v0*Lc*v1' v0*Lc*v2';...

```



```

v1*Lc*v0' v1*Lc*v1' v1*Lc*v2'; v2*Lc*v0' v2*Lc*v1' v2*Lc*v2'])
Imag=[fr(3,1)*I fr(3,2)*I fr(3,3)*I;...
fr(2,1)*I fr(2,2)*I fr(2,3)*I;...
fr(1,1)*I fr(1,2)*I fr(1,3)*I];
figure(5)
imshow(Imag, [-128 127])

```

## Sección: 2.2

**Tipo:** MATLAB

**Descripción:** Filtra los coeficientes de Fourier de la foto `lenna.jpg` eliminando los que están por debajo de ciertos valores umbral. La función de filtrado `tr8x8bn` es una modificación de un programa elaborado por M. Teresa Carrillo con fines académicos.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% CARGA UNA IMAGEN EN BLANCO Y NEGRO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xg=imread('lenna.jpg');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% HACE LA TRANSFORMADA DE COSENO EN BLOQUES 8X8
%% Y ELIMINA LOS COEFICIENTES DE FOURIER MENORES QUE th
%% EL PORCENTAJE DE CEROS ES p0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Umbrales=[0 40 80 120 1000]
% LA IMAGEN ORIGINAL TIENE EL 29.7% DE CEROS
imshow(xg);
th=40;
[p1,nada]=tr8x8bn(xg,gray(256),th);
p1
th=80;
[p2,nada]=tr8x8bn(xg,gray(256),th);
p2

```

```

th=120;
[p3,nada]=tr8x8bn(xg,gray(256),th);
p3
th=1000;
[p4,nada]=tr8x8bn(xg,gray(256),th);
p4
-----
function [perf0,perfl2]=trcos8x8bn(X,map,t)
% Aplica la transformación del coseno (dct2) a la imagen X,
% con paleta map,
% descompuesta en bloques 8x8.
% X debe ser una matriz ( $2^J \times 2^J$ ).
% map una paleta de grises. (vg. X, map obtenidas con load('woman')).
% Pasa un threshold t , (vg. 10).
% Representa:
% la imagen original,
% la imagen reconstruida por bloques 8x8 (idct2) despues
% de haberla pasado el "threshold".
% Calcula:
% perf0: tanto por ciento de ceros en la imagen transformada despues
% de pasar el "threshold".
% perfl2: cociente de las normas en l2 de la imagen transformada
% y de la imagen original.
n=size(X);
n=n(1)-7;
for i=1:8:n
for j=1:8:n
xcl=dct2(X(i:i+7,j:j+7));
xc(i:i+7,j:j+7)=xcl(1:8,1:8);

```



```

Umbrales=[0 40 80 120]
% LA IMAGEN ORIGINAL TIENE EL 10.3% DE CEROS
imshow(xg);
th=40;
[p1,nada]=tr8x8c(xg,th);
p1
th=80;
[p1,nada]=tr8x8c(xg,th);
p1
th=120;
[p1,nada]=tr8x8c(xg,th);
p1
-----
function [perf0,perfl2]=trcos8x8(X,t)
% Aplica la transformación del coseno a la imagen X
% descompuesta en bloques 8x8.
% X debe ser una matriz ( $2^J \times 2^J \times 3$ ).
% Pasa un threshold t ), (vg. 10).
[n1 n2 n3]=size(X);
n1=n1-7;
n2=n2-7;
for m=1:3
for i=1:8:n1
for j=1:8:n2
xcl(:,:,m)=dct2(X(i:i+7,j:j+7,m));
xc(i:i+7,j:j+7,m)=xcl(1:8,1:8,m);
end
end
end
end

```

```

xc(abs(xc)<t)=0;
for m=1:3
for i=1:8:n1
for j=1:8:n2
Yl(:, :, m)=idct2(xc(i:i+7, j:j+7, m));
Y(i:i+7, j:j+7, m)=Yl(1:8, 1:8, m);
end
end
end
figure
Y=uint8(Y);
imshow(Y)
% Compute compression score.
perf0 = 100*(length(find(xc==0))/(size(xc,1)*size(xc,2)*3));
% Compute  $L^2$  recovery score.
xc=double(xc); Xd=double(X);
perf12 = 100*(sum((xc(:).*xc(:)))/sum(Xd(:).*Xd(:)));

```

---

**Sección:** Exp. 3.1

**Tipo:** MATLAB

**Descripción:** Realiza las iteraciones indicadas de los  $\vec{x}_n$ . Se debe dar el valor inicial de  $\mathbf{x}$ . Cada vez que se ejecuta el programa se lleva a cabo una iteración.

