examenlcc.pdf

Kryzux

Laboratorio de Computación Científica

1º Grado en Física

Facultad de Ciencias Físicas
Universidad Complutense de Madrid

TURNO 3. EXAMEN A RESUELTO

Problema 1. **[3 puntos]** Una superficie de revolución se obtiene girando una curva alrededor del eje z, de forma que su altura sólo depende de la distancia $r = \sqrt{x^2 + y^2}$ a dicho eje.

a) Dibujar la superficie de revolución $z = e^{-r}\cos(3r)$, donde $r = \sqrt{x^2 + y^2}$. Esta superficie se dibujará para $0 \le r \le 3$ usando una retícula de 40 divisiones en x e y. Para ello, se definirán primero unas matrices de coordenadas *r* y θ, equiespaciadas en el rango $0 \le r \le 3$, $0 \le \theta \le 2\pi$ y se calcularán:

$$x = r\cos(\theta); \quad y = r\sin(\theta)$$ **(1.5 puntos)**.

```
r=linspace(0,3,40);
theta=linspace(0,2*pi,40);
[rg,thetag]=meshgrid(r,theta);
x=rg.*cos(thetag);
y=rg.*sin(thetag);
z=exp(-rg).*cos(3*rg);
mesh(x,y,z);
```

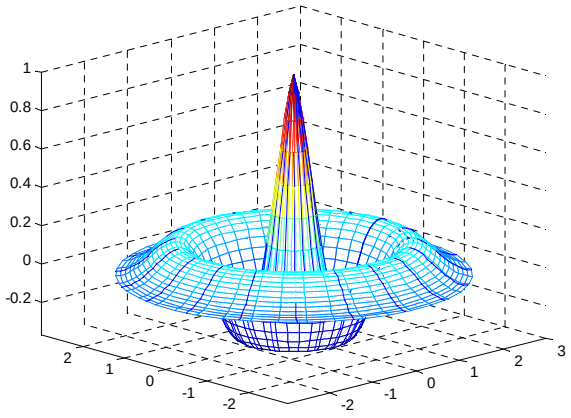
b) Programar una función *turno31(alpha)* que dibuje la curva que engendra la superficie $z = e^{-r}\cos(3r)$ cuando se gira dicha curva un ángulo *alpha* alrededor del eje z. Si definimos *alpha*=0 en el plano xz, la curva vendría definida por las ecuaciones:

$$x = r\cos(alpha); \quad y = r\sin(alpha); \quad z = e^{-r}\cos(3r)$$ **(1 punto)**

```
function turno31(alpha)
r=linspace(0,3,40);
x=r*cos(alpha);
y=r*sin(alpha);
z=exp(-r).*cos(3*r);
plot3(x,y,z);
```

c) Usando la función anterior, superponer a la superficie del apartado a la curva que la engendra para 20 valores del ángulo alpha, equiespaciados entre 0 y 2π **(0.5 puntos).**

```
hold on
for alpha=linspace(0,2*pi,20); turno31(alpha); end
```



Problema 2. [3.5 puntos] Se desea programar un método de Jacobi adaptativo, en el que se reduzca el coeficiente de relajación cada vez que el error obtenido excede un valor máximo.

a) Programar una función *[x,it,w]=turno32(A,b,x0,tol,errormax,factor)* que implemente este método. La función empezará usando el método de Jacobi estándar, es decir, con coeficiente de relajación *w*=1 e irá multiplicando el coeficiente de relajación por *factor* (0<*factor*<1) de forma sucesiva (*w*=1, w=*factor*, *w=factor*^{2}, *w=factor*^{3}, etc.) cada vez que se llegue a un error mayor que *errormax*. En ese caso, además de reducir *w* se retomará el valor de *x0* inicial **(2.5 puntos)**

```
function [x,it,w]=turno32(A,b,x0,tol,errormax,factor)
D=diag(diag(A));
L=A-triu(A);
U=A-tril(A);
w=1;    %Empezamos con coeficiente de relajación 1 (no relajación)
it=0;
x00=x0;  %Almacenar el x0 inicial para poder volver a él.
error=2*tol;  %Para que haga la primera iteracion
while error>tol    %Aun no ha convergido
   x=inv(D)*(b-(L+U)*x0);  %Predicción de Jacobi
   x=w*x+(1-w)*x0;      %Aplicamos relajación a la anterior predicción
   error=norm(x-x0);
   x0=x;
   it=it+1;
   if error>errormax  %Si el error es demasiado grande...
      w=w*factor;     %reducir el coeficiente de relajación...
      x0=x00;         %y volver a la predicción inicial
   end
end
```

b) Calcular la solución del sistema:

$$3x_1 + 2x_2 + 5x_3 = 11$$

$$3x_1 + 2x_2 + x_3 = 3$$

$$-4x_1 + 2x_2 + 3x_3 = 0$$

partiendo de la solución inicial (0,0,0), con tolerancia *tol*=0.001, *factor*=0.9 y *errormax*=100. Indica la solución obtenida y qué coeficiente de relajación has necesitado para obtener la convergencia **(1 punto)**

```
A=[3 2 5; 3 2 1;-4 2 3];
b=[11;3;0];
x0=[0;0;0];
[x,it,w]=turno32(A,b,x0,0.001,100,.9);
```

