li2003introductionli2003introduction3

Tarea 2: Factorización de Cholesky y raices de un polinomio

Sebastián Valencia Calderón
201111578

1 Introducción

Los sistemas de ecuaciones lineales, aparecen en una gran variedad de aplicaciones en ciencia e ingeniería. Su naturaleza algorítmica, da lugar al diseño de herramientas computacionales para hallar la solución de estos sistemas. Existen dos categorías de los métodos que solucionan estos sistemas de manera secuencial. Los métodos iterativos y los métodos directos. Métodos como el de Jacobi, o Gauss-Seidel, pertenecen a la primera categoría, mientras a la última, pertenecen métodos más naturales, como la eliminación Gaussiana. De los diferentes métodos, interesan la eficiencia, es decir, la velocidad de convergencia y la exactitud de la aproximación de esta respuesta.

Un sistema lineal, se puede describir a muy alto nivel como un conjunto de ecuaciones lineales en varias variables. La representación natural que resulta conveniente en los métodos iterativos no es como anteriormente matricial, sino la algebraica general. Es decir, un sistema de ecuaciones de la siguiente forma:

$$
\begin{aligned}
a_{11} \times x_1 + a_{12} \times x_2 + \cdots + a_{1n} \times x_m &= b_1 \\
a_{21} \times x_1 + a_{22} \times x_2 + \cdots + a_{2n} \times x_2 &= b_2 \\
&\vdots \\
a_{m1} \times x_1 + a_{m2} \times x_2 + \cdots + a_{mn} \times x_m &= b_n
\end{aligned}
$$

Dado un sistema lineal algebraico, se pretende estudiar un conjunto básico de los métodos iterativos, de manera más precisa, los dos enunciados anteriormente: los métodos de Jacobi y de Gauss-Seidel. Con el desarrollo de estos ejercicios, se pretende cumplir con los siguientes objetivos:

1. Comprender la necesidad de implementación de métodos iterativos en la solución de sistemas lineales.
2. Implementar e identificar las características de diferentes métodos iterativos de solución de sistemas lineales.
3. Introducir casos de uso de sistemas lineales que requieren el uso de métodos iterativos.

2 Procedimiento

Para cumplir los objetivos enumerados anteriormente, se desarrollan algoritmos para resolver de manera iterativa sistemas de ecuaciones lineales, de manera específica, se desarrolla una función que implementa la iteración de Jacobi, y otra par implementar la iteración de Gauss-Seidel. Una vez los algoritmos están desarrollados y comprendidos, se procede a probarlos haciendo uso de un sistema lineal cuya solución analítica y algebraica es bien conocida, de tal manera que se puede comparar el resultado numérico obtenido a través de los algoritmos, y el valor conocido para distintos tamaños de la matriz (la comparación, se hace con los métodos de solución nativos en MATLAB).

Una vez se tenga comprensión de estos conceptos y de la geometría de las soluciones propuestas, se aplican los algoritmos para resolver sistemas de ecuaciones lineales que surgen en la vida real, de manera más específica, en la solución de circuitos lineales, donde se sabe que se trabaja con sistemas de ecuaciones lineales de a veces muchas variables.

3 Resultados

A continuación, se exponen los resultados, las metodologías propuestas para los análisis y las herramientas de ejecución para cada uno de los problemas propuestos.

1. Se implementan en MATLAB, los algoritmos, uno para la iteración de Jacobi, otro para la iteración de Gauss-Seidel. Para garantizar un entendimiento completo y detallado, se incluye a continuación, la deducción de los algoritmos.

    - **Métodos iterativos.** La solución de sistemas lineales de naturaleza dispersa y de gran tamaño, se ha convertido en un eje central de investigación en computación científica, y computación de alto desempeño. El tamaño de los problemas solucionados con este tipo de técnicas, aumenta con el paso del tiempo. Dentro de la categoría de métodos iterativos para la solución de sistemas lineales, existen dos sub-categorías, los métodos iterativos estacionarios, técnica inventada por Gauss, y los métodos iterativos no estacionarios, una metodología más moderna, que escapa del alcance de este informe. Una definición más precisa y exhaustiva, está contenida en [10].