---

```

c=[2;1;1;1;1;2;1;0;0;0;1];
f1=[1 1 1 0 0 0 0 0 0];
x=x+(c(1)-f1*x)*f1'/(f1*f1');
f2=[0 0 0 1 1 1 0 0 0];
x=x+(c(2)-f2*x)*f2'/(f2*f2');
f3=[0 0 0 0 0 0 1 1 1];
x=x+(c(3)-f3*x)*f3'/(f3*f3');

```

```

f4=[1 0 0 1 0 0 1 0 0];
x=x+(c(4)-f4*x)*f4'/(f4*f4');
f5=[0 1 0 0 1 0 0 1 0];
x=x+(c(5)-f5*x)*f5'/(f5*f5');
f6=[0 0 1 0 0 1 0 0 1];
x=x+(c(6)-f6*x)*f6'/(f6*f6');
f7=[1 0 0 0 0 0 0 0 0];
x=x+(c(7)-f7*x)*f7'/(f7*f7');
f8=[0 0 0 0 0 0 1 0 0];
x=x+(c(8)-f8*x)*f8'/(f8*f8');
f9=[0 0 0 0 0 0 0 0 1];
x=x+(c(9)-f9*x)*f9'/(f9*f9');
f10=[0 0 1 0 0 0 0 0 0];
x=x+(c(10)-f10*x)*f10'/(f10*f10')

```

---

**Sección:** Exp. 3.1

**Tipo:** MATLAB

**Descripción:** Ejemplo de reconstrucción algebraica.

---

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% RECONSTRUCCIÓN ALGEBRAICA
%% DE UNA MUESTRA 3X3 DEL TIPO:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% OXX
%% XOO
%% OXO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PARÁMETROS
I=ones(64, 64);
%% VALORES INICIALES

```

```

%% ES MEJOR OMITIRLOS Y EJECUTAR
%% VARIAS VECES EL PROGRAMA,
%% METIÉNDOLOS EN LA PRIMERA ITERACIÓN
%% n=0
%% x=zeros(9,1)
%% LLAMADA AL PROGRAMA ANTERIOR

tomo

n=n+1;

iteracion_numero=n

%% DIBUJO

Imag=[x(1)*I x(2)*I x(3)*I;...
x(4)*I x(5)*I x(6)*I;...
x(7)*I x(8)*I x(9)*I];

imshow(Imag)

```

---

### Sección: 3.2

**Tipo:** MATLAB

**Descripción:** Reconstruye una imagen (phan.gif) consistente en un círculo negro con una burbuja (un agujero) a la derecha, usando la transformada de Radon.

---

```

figure(1)

I=ones(100,100);

I(25:75, 25:75)=0;

imshow(I)

theta=0:180;

[R, xp]= radon(I,theta);

figure(2)

I = imread('phan.gif');

I = im2double(I);

imshow(I)

```





```
theta=[0:h:180];
[R,a]=radon(X,theta);
figure
imshow(iradon(R,h));
```

---

**Sección:** Exp. 3.2

**Tipo:** MATLAB

**Descripción:** Dibuja las “sombras” correspondientes a los tres tubos elegidos.

---

```
x=[-1:.01:1];
subplot(1,3,1)
y1=2*sqrt(1-x.*x);
plot(x,y1)
axis([-1 1 0 3])
subplot(1,3,2)
y2=2*sqrt(1-x.*x)-sqrt(1-min(1,4*x.*x));
plot(x,y2)
axis([-1 1 0 3])
subplot(1,3,3)
y3=2*sqrt(1-x.*x)+sqrt(1-min(1,4*x.*x));
plot(x,y3)
print -deps fsomb.eps
```

---

**Sección:** Exp. 3.2

**Tipo:** FORTRAN

**Descripción:** Crea las listas de datos correspondientes a la reconstrucción a partir de las sombras de los tubos considerados. La función  $S$  puede elegirse como  $S_1$ ,  $S_2$ ,  $S_3$ . Los dibujos tridimensionales se hacen en MATLAB, una vez cargados los datos en la matriz  $A_1$ , con  $[r,theta]=meshgrid(A_1(:,1),0:0.3:6.3)$ ;  $[z,theta]=meshgrid(A_1(:,2),0:0.3:6.3)$ ;  $mesh(r.*cos(theta),r.*sin(theta),z)$ .

---

c Algoritmo de reconstrucción para objetos 2D en el disco

