PRACTICA 2

RESOLUCION DE ECUACIONES LINEALES

Fernando Javier Nossa Chaparro

Carlos Rizo Maciá

Estudiantes MSC. computing graphics and video game development.

Universidad Rey Juan Carlos

2019

***Resumen*—El contenido de este documento resume nuestra experiencia en la práctica de cómo hallar raíces numéricas de funciones lineales en el laboratorio. Esta práctica enseña la aplicación de resolución de un sistema de ecuaciones.**

I. INTRODUCCION

En el transcurso de la práctica crearemos un programa en Matlab que aplique los métodos de Gauss-Seidel y Jacobi con el objetivo de encontrar una solución aproximada a cada una de las incógnitas propuestas en el sistema.

Se nos propone aproximar la solución de los siguientes sistemas de ecuaciones:

Ejercicio 1:

$$\begin{cases} 3x_1 - 0,1x_2 - 0,2x_3 = 7,85 \\ 0,1x_1 + 7x_2 - 0,3x_3 = -19,3 \\ 0,3x_1 - 0,2x_2 + 10x_3 = 71,4 \end{cases}$$

Representación matricial:

```
A=
[  3, -0.1,-0.2;
 0.1,  7, -0.3;
 0.3, -0.2, 10]

b=
[7.85,-19.3,71.4]
```

Ejercicio 2:

$$\begin{cases} 5x_1 + 2x_2 - 1x_3 + 1x_4 = 12 \\ 1x_1 + 7x_2 + 3x_3 - 1x_4 = 2 \\ -1x_1 + 4x_2 + 9x_3 + 2x_4 = 1 \\ 1x_1 - 1x_2 + 1x_3 + 4x_4 = 3 \end{cases}$$

Representación matricial:

```
A=
[5,  2, -1, 1;
 1,  7,  3, -1;
-1,  4,  9, 2;
 1, -1,  1, 4]
b =
[12,2,1,3]
```

Imagen 1: Planteamiento del ejercicio

II. GAUSS-SEIDEL

Antes de empezar a aplicar Gauss-Seidel debemos recordar que si un sistema es diagonalmente dominante es condición suficiente para saber que este converge. Sin embargo, que no lo sea no significa que el sistema no pueda converger. Teniendo esto en cuenta nos ahorramos los cálculos de comprobación de si la matriz es diagonalmente convergente.

$$x_i^{\,k} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{\,k} - \sum_{j=i+1}^{n} a_{ij} x_j^{\,k-1}}{a_{ii}}$$

Imagen 2: Fórmula de aproximación Gauss Seidel

Cada ecuación del sistema se resuelve para una variable $x_i^{\,k}$ siendo i el número de la incógnita que estamos resolviendo y k la iteración de cálculo en la que estamos. Para poder empezar a calcular necesitamos de un $x_i^{\,k-1}$ que funcione de valor inicial. Para eso aplicaremos el vector nulo (0,0,0).

Consideramos la variable calculatedError y la igualamos a infinito antes de empezar el método. De esta forma nos aseguramos que la primera iteración siempre se haga y que el siguiente valor calculado de calculatedError sea menor que el inicialmente asignado.

```
while (tolerancia < calculatedError)
```

Imagen 3: Condición de parada

Esto también nos asegura que no dejaremos de iterar hasta conseguir una diferencia menor a la tolerancia asignada.

Para cada fila del sistema procedemos a calcular los valores necesarios para cada sistema. Cada sumatorio se calcula en su propio bucle y se va acumulando. Cabe destacar que los

sumatorios se resetean cada vez que se inicia el cálculo para una nueva fila por qué en el caso de no hacerlo estaremos aplicando valores en la fila equivocada.

```
for i=1:n
%Resetear los valores de los sumatorios
para que no ensucien los siguientes
cálculos
   sumK=0;
   sumKminus=0;
   for j=1:(i-1)
       sumK = sumK +  A(i,j)*x(j);
   end
   for j=i+1:n
       sumKminus = sumKminus +
A(i,j)*xMinusOne(j);
   end
x(i)= (b(i)- sumK -sumKminus) /A(i,i);
end
```

Imagen 4: Aplicación de la fórmula

Por último aplicamos antes de terminar cada fila calculamos el nuevo valor de x_i^k aplicando la fórmula. Una vez hemos terminado de calcular nos preparamos para la siguiente iteración. Incrementamos k, calculamos el error que hemos cometido y, lo más importante, igualamos xMinusOne a x. Esto se debe a que las x_i^{k-1} de la siguiente iteración son mis x_i^k actuales.

