**Ejercicio 1:**

Encuentre por descomposición LU (usando para esto funciones de Matlab) la solución del siguiente sistema de ecuaciones:



**Solución:**

Ingresamos la matriz A:

>> A = [-4 1 1 0; 1 -4 0 1; 1 0 -4 1; 0 1 1 -4]

A =

-4 1 1 0

1 -4 0 1

1 0 -4 1

0 1 1 -4

Ingresamos el vector b:

>> b = [-200 -400 0 -200]

b =

-200 -400 0 -200

Hallamos la descomposición LU de A:

>> [L,U] = lu(A)

L =

1.0000 0 0 0

-0.2500 1.0000 0 0

-0.2500 -0.0667 1.0000 0

0 -0.2667 -0.2857 1.0000

U =

-4.0000 1.0000 1.0000 0

0 -3.7500 0.2500 1.0000

0 0 -3.7333 1.0667

0 0 0 -3.4286

Estrategia de solución:

Dado el sistema Ax = b podemos descomponer a la matriz de coeficientes A en su descoposición LU de tal manera que:

Ax = b

LUx = b

Llamemos z a Ux y sustituyamos en la ecuación anterior:

Ux = z , Lz = b

A continuación resolvemos el siguiente sistema:

Lz = b

Obteniendo el vector solución de ese sistema.

Por último con el vector z obtenido en el paso anterior, calculamos el vector solución x del sistema Ux = z

>> z = b/L

z =

-328.5714 -457.1429 -57.1429 -200.0000

>> z = inv(L)\*b'

z =

-200.0000

-450.0000

-80.0000

-342.8571

>> x = inv(U)\*z

x =

100.0000

150.0000

50.0000

100.0000

**Ejercicio 2:** Resuelva este sistema por método SOR:



También encuentre el valor óptimo del factor de relajación *w*

***Solución:***

*Función en Matlab para hallar la aproximación de la solución“x” aplicando el método SOR.*

function [ x ] = SOR( A, b, xViejo, w, tol, maxIt )

% Aproxima la solución del sistema Ax = b mediante el método iterativo SOR

% Inputs:

% A matriz de coeficientes del sistema lineal (debe ser una

% matriz cuadrada.

% b vector del lado derecho del sistema lineal.

% xViejo vector que contiene los valores iniciales para la

% solución del sistema lineal.

% w parámetro de relajación.

% tol tolerancia de convergencia. Aplicada a la máxima norma

% de la diferencia entre las sucesivas aproximaciones.

% maxIt cantidad máxima de iteraciones a realizar.

%

% Output:

% x solución aproximada del sistema lineal.

n = length(b);

[f c] = size(A);

if (c ~= n)

disp ('error: la dimensión de la matriz y del vector no son compatibles')

return

end;

xNuevo = zeros(1,n);

for k = 1:maxIt

xNuevo(1) = (1 - w) \* xViejo(1) + w \* (b(1) - sum(A(1,2:n) .\* xViejo(2:n))) / A(1,1)

for i = 2:(n-1)

xNuevo(i) = (1 - w) \* xViejo(i) + w \* (b(i) - sum(A(i,1:i-1) .\* xNuevo(1:i-1)) - sum(A(i,i+1:n) .\* xViejo(i+1:n))) / A(i,i)

end;

xNuevo(n) = (1 - w) \* xViejo(n) + w \* (b(n) - sum(A(n,1:n-1) .\* xNuevo(1:n-1))) / A(n,n)

convergencia = max( abs( xNuevo - xViejo ) );

if(convergencia < tol)

x = xNuevo;

return

else

xViejo = xNuevo;

end;

end

x = xNuevo;

end

Se ingresan los datos en matlab por consola:

>> A = [4 -2 1; 1 5 -3; 2 2 5]

A =

4 -2 1

1 5 -3

2 2 5

>> b = [11 -6 7]

b =

11 -6 7

>> x0 = [0 0 0]

X0 =

0 0 0

Invocamos la función SOR con la matriz de coeficientes A, el vector b del lado derecho del sistema lineal, w=1 (Gauss-Siedel) y un máximo de 10 iteraciones.

>> xRes = SOR(A,b,x0,1,eps,10)