```

c unidad con simetría radial a partir de la "sombra" S=S(x).
c
c
c Entrada de la precisión
print*, 'Precisión:=0.01'
h=0.01
c print*, 'x'
c read*, x
c
c BUCLE PRINCIPAL
do 100 x=0,1.3,h
c
c Regla del trapecio en [0,pi] con paso h
result=0.
do 20 theta=0,3.141592654,h
call fintegr(F,x,theta,h)
result=result+F
20 continue
result=h*result
c
c Imprime resultados
print*,x,result
100 continue
stop
end
c
c Función a integrar . Depende de x y theta
c y su valor quedará almacenado en F.
subroutine fintegr(F,x,theta,h)

```

```

N=int(1./h)
var=x*cos(theta)
F=1.23370055*S(var)
do 10 k=N,0,-1
t=float(2*k+1)
F=F-S(var+t*h)/t/t
10 continue
F=0.202642367*F/h
return
end
c
c
c Define la función sombra
c
c TUBO MACIZO R=1
function S1(u)
if (abs(u).ge.1.) then
S1=0
else
S1=2*sqrt(1.-u*u)
endif
return
end
c TUBO HUECO (CORONA .5<R<1)
function S2(u)
S2=S1(u)-.5*S1(2.*u)
return
end
c TUBO CON ALMA R<.5 DE DOBLE DENSIDAD

```

```

function S3(u)
S3=S1(u)+.5*S1(2.*u)
return
end
c FUNCION
function S(u)
S=S3(u)
return
end

```

---

**Sección:** 4.1

**Tipo:** FORTRAN

**Descripción:** Genera datos aleatorios que se pueden aproximar con el teorema central del límite.

---

```

c Teorema central del límite tirando 10 dados n veces
c
c
dimension a(60)
c Inicia a
do 5 i=1,60
5 a(i)=0
c
c Semilla aleatoria
open(2,FILE='datos.txt')
j=Mod(TIMES(),1000)
do 7 i=1,j
7 x=Rand()
c
c Número de veces que se tiran los 10 dados

```

```

n=1000
epsi=1./float(n)
c
do 10 i=1,n
m=0
do 20 k=1,10
20 m=m+int(6*Rand()+1)
a(m)=a(m)+epsi
10 continue
print*,a
c
cc Escribe el resultado
c do 30 i=1,60
cc30 write(2,*) i,'-->',a(i)*100./float(n)
c close(2)
stop
end

```

---

**Sección:** 4.2

**Tipo:** FORTRAN

**Descripción:** Aproxima la integral correspondiente a los paseos aleatorios en dimensión 3.

---

```

c Integral de los paseos aleatorios D=3
c
c
h=0.005
pi=3.141592654
x=0.
do 10 u1=h, pi, h

```

```

do 10 u2=h, pi, h
do 10 u3=h, pi, h
x=x+h*h*h/(sin(u1/2.)*sin(u1/2.)+sin(u2/2.)*sin(u2/2.)
* +sin(u3/2.)*sin(u3/2.))
10 continue
print*,x
stop
end

```

---

**Sección:** 4.3

**Tipo:** MATLAB

**Descripción:** Simulación discretizada de un proceso de difusión que ocurre entre dos paredes y que tiene una distribución inicial con dos máximos.

---

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PROCESO DE DIFUSIÓN (MOV. BROWNIANO) DISCRETIZADO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 96 LUGARES PARA LAS PARTÍCULAS
x=[1:96];
% DISTRIBUCIÓN INICIAL
p=(96-x).*(1+sin(6*pi*x/96));
p(1:24)=0;
p(88:96)=0;
figure
plot(x,p)
% MATRIZ PARA LAS ITERACIONES
m=96;
A=.5*diag(ones(m-1,1),1)+.5*diag(ones(m-1,1),-1);
% LAS PARTÍCULAS REBOTAN EN LOS EXTREMOS
A(1,1)=.5;