Tras 596 iteraciones, se obtiene w= 0.6561 y la solución:

```
x =

   0.9991   -1.0008    2.0009
```

QUE LO DIFÍCIL SEA ELEGIR EL CASCO.
HAZLO FÁCIL

PRÁCTICAS EN PISTA DE EXÁMEN OFICIAL DGT
MOTOS CUSTOM MUY MANEJABLES
EXÁMEN EN ALCALA DE HENARES = MÁS FÁCIL

Autoescuelas 2000

autoescuela2000.com

Problema 3. [3.5 puntos]

a) Estimar la probabilidad de que la curva $f(x) = sen(x^2 - 2x)$ esté por encima de la curva $g(x) = cos(x/3)$ cuando x es una variable aleatoria uniformemente distribuida en el intervalo $[0, \pi]$. Para ello se programará una función *turno33(n)*, que cree un vector de *n* números aleatorios en ese intervalo y cuente para cuántos de esos *n* valores se cumple la condición especificada, estimando la probabilidad. La función se ejecutará para varios valores de n para asegurar la convergencia **(2 puntos).**

```
%Programamos una función que estime la probabilidad a partir de n

%simulaciones
function prob=turno33(n)
x=pi*rand(n,1);   %vector con n numeros aleatorios entre 0 y pi
contador=0;
for i=1:n
    if sin(x(i)^2-2*x(i)) > cos (x(i)/3)
        contador=contador+1;
    end
end
prob=contador/n;
```

Ejecutamos esta función para distintos valores de n, para estimar si se produce convergencia:

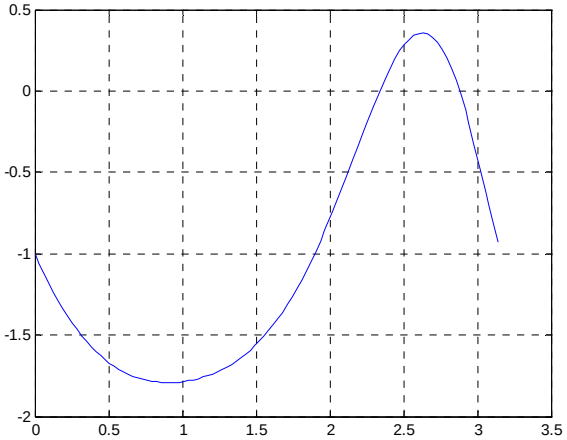
```
>> [turno33(100) turno33(500) turno33(1000) turno33(5000) turno33(10000) turno33(50000)]
ans =
  0.2100   0.1980   0.2030   0.1714   0.1703   0.1734
```

A partir de estos datos, estimamos que la probabilidad es aproximadamente 0.17.

b) Hallar todos los puntos de intersección entre las curvas $f(x)$ y $g(x)$ del apartado anterior. Para ello se calcularán las raíces de la función diferencia $r(x) = f(x) - g(x)$ en el intervalo $[0, \pi]$ usando la función interna de Matlab *fzero* cuantas veces sea necesario. **(1.5 puntos)**.

```
%definimos la función resta como una función inline y la dibujamos
r=inline('sin(x.^2-2*x)-cos(x/3)');
x=linspace(0,pi);
y=r(x);
plot(x,y)
%Vemos que la función tiene dos raíces, una cerca de 2 y otra de 3.
```

```
>> fzero(r,2)

ans =

    2.3384

>> fzero(r,3)

ans =

    2.8789
```



TURNO 3. EXAMEN B RESUELTO

Problema 1. [3.5 puntos] El juego del sudoku consiste en rellenar una matriz de 9x9 de forma que se cumplan las siguientes condiciones:

-Todas las filas de la matriz deben incluir todos los dígitos del 1 al 9, sin repetir ninguno.

-Todas las columnas de la matriz deben incluir todos los dígitos del 1 al 9, sin repetir ninguno.