Tras superar la tolerancia asignada (en este caso 0.001) obtenemos los siguientes valores para el ejercicio 1:

X en k=1	X en k=2	X en k=3
2.616667	2.990557	3.000032
-2.794524	-2.499625	-2.499988
7.005610	7.000291	6.999999

Imagen 5: Valores obtenido por Gauss Seidel en el ejercicio 1 con una tolerancia de 0.001.

Y para el ejercicio 2:

X en k=1	X en k=3	X en k=7	X en k=11
2.400000	2.603965	2.712048	2.725423
-0.057143	-0.302859	-0.391736	-0.402545
0.403175	0.547382	0.625600	0.635056
0.034921	-0.113552	-0.182346	-0.190756

Imagen 6: Valores obtenido por Gauss Seidel en el ejercicio 2 con una tolerancia de 0.001.

Lo primero que podemos observar es la diferencia de iteraciones en encontrar valores adecuados. El ejercicio 1 solo necesita 3 iteraciones mientras que el 2 necesita de 11 iteraciones. Esto puede ser debido a que el segundo sistema consta de una incógnita más creando así una dependencia más que hace más costoso encontrar el punto de convergencia.

III. JACOBI

De la misma manera que en el método anterior aplicamos una fórmula para cada una de las incógnitas del sistema en función de las otras variables y para el valor inicial x_i^{k-1} aplicaremos el vector nulo (0,0,0).

$$x_i^k = \frac{b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{k-1}}{a_{ii}}$$

Imagen 7: Fórmula de aproximación de Jacobi

Este método se diferencia de Gauss-Seidel en que el método de Jacobi no utiliza un valor de la iteración actual, solo la anterior. Esto conlleva que la convergencia de este método sea más lenta.

Por tanto, aplicamos las mismas operaciones que antes pero solo calculamos y aplicamos sumKminus.

```
for j=1:n
       if(i~=j)
       sumKminus = sumKminus +
A(i,j)*xMinusOne(j);
       end
end

x(i)= (b(i) -sumKminus) /A(i,i);
```

Imagen 8: cálculo de la iteración con Jacobi.

Tras superar la tolerancia asignada (en este caso 0.001) obtenemos los siguientes valores para el ejercicio 1:

X en k=1	X en k=2	X en k=3
2.616667	3.000762	3.000806
-2.757143	-2.488524	-2.499738
7.140000	7.006357	7.000207

Imagen 9: Valores obtenido por Jacobi en el ejercicio 1 con una tolerancia de 0.001.

Se puede ver que el primer valor de x_1 en k=1 es igual tanto en Jacobi como en Gauss-Seidel ya que estos realizan el mismo cálculo con los mismos valores en este punto. Una vez miramos x_2 ya empezamos a ver las diferencias entre ambos métodos. Ambos convergen en 3 iteraciones en este caso pero curiosamente x_1 en k=3 parece que esté aumentando, alejándose así del valor objetivo que sabemos que es 3. Volvemos a aplicar Jacobi con una tolerancia aún menor de 0.00001.

X en k=4	X en k=5
3.000022	2.999999
-2.500003	-2.500001
6.999981	6.999981

Imagen 10: Valores obtenido por Jacobi en el ejercicio 1 con una tolerancia de 0.00001.

Como podemos observar esto último no es realmente un problema ya que con una tolerancia menor sigue convergiendo hacia nuestro valor objetivo.

Y para el ejercicio 2:

X en k=1	X en k=7	X en k=16	X en k=20
2.400000	2.606683	2.717586	2.724125
0.285714	-0.290548	-0.395108	-0.401137
0.111111	0.519748	0.626946	0.633303
0.750000	-0.075064	-0.182637	-0.188902

Imagen 11: Valores obtenido por Jacobi en el ejercicio 2 con una tolerancia de 0.001.

Aquí realmente vemos la diferencia entre Gauss-Seidel y Jacobi. Para conseguir unos valores con la misma tolerancia ha tardado 9 iteraciones más. Esto significa que para sistemas de ecuaciones con 50 incógnitas estas diferencia puede agrandar el tiempo de cálculo mucho.

IV. CONCLUSIONES

Tras aplicar Gauss-Seidel y Jacobi para la resolución de ecuaciones lineales podemos concluir que ambos son válidos pero con diferencias notables. Jacobi es más fácil de entender y programar pero la diferencia de la velocidad de convergencia con Gauss-Seidel se puede hacer abismal para sistemas mucho más grandes. Gauss-Seidel requiere de un cálculo más y de un paso más de entendimiento pero esta pequeña diferencia de dificultad no compensa la pérdida de tiempo en cálculo siempre que se utilicen sistemas muy grandes. Si estamos calculando sistemas más pequeños y/o no nos interesa la velocidad de cómputo Jacobi es tan buena solución como Gauss-Seidel ya que en ambos casos la aproximación a la solución es igual de válida.