      Los métodos iterativos para la solución de sistemas de ecuaciones lineales, comienzan con una estimación de la respuesta $\hat{x}^{(0)}$, y sucesivamente, se mejora con base en las aproximaciones anteriores, hasta alcanzar una solución en el radio de convergencia deseado. Teóricamente, un número infinito de iteraciones es requerido, sin embargo, en práctica, las iteraciones pueden terminar al alcanzar un residuo, u otra métrica de error tan pequeños como se desee. Los métodos iterativos son fáciles de proponer, de implementar y en ocasiones, dependiendo de la naturaleza del problema, pueden ser muy eficientes.

La deducción de los sistemas de interés, se muestra a continuación haciendo uso de un sistema general de tamaño 3. Dado el sistema de ecuaciones lineales:

$$a_{11} \times x_1 + a_{12} + \times x_2 + a_{13} \times x_3 = b_1$$
$$a_{21} \times x_1 + a_{22} + \times x_2 + a_{23} \times x_3 = b_2$$
$$a_{31} \times x_1 + a_{32} + \times x_2 + a_{33} \times x_3 = b_3$$

A partir de ellas, es posible despejar para cada ecuación, la incógnita relacionada a ella: $eq^{(i)} \Rightarrow x_i$:

$$x1 = \frac{[b_1 - a_{12} \times x_2 - a_{13} \times x_3]}{a_{11}}$$
$$x2 = \frac{[b_2 - a_{21} \times x_1 - a_{23} \times x_3]}{a_{22}}$$
$$x3 = \frac{[b_3 - a_{31} \times x_1 - a_{32} \times x_2]}{a_{33}}$$

Dado un vector inicial $\hat{x}^{(0)}$, se puede calcular el vector $\hat{x}^{(k)}$, en el sistema descrito anteriormente con la siguiente regla de sustitución:

$$x1^{(k)} = \frac{\left[b_1 - a_{12} \times x_2^{(k-1)} - a_{13} \times x_3^{(k-1)}\right]}{a_{11}}$$
$$x2^{(k)} = \frac{\left[b_2 - a_{21} \times x_1^{(k-1)} - a_{23} \times x_3^{(k-1)}\right]}{a_{22}}$$
$$x3^{(k)} = \frac{\left[b_3 - a_{31} \times x_1^{(k-1)} - a_{32} \times x_2^{(k-1)}\right]}{a_{33}}$$

De manera análoga, es posible proponer la siguiente regla de sustitución iterativa:

$$x1^{(k)} = \frac{\left[b_1 - a_{12} \times x_2^{(k-1)} - a_{13} \times x_3^{(k-1)}\right]}{a_{11}}$$
$$x2^{(k)} = \frac{\left[b_2 - a_{21} \times x_1^{(k)} - a_{23} \times x_3^{(k-1)}\right]}{a_{22}}$$
$$x3^{(k)} = \frac{\left[b_3 - a_{31} \times x_1^{(k)} - a_{32} \times x_2^{(k)}\right]}{a_{33}}$$

La primera forma de sustitución iterativa, es la de **Jacobi**, la segunda la de **Gauss-Seidel**. La deducción descita anteriormente, es una variación de la referencia [1], y de la referencia [4] . Una generalización natural que surge a partir de las anteriores ilustraciones para el método de Jacobi y el de Gauss-Seidel,

son respectivamente:

Data: $A \in \mathbb{R}^{n \times n} \mid a_{ii} \in \mathbb{R} - \{0\}$
Data: $\hat{b} \in \mathbb{R}^n$
Data: $\hat{x}^{(0)} \in \mathbb{R}^n$

for $i \in [1,\ 2,\ \ldots,\ n]$ **do**
$\quad x_i^{(k)} = \frac{b_i - \sum_{j=1}^{i-1}\left[a_{ij} \times x_j^{(k-1)}\right] - \sum_{j=i+1}^{n}\left[a_{ij} \times x_j^{(k-1)}\right]}{a_{ii}}$
end

Algoritmo 1: Iteración de Jacobi

Data: $A \in \mathbb{R}^{n \times n} \mid a_{ii} \in \mathbb{R} - \{0\}$
Data: $\hat{b} \in \mathbb{R}^n$
Data: $\hat{x}^{(0)} \in \mathbb{R}^n$

for $i \in [1,\ 2,\ \ldots,\ n]$ **do**
$\quad x_i^{(k)} = \frac{b_i - \sum_{j=1}^{i-1}\left[a_{ij} \times x_j^{(k)}\right] - \sum_{j=i+1}^{n}\left[a_{ij} \times x_j^{(k-1)}\right]}{a_{ii}}$
end