Obtenemos el siguiente resultado en pantalla (k denota la iteración actual):

k =

1

xNuevo =

2.7500 0 0

xNuevo =

2.7500 -1.7500 0

xNuevo =

2.7500 -1.7500 1.0000

k =

2

xNuevo =

1.6250 -1.7500 1.0000

xNuevo =

1.6250 -0.9250 1.0000

xNuevo =

1.6250 -0.9250 1.1200

k =

3

xNuevo =

2.0075 -0.9250 1.1200

xNuevo =

2.0075 -0.9295 1.1200

xNuevo =

2.0075 -0.9295 0.9688

k =

4

xNuevo =

2.0431 -0.9295 0.9688

xNuevo =

2.0431 -1.0273 0.9688

xNuevo =

2.0431 -1.0273 0.9937

k =

5

xNuevo =

1.9879 -1.0273 0.9937

xNuevo =

1.9879 -1.0014 0.9937

xNuevo =

1.9879 -1.0014 1.0054

k =

6

xNuevo =

1.9980 -1.0014 1.0054

xNuevo =

1.9980 -0.9964 1.0054

xNuevo =

1.9980 -0.9964 0.9994

k =

7

xNuevo =

2.0020 -0.9964 0.9994

xNuevo =

2.0020 -1.0008 0.9994

xNuevo =

2.0020 -1.0008 0.9995

k =

8

xNuevo =

1.9997 -1.0008 0.9995

xNuevo =

1.9997 -1.0002 0.9995

xNuevo =

1.9997 -1.0002 1.0002

k =

9

xNuevo =

1.9998 -1.0002 1.0002

xNuevo =

1.9998 -0.9998 1.0002

xNuevo =

1.9998 -0.9998 1.0000

k =

10

xNuevo =

2.0001 -0.9998 1.0000

xNuevo =

2.0001 -1.0000 1.0000

xNuevo =

2.0001 -1.0000 1.0000

xRes =

2.0001 -1.0000 1.0000

Se observa que para la iteración 10, el método converge a la solución correcta.

xRes2 = SOR(A,b,x,1.4322,eps,1000) solución correcta

xRes3 = SOR(A,b,x,1.432**3**,eps,1000) solución con error pequeño

xRes3 =

2.0001 -1.0000 0.9999

xRes4 = SOR(A,b,x,1.5,eps,1000) solución con error

xRes 4=

1.0e+043 \*

0.8106 0.4751 -1.3053

Para hallar un w óptimo, hallamos un w tal que minimice el radio espectral de la matriz de iteración del método SOR:

La siguiente función halla la matriz de iteración M del método SOR y calcula el radio espectral:

function [ r ] = radioEspectralSOR( A, w)

% Calcula el radio espectral de la matriz de iteración del método SOR

% para un parámetro w dado.

% Obtiene la descomposición de A en D, L y U

D = diag(diag(A));

L = tril(A) - D;

U = triu(A) - D;

% Matriz de iteración SOR

M = inv(D + w\*L)\*(-w\*U + D\*(1 - w));

% calcula el radio espectral como el mayor autovalor en valor absoluto de la % matriz de iteración M.

radioEspectral = max( abs( eig( M ) ) );

r = radioEspectral;

end

Usamos esta función para elaborar otra que calcule el w óptimo:

function [ wOpt] = wOptimo( A )

% Calcula un valor óptimo para el parámetro de relajación sucesiva w.

% A: matriz original a partir de la cual se obtiene el radio espectral de

% su matriz de iteración SOR

%inicializa el valor mínimo en el peor caso

%inicializa wOpt en un valor inválido para detectar errores.

min = 2;

wOpt = -1;

% Calcula wOpt con 4 decimales de precisión

% prueba por fuerza bruta valores para w entre 0.0001 y 1.9999

% con pasos de 0.0001

for w = 0.0001:0.0001:1.9999

r = radioEspectralSOR(A,w);

if(r < min)

min = r;

wOpt=w;

end;

end

disp('w optimo:');

disp(wOpt);

end

Podemos observar los siguientes resultados en Matlab al intentar calcular el valor óptimo para w (a modo ilustrativo se muestran las iteraciones desde 0.1 a 1.9 con saltos de 0.1 y teniendo en cuenta especialmente el valor w=0.9178):

Los resultados están devueltos como pares [r,w] donde r es el radio espectral de la matriz y w el valor correspondiente al parámetro de subrelajación.

>> wOptimo(A)

ans =

0.9183 0.1000

ans =

0.8362 0.2000

ans =

0.7538 0.3000

ans =

0.6713 0.4000

ans =

0.5888 0.5000

ans =

0.5073 0.6000

ans =

0.4282 0.7000

ans =

0.3567 0.8000

ans =

0.3118 0.9000

ans =

0.3464 1.0000

**ans =**

**0.3103 0.9178**

ans =

0.4647 1.1000

ans =

0.6133 1.2000

ans =

0.7722 1.3000

ans =

0.9364 1.4000

ans =

1.1043 1.5000

ans =

1.2752 1.6000

ans =

1.4489 1.7000

ans =

1.6251 1.8000

ans =

1.8036 1.9000

**ans =**

**0.9178**

Por cuestiones de espacio no se muestran todas las iteraciones, el experimento fue realizado para valores de w entre 0.0001 y 1.9999 con saltos de 0.0001 (es decir con una precisión de 4 decimales, para todos los valores en ese intervalo) y el resultado obtenido fue exactamente el mismo.

