Trabajo práctico Nº3

Resolución de sistemas lineales   


Metodos de computación científica -2013

Brenda Soledad Dilschneider L.U. 92774

Profesora: Dra. Nélida Beatriz Brignole

**Ejercicio 1**

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



Desarrollo:

Ingreso la matriz A y el vector b y hallo la descomposición L mediante la función lu(A) de Matlab:

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

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

>> [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

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



También encuentre el valor óptimo del factor de relajación w

Desarrollo***:***

*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];

>> b = [11 -6 7];

>> x0 = [0 0 0];

Invoco 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)

Obtengo 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

Se observan 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**

Resuelva el sistema:

1. Usando el método de Jacobi.
2. Usando el método de Gauss-Seidel.
3. ¿Cuánto mas rápida es la convergencia en la parte b) que en la parte a)?

Desarrollo:

1. Método de Jacobi

Para poder utilizar el método de Jacobi, intercambiamos la ecuación 3 por la 2

Despejo la incógnita correspondiente:

Primera iteración:

Tomo .

Segunda iteración:

Tercera iteración:

Cuarta iteración:

Quinta iteración:

Sexta iteración:

Séptima iteración:

Verificación:

1. Método de Gauss-Seidel

Intercambiamos la ecuación 3 por la 2

Despejo la incógnita correspondiente:

Primera iteración:

Segunda iteración:

Tercera iteración:

Cuarta iteración:

Verificación:

1. El método de Gauss-Seidel tiene 3 iteraciones menos que el método de Jacobi. Converge mas rápido debido a que se van utilizando los valores de las incógnitas recién calculados en la misma iteración.