Algoritmo 2: Iteración de Gauss-Seidel

La referencia [**?**], contiene una sugerencia para la implementación real del algoritmo. Con base en esta sugerencia, se implementan en MATLAB, los métodos iterativos de Jacobi y de Gauss-Seidel. El script 2, es una implementación del primer método (Jacobi), el script 3, implementa la iteración de Gauss-Seidel.

2. Para validar los algoritmos implementados, se propone el uso de una matriz tridiagonal, la cual, por su naturaleza dispersa (en inglés sparse), se presta para ser parte de un sistema de ecuaciones lineales que pueda solucionarse por medio del uso de los algoritmos implementados. El uso de los algoritmos implementados, se documenta en el siguiente script.

Script 1: Uso de los scripts previamente implementados.

```
1 A = [ 2 -1 0 ; -1 3 -1 ; 0 -1 2];
2 b = [1 8 -5];
3
4 [kjac, xjac] = jacobi(A, b, zeros(3, 1));
5 [kgas, xgas] = gaussseidel(A, b, zeros(3, 1));
```

En este caso, con $k = 22$, la iteración de Jacobi converge a la solución, mientras la iteración de Gauss-Seidel, lo hace en $k = 10$. En ambos casos, el resultado (`xjac`, `xgas`), es:

$$\begin{pmatrix} 2.0 \\ 3.0 \\ -1.0 \end{pmatrix}$$

Dado un $n \geq 3$. $A \in \mathbb{R}^{n \times n}$ es una matriz tridiagonal, si tiene que posee únicamente elementos no nulos en la diagonal principal y en las dos adyacentes a esta, de resto, todos los elementos son 0.

$$a_{ij} = 0 \leftrightarrow |i - j| > 1 \; i, \; j \in \{1, 2, \ldots, n\}$$

Este tipo de matrices, aparecen frecuentemente en la solución numérica de ecuaciones diferenciales, y poseen la particularidad de que su factorización LU, es computacionalmente barata. Un ejemplo de este tipo de matriz es:

$$\begin{pmatrix} 4.0 & -1.0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1.0 & 4.0 & -1.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1.0 & 4.0 & -1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1.0 & 4.0 & -1.0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1.0 & 4.0 & -1.0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1.0 & 4.0 & -1.0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1.0 & 4.0 & -1.0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1.0 & 4.0 \end{pmatrix}$$

Más información relacionada con la forma en la que algunos algoritmos aprovechan esta forma, se encuentra en la referencia [**?**]. Para validar la implementación, de los algoritmos, se crea un script de generación de sistemas lineales donde la matriz de coeficientes es tridiagonal. El script que materializa dicha intensión es el script 4. Los métodos para comparar con los implementados aquí, son:

- `inv(A) * b'`, el objetivo de esta expresión, es hallar la inversa de una matriz A, y multiplicar A^{-1} y el vector b, para obtener el valor del vector x. El único parámetro de `inv()`, es la matriz a invertir, este no se implementa, reduciendo el sistema aumentado $[\ A \mid b\ ]$, y aplicando eliminación de Gauss, pues MATLAB, documenta demoras en la aplicación de esta función para resolver un sistema, pues interviene ademas, la multiplicación de matrices ($\mathrm{O}(n^3)$).
- `linsolve()`, recibe como parámetros la matriz A, y el vector b. Su funcionamiento, fue explicado anteriormente.
- `rref()`, recibe como parámetro el sistema aumentado $[\ A \mid b\ ]$, y aplicando eliminación de Gauss, determina la forma escalonada reducida para obtener las respuestas requeridas.
- `mldivide()`, recibe como parámetro la matriz A, y el vector b. Aplicando optimización de mínimos cuadrados, este llega a la respuesta en el caso de que A sea no singular.

La comparación, se hace con el script 5. Los residuos en cada caso, se relacionan en la siguiente figura, la metodología para relacionar los residuos, y los métodos, se consultó en las referencias [**?**] y [5].