-Las 9 submatrices de 3x3 en que puede dividirse la matriz completa sin solapamiento deben incluir todos los dígitos del 1 al 9, sin repetir ninguno.

a) Programar una función *test=turno31(vector)* que reciba un vector de 9 elementos y compruebe que este vector incluye todos los dígitos del 1 al 9, sin repetir ninguno. La función deberá advertir cuando falte alguno de los dígitos y cuando haya un dígito repetido, devolviendo *test*=1 si se cumplen las condiciones de sudoku y *test=0* si no se cumplen **(2.5 puntos)**.

```
function test=turno31(vector)
test=1;   %Por defecto, test vale 1. Lo cambiamos si hay problemas
for i=1:9   %Comprobar todos los dígitos, del 1 al 9
   contador=0;
   for j=1:9    %Contar el número de veces que aparece en el vector.
     if vector(j)==i
         contador=contador+1;
     end
   end
   if contador==0
       fprintf('El número %i no aparece\n',i)
       test=0;
   elseif contador>1
       fprintf('El número %i esta repetido\n',i)
       test=0;
   end
end
```

b) Escribir un script que compruebe si una matriz A de 9x9 tiene un sudoku llamando a la función anterior para todas sus filas, columnas y submatrices de 3x3. **(1 punto).**
Sugerencia: podemos extraer la cuadrícula i de la matriz A en Matlab usando la fórmula:
CUAD=A(1+3*floor((i-1)/3):3+3*floor((i-1)/3),1+3*rem(i-1,3):3+3*rem(i-1,3));

```
for i=1:9;
    fprintf('Comprobando fila %i\n',i);
    test=turno31(A(i,:));
    if test==0   break; end
    fprintf('Comprobando columna %i\n',i);
    test=turno31(A(:,i));
    if test==0   break; end
    fprintf('Comprobando cuadricula %i\n',i);
    CUAD=A(1+3*floor((i-1)/3):3+3*floor((i-1)/3),1+3*rem(i-1,3):3+3*rem(i-1,3));
    test=turno31(CUAD);
    if test==0   break; end
end
if(test) disp('SUDOKU!!!!\n');
else   disp('NO ES SUDOKU\n');   end
```

Problema 2. [3 puntos] Dado un sistema de ecuaciones de la forma $Ax = b$ los métodos iterativos estudiados dividen la matriz A en dos partes: $A = A_0 + A_N$, y plantean un proceso iterativo de la forma:

$$A_0 x_0 + A_N x_N = b \qquad \rightarrow \qquad x_N = A_N^{-1}(b - A_0 x_0)$$

siendo x_0 el valor en la iteración previa, y x_N el nuevo valor calculado.

a) Programar una función *[x,it]=turno32(A,b,AN,x0,tol)* que resuelva el sistema de ecuaciones partiendo de la solución inicial *x0* y con tolerancia *tol* usando el método definido por la matriz AN, basado en la notación anterior **(2 puntos)**.

```
function [x,it]=turno32(A,b,AN,x0,tol)
A0=A-AN;
it=0;
error=tol+1;  %Para que siempre haga la primera iteración
while error>tol
    x=inv(AN)*(b-A0*x0);
    error=norm(x-x0);
    it=it+1;
    x0=x;
end
```

b) Indica cómo definirías con Matlab la matriz AN a partir de A para recuperar los métodos de Jacobi y Gauss-Seidel **(1 punto)**

```
%Para el método de Jacobi:
AN=diag(diag(A));
%Para el método de Gauss-Seidel:
AN=tril(A);
```

Problema 3. [3.5 puntos]

a) Estimar la probabilidad de que la curva $f(x) = sen(\sqrt{x^2+1})$ esté por encima de la curva $g(x) = exp(-x/3)$ cuando x es una variable aleatoria uniformemente distribuida en el intervalo $[0, \pi]$. Para ello se programará una función *turno33(n)*, que cree un vector de *n* números aleatorios en ese intervalo y cuente para cuántos de esos *n* valores se cumple la condición especificada, estimando la probabilidad. La función se ejecutará para varios valores de n para asegurar la convergencia **(2 puntos).**

```
%Programamos una función que estime la probabilidad a partir de n

%simulaciones
function prob=turno33(n)
x=pi*rand(n,1);   %vector con n numeros aleatorios entre 0 y pi
contador=0;
for i=1:n
    if sin(sqrt(x(i)^2+1)) > exp (-x(i)/3)
        contador=contador+1;
    end
end
prob=contador/n;
```

Ejecutamos esta función para distintos valores de n, para estimar si se produce convergencia:

```
>> [turno33(100) turno33(500) turno33(1000) turno33(5000) turno33(10000) turno33(50000)]
ans =
  0.6300   0.6640   0.6650   0.6614   0.6719   0.6736
```

A partir de estos datos, estimamos que la probabilidad es aproximadamente 0.67.

b) Hallar todos los puntos de intersección entre las curvas $f(x)$ y $g(x)$ del apartado anterior. Para ello se calcularán las raíces de la función diferencia $r(x) = f(x) - g(x)$ en el intervalo $[0, \pi]$ usando la función interna de Matlab *fzero* cuantas veces sea necesario. **(1.5 puntos)**.

```
%definimos la función resta como una función inline y la dibujamos
r=inline('sin(sqrt(x.^2+1))-exp(-x/3)');
x=linspace(0,pi);
y=r(x);
plot(x,y)
%Vemos que la función tiene dos raíces, una cerca de 0.5 y otra de 2.5
```

```
>> fzero(r,0.5)

ans =

    0.3885

>> fzero(r,2.5)

ans =

    2.4992
```