```

```

A(96,96)=.5;
%%% ITERA
figure
for m=1:40
% ITERACIONES MOV. BROWNIANO
% PINTA UNA ITERACIÓN DE CADA 5
plot(x,p);
p=p*A;
p=p*A;
p=p*A;
p=p*A;
p=p*A;
hold on
end
hold off

```

---

**Sección:** Exp. 4.1

**Tipo:** FORTRAN

**Descripción:** Calcula el número de formas de obtener cada puntuación tirando 10 dados.

---

```

c Todas las posibilidades tirando 10 dados n veces
c
c
dimension J(60)
c Inicia a
do 5 i=1,60
5 J(i)=0
c
open(2,FILE='edado.txt')

```

```

c
do 10 i1=1,6
do 10 i2=1,6
do 10 i3=1,6
do 10 i4=1,6
do 10 i5=1,6
do 10 i6=1,6
do 10 i7=1,6
do 10 i8=1,6
do 10 i9=1,6
do 10 ia=1,6
m=i1+i2+i3+i4+i5+i6+i7+i8+i9+ia
J(m)=J(m)+1.0
10 continue
c
c Escribe el resultado
do 30 i=1,60
30 write(2,*) 'Sale un ',i,'-->',J(i),' veces'
close(2)
print*, J
stop
end

```

---

**Sección:** Exp. 4.1

**Tipo:** MATLAB

**Descripción:** Dibuja los datos correspondientes al experimento de lanzar los dados 100 veces.

---

```

n=[10:1:60];
%figure(1)

```



```

plot(n,0.073869756*exp(-3*(n-35).*(n-35)/175),'-');
axis([10 60 0 0.1])
print -deps ga100.eps
D=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
1 1 0 3 2 5 5 5 7 7 10 3 5 6 8 6 4 7 5 2 2 2 0 1 3 ...
0 0 0 0 0 0 0 0 0 0 0 0 0 0]/100;
%hold on
plot(n,D,'-');
print -deps da100.eps
% error máximo
maxerror=max(abs(0.073869756*exp(-3*(n-35).*(n-35)/175)-D))
% error promedio
errorprom=sum(abs(0.073869756*exp(-3*(n-35).*(n-35)/175)-D))/51
hold off

```

---

**Sección:** Exp. 4.1

**Tipo:** MATLAB

**Descripción:** Muestra el histograma al tirar n dados 100 veces y se compara con lo que predice el teorema central del límite.

---

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ¿SE CUMPLE EL TEOREMA CENTRAL DEL LÍMITE?
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SE HALLA EL HISTOGRAMA AL TIRAR n
% DADOS Y SUMAR LAS PUNTUACIONES
n=10000;
y=sum(rand(100,n));
x=[40:60];
hist(y,x);
hold on

```

```

% SE COMPARA CON LA GAUSSIANA
x2=[40:.1:60];
y2=exp(-(x2-50).^2*3/50)/sqrt(2*pi*25/3);
plot(x2,n*y2,'r')
hold off

```

---

**Sección:** Exp. 4.2

**Tipo:** FORTRAN

**Descripción:** Calcula las proporciones en un número a elegir de iteraciones en la urna de Pólya.

---

```

c Urna de Pólya
c
dimension n(10000)
j=Mod(TIME8(),1000)
do 7 i=1,j
7 x=Rand()
c Una bola de cada color
n(1)=1
n(2)=0
c Número de bolas blancas
nb=1
c Número total de extracciones
print*, 'Número de extracciones'
read*,nt
do 10 j=2, nt
c Proporción
print*, j, float(nb)/float(j)
i=int(j*Rand()+1)
n(j+1)=n(i)

```

```
nb=nb+n(i)
10 continue
stop
end
```

---

**Sección:** 5.2

**Tipo:** MATLAB

**Descripción:** Dibuja las trayectorias para el campo de velocidades correspondiente a  $1 - z^{-2}$  que representa el flujo en torno a un obstáculo circular en ausencia de circulación.

---

```
figure(1)
%
% La ecuación está en polares. El ángulo varía entre 0 y pi.
ang=[0.01:0.01:3.14];
%
% El parámetro selecciona diferentes trayectorias
param=[.05:.15:2]';
la=param*(1./sin(ang));
r=.5*(la+sqrt(la.*la+4));
x=r*diag(cos(ang));
y=r*diag(sin(ang));
plot(x',y',x',-y',0,0)
axis([-5 5 -4 4])
axis off
```