	10	20	50	75	100	500
JACOBI	3.8e-05	8.44e-05	8.47e-05	9.41e-05	5.78e-05	6.83e-05
GAUSSEIDEL	2.84e-05	4.01e-05	1.93e-05	3.18e-05	3.35e-05	2.63e-05
RREF	0.000182	0.00403	0.0203	0.0297	0.0435	0.483
INV	1.78e-14	7.33e-14	2.18e-13	1.41e-13	2.8e-13	6.85e-13
LINSOLVE	3.97e-15	4.28e-14	6.81e-14	7.11e-14	8.17e-14	2.12e-13
MLDIVIDE	2.01e-14	4.05e-14	9.99e-14	1.04e-13	1.35e-13	2.69e-13

`Figura 1: Errores obtenidos con la solución numérica y la algebraica de cada sistema para un orden o tamaño variante. El eje x es el tamaño del problema, y el eje y el error absoluto medido para cada método de solución.`

Puede observarse que el peor desempeño según el error obtenido, es el de la forma escalonada reducid seguido por el método iterativo de Jacobi. Esto, puede deberse a la naturaleza del problema, pues una matriz de este tipo, es bastante fácil (rápido) de resolver mediante una factorización LU, la cuál se sabe, está respaldada por la eliminación de Gauss, que a su vez, respalda los metodos `inv`, `linsolve`, `mldivide`. Mientras que la eliminación de Gauss aprovecha la estructura o forma de la matriz, los métodos iterativos, no tienen en cuenta estos aspectos que pueden aprovecharse para mejor convergencia o reducir el error.

4 Conclusiones

Mediante el desarrollo del laboratorio propuesto, se pudo verificar la importancia del diseño y análisis de algoritmos (en particular, los iterativos), en el álgebra lineal numérica o cálculo o álgebra matricial. Conocer algoritmos para resolver sistemas de ecuaciones lineales, permite hacer uso de máquinas digitales para implementar, obtener y verificar la solución de sistemas de ecuaciones lineales. Además de ver esto, se estudió otra metodología para la solución numérica de los sistemas de ecuaciones lineales, la metodología iterativa estacionaria, la cual tiene en cuenta las soluciones pasadas (Jacobi), o las presentes y pasadas (Gauss-Seidel), para llegar a una mejor aproximación de la solución. Conocer los métodos directos, y los iterativos para la solución de sistemas lineales, permite identificar la pertinencia de uso en los diferentes casos posibles. Ademas de esto, se puede concluir lo siguiente:

- La pertinencia de métodos iterativos, resulta precisa en matrices donde la estructura no en forma sino contenido, se pueda explotar, es decir, resulta más conveniente el uso de este tipo de algoritmos con matrices dispersas, lo cual puede llevar a una convergencia rápida en términos de eficiencia de recursos usados. Sin embargo, este tipo de matrices tiene una forma que puede ser explotada en cierta disposición (tridiagonal o en banda), a través de los métodos directos de solución. Conocer la forma y fondo de las matrices, resulta conveniente a la hora de determinar el mejor algoritmo a usar, o por qué es posible descartar algunos de ellos.

- El uso de aproximaciones iterativas a partir de un valor supuesto o adivinado, resulta una forma intuitiva de resolver un problema numérico, en este caso, la deducción es natural si se despeja la incógnita relacionada a cada ecuación. Esta sencilla observación, se presta para implementar un algoritmo robusto y estable que aprovecha el entendimiento del problema y las estructuras matemáticas detrás de el. Es decir, es demasiado importante para el diseño de algoritmos numéricos conocer la estructura formal o matemática de cada problema.

5 Bibliografía

[1] Isaacson, E. and Keller, H.B. *Analysis of Numerical Methods.* 1994. Dover Publications - Dover Books on Mathematics. Página 61 - 62.

[2] Yang, W.Y. and Cao, W. and Chung, T.S. and Morris, J. *Applied Numerical Methods Using MATLAB.* 2006. Wiley. Página 98 - 103.

[3] Zalizniak, V. *Essentials of Scientific Computing: Numerical Methods for Science and Engineering.* 2008. Elsevier Science. Página 34 - 42.

[4] Suli, E. and Mayers, D.F. *An Introduction to Numerical Analysis.* 2003. Cambridge University Press. Página 87 - 98.