Cabe destacar que este método para optimizar el valor de w no es muy eficiente ya que a mayor cantidad de decimales en la precisión para calcular w, mayor la cantidad de iteraciones. Es un algoritmo bastante lento. Podría llegar a ser de utilidad en casos que la matriz de iteración sea siempre la misma.

**Ejercicio 3:** El método de cuadrados mínimos puede usarse para ajustar una curva a través de n puntos dato 

Si se usa una ecuación cuadrática (parábola) para ajuste de curvas, y(x) se toma como:

 y el error entre la curva teórica y los puntos dato se usa para construir la función: .La función E se minimiza con respecto a a, b, y c para encontrar la parábola de mejor ajuste. Para esto, las condiciones:



Se usan para obtener el siguiente sistema lineal de ecuaciones en a, b y c:



En su computadora escriba un programa para ajustar una ecuación cuadrática a través de n puntos dato.

**Solución:**

Se resuelve mediante la siguiente función llamada cuadradosMinimos que toma como parámetros un conjunto de puntos (xi,yi) dados como un vector de datos “x” y otro vector de datos “y”. Esta función tiene como salida los coeficientes a, b y c correspondientes a la ecuación de la curva:

y(x) = cx2 + bx + a

que aproxima los pares de datos (x,y).

function [ a, b, c ] = cuadradosMinimos( x, y )

% Calcula los coeficientes a, b y c de una ecuación cuadrática

% que mejor aproxima los puntos de datos (x,y)

% Inputs:

% x: vector de abscisas.

% y: vector de ordenadas al orígen.

% Armamos un sistema Au = v donde el vector "u" de incógnitas está formado por

% los coeficientes a, b y c. La matriz de coeficientes está formada por las

% correspondientes sumatorias del lado izquierdo del sistema. Por último el

% vector v está formado por las sumatorias del lado derecho del sistema

% lineal.

n = length(x); %n es la cantidad de datos.

if(n ~= length(y))

disp('La cantidad de datos en x e y deben coincidir')

return

end;

% A: se inicializa la matriz de coeficientes A

A = zeros(3);

% se cargan los coeficientes en A.

% cabe notar que el coeficiente en la posición (i,j) es igual a la

% sumatoria de todos los x\_i elevados a la (i + j - 2).

for i = 1:3

for j = 1:3

A(i,j) = sum(x(1:n).^(i+j-2));

end

end

A

% v: se inicializa el vector del lado derecho del sistema lineal.

v = diag(zeros(3));

% se cargan los coeficientes del vector v.

% notar que dichos coeficientes son la sumatoria de los y\_i multiplicados

% por el x\_i elevado a la (i-1) donde i es el número de fila.

for i = 1:3

v(i) = sum((x(1:n).^(i-1)).\*y(1:n));

end;

v

% Ahora una vez obtenidos la matriz A y el vector b procedemos a hallar la

% solución del sistema lineal que da como resultado un vector con los

% coeficientes [a b c]

u = inv(A)\*v;

a = u(1); b = u(2); c = u(3);

end

Supongamos que en matlab ingresamos los siguientes vectores de datos:

>> x = [-4 -2 -1 0 1 2 4]

x =

-4 -2 -1 0 1 2 4

>> y = [16 4 1 0 1 4 16]

y =

16 4 1 0 1 4 16

>> [a b c] = cuadradosMinimos(x,y)

A =

7 0 42

0 42 0

42 0 546

v =

42

0

546

a =

0

b =

0

c =

1

**Ejercicio 4:** Use el programa generado en el ej. 3 para expresar la variación de temperatura a lo largo de una pared para los siguientes datos:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i (número de punto) | 1 | 2 | 3 | 4 | 5 | 6 |
| xi (distancia, in) | 0 | 1 | 2 | 3 | 4 | 5 |
| yi (temperatura, ºC) | 2.5 | 9.2 | 13.3 | 26.7 | 31.8 | 50.4 |

**Solución:**

>> x =[0 1 2 3 4 5]

x =

0 1 2 3 4 5

>> y = [2.5 9.2 13.3 26.7 31.8 50.4]

y =

2.5000 9.2000 13.3000 26.7000 31.8000 50.4000

>> [a b c] = cuadradosMinimos(x,y)

a =

3.1893

b =

3.4932

c =

1.1339

A través de la siguiente figura podemos observar que la función hallada es una buena aproximación a los datos de entrada:

y(x) = 1.1339 x2 + 3.4932 x + 3.1893



La figura anterior fue obtenida en matlab ingresando los siguientes comandos en matlab:

//grafica los puntos aislados en color verde.

>> plot(x,y, '\*g')

//mantiene abierta la ventana con el grafico recientemente creado.

>> hold on

// Dibuja sobre el mísmo gráfico, la función cuadrática aproximada con los coeficientes obtenidos //mediante el método de los cuadrados mínimos.

>> fplot('(1.1339)\*x^2 + (3.4932)\*x + 3.1893', [0 5])