[5] Heath, M.T. *Scientific Computing: An Introductory Survey.* 2005. McGraw-Hill. Página 67.

[6] Cheney, E.W. and Kincaid, D.R. *Numerical Mathematics and Computing.* 2012. Cengage Learning. Página 322 - 328.

[7] Rosloniec, S. *Fundamental Numerical Methods for Electrical Engineering.* 2008. Springer Berlin Heidelberg - Lecture Notes in Electrical Engineering. Página 17 - 27.

6 Scripts

Script 2: Iteración de Jacobi.

```
function [k, guess] = jacobi(A, B, guess)
    kmax = 100; tol = 10e-10; epsilon = 0.5e-4;

    n = size(A, 1);
    y = zeros(n, 1);
    exit = 0;

    for k = 1:kmax
        y = guess;
        for i = 1: n
            sum = B(i);
            diag = A(i, i);

            if(abs(diag) < tol)
                error('Diagonal element too small');
            end;

            sum = sum - A(i, 1:(i - 1)) * y(1:(i - 1)) - A(i, (i + 1):n) * y
                ((i + 1):n);
            guess(i) = sum / diag;

        end;

        if norm(guess - y) < epsilon
            exit = 1;
            break;
        end
    end;

    if(~exit)
        error('Diagonal element too small');
    end;
end
```

Script 3: Iteración de Gauss-Seidel.

```
function [k, guess] = gaussseidel(A, B, guess)
    kmax = 100; tol = 10e-10; epsilon = 0.5e-4;

    n = size(A, 1);
    y = zeros(n, 1);
    exit = 0;

    for k = 1:kmax
        y = guess;
        for i = 1: n
            sum = B(i);
            diag = A(i, i);

            if(abs(diag) < tol)
                error('Diagonal element too small');
            end;

            sum = sum - A(i, 1:(i - 1)) * guess(1:(i - 1)) - A(i, (i + 1):n)
                 * guess((i + 1):n);
            guess(i) = sum / diag;

        end;

        if norm(guess - y) < epsilon
            exit = 1;
            break;
        end
    end;

    if(~exit)
        error('Diagonal element too small');
    end;
end
```

Script 4: Generación de un sistema lineal con una matriz tridiagonal.

```
function [A, b] = gensystem(n)
    A = diag(4*ones(1,n),0)+diag(- 1*ones(1,n-1),1)+diag(-1*ones(1,n-1),-1);
    b = randi(100, n, 1);
end
```

Script 5: Solución de varios sistemas tridiagonales, haciendo uso de varios métodos.

```
residuals = zeros(6, 6);
sizes = [5 20 50 75 100 500];

% JACOBI
for i=1:6
    sz = sizes(i);
    [A b] = gensystem(sz);
    [k x] = jacobi(A, b, zeros(sz, 1));
    residual = norm(b - A * x);
    residuals(1, i) = residual;
end;

% GAUSS-SEIDEL
for i=1:6
    sz = sizes(i);
    [A b] = gensystem(sz);
    [k x] = gaussseidel(A, b, zeros(sz, 1));
    residual = norm(b - A * x);
    residuals(2, i) = residual;
end;

% RREF
for i=1:6
    sz = sizes(i);
    [A b] = gensystem(sz);
    augmented = [A b];
    x = rref(augmented);
    x = x(:, end);
    residual = norm(b - A * x);
    residuals(3, i) = residual;
end;

% INV
for i=1:6
    sz = sizes(i);
    [A b] = gensystem(sz);
    x = inv(A) * b;
    residual = norm(b - A * x);
    residuals(4, i) = residual;
end;

% LINSOLVE
for i=1:6
    sz = sizes(i);
    [A b] = gensystem(sz);
```

```
46      x = linsolve(A, b);
47      residual = norm(b - A * x);
48      residuals(5, i) = residual;
49  end;
50
51  % MLDIVIDE
52  for i=1:6
53      sz = sizes(i);
54      [A b] = gensystem(sz);
55      x = A\b;
56      residual = norm(b - A * x);
57      residuals(6, i) = residual;
58  end;
59
60  plotter(residuals, parula);
61  set(gcf, 'Position', [400 400 700 700]);
62  saveas(gcf, '../img/comparisson.png');
```

Lista de Scripts