Solución de Sistemas de Ecuaciones lineales dispersas y de gran dimensión

Francisco José Correa
Universidad EAFIT
fcorrea@eafit.edu.co

Marcela Gutiérrez Mejía
Universidad EAFIT
mgutie12@eafit.edu.co

Juan David Jaramillo
Universidad EAFIT
jjaram14@eafit.edu.co

Resumen

En este documento presentamos un análisis de los tiempos de cómputo y la calidad de las soluciones obtenidas al resolver sistemas de ecuaciones tridiagonales y pentadiagonales con métodos directos e iterativos. Para problemas de gran dimensión, se obtienen mejores soluciones con los métodos iterativos, mientras que para los problemas con dimensión menor a 50×50 pueden obtenerse mejores soluciones con los métodos directos.

Palabras Claves: *Matrices Dispersas, Matrices Banda, Matrices de gran dimensión, Métodos de Almacenamiento, Factorización de Matrices.*

1. Introducción

En la solución de sistemas de ecuaciones lineales, algunas de las matrices asociadas pueden tener una estructura que es posible aprovechar para mejorar la eficiencia del método de solución. La complejidad de los algoritmos puede variar dependiendo de la estructura de la matriz y en algunos casos, representa grandes diferencias al momento realizar cálculos con ellas. Se busca utilizar métodos que permitan agilizar los cálculos y ser lo más eficiente posibles en cuanto al uso de recursos.

Dado que muchos problemas requieren el uso de matrices de gran tamaño, es necesario encontrar métodos y procedimientos que faciliten su tratamiento y la solución de sistemas de ecuaciones asociadas. Para esto es necesario valerse de las propiedades que pueda tener una matriz, por ejemplo, evitar el almacenamiento de todos sus elementos al representarlas de una manera más simple. Como el caso de las Matrices Dispersas.

Las matrices dispersas son aquellas que poseen un alto porcentaje de sus elementos iguales a cero. Esta característica permite usar estructuras especiales para almacenar únicamente los elementos diferentes de cero para reducir el uso

de memoria y agilizar los cálculos, lo que hace que la información relevante pueda almacenarse reduciendo la cantidad de memoria utilizada[5].

Se pretende analizar este tipo de matrices, sus diferentes formas de almacenamiento y el tratamiento con el fin de encontrar formas para mejorar el tiempo de cómputo requerido y la memoria utilizada. Algunos de los tipos de almacenamiento más conocidos para matrices dispersas son, el almacenamiento por coordenadas y el almacenamiento por filas o columnas comprimidas. En el caso de las matrices tridiagonales o pentadiagonales simplemente se almacenan las diagonales de la matriz en vectores.

La intención es analizar los resultados y el comportamiento de los métodos de factorización existentes como Crout, Cholesky y Dolytle, y de los métodos iterativos de Jacobi y Gauss-Seidel, y a partir de éstos, realizar modificaciones que permitan hacer más eficientes los procesos para el tratamiento de este tipo de problemas.

2. Conceptos Preliminares

A continuación, se presentan algunos conceptos que se consideran fundamentales para el desarrollo del tema tratado en este trabajo.

Para las siguientes definiciones, es importante tener presente la posición en la que se encuentran los elementos de la matriz, es decir,

$$a_{ij} \rightarrow \begin{cases} \text{Si } i = j & \text{el elemento está en la diagonal} \\ \text{Si } i < j & \text{el elemento está sobre la diagonal} \\ \text{Si } i > j & \text{el elemento está debajo de la diagonal} \end{cases}$$

Definición 1 Matriz triangular superior
Una matriz triangular superior de $n \times n$, $U = (u_{ij})$ *tiene, para toda* $j = 1, 2, \ldots, n$ *, los elementos*

$$u_{ij} = \begin{cases} 0 & si\ i > j \\ Cualquier\ valor & i \leq j \end{cases}$$

Un ejemplo de este tipo de matrices es:

$$\mathrm{U} = \begin{bmatrix} 1 & -3 & 7 & 21 & 0 \\ 0 & 3 & 6 & 0 & 2 \\ 0 & 0 & 5 & 12 & 13 \\ 0 & 0 & 0 & 7 & 10 \\ 0 & 0 & 0 & 0 & -11 \end{bmatrix}$$

Nótese que todos los elementos por debajo de la diagonal son ceros y que los elementos que están en la diagonal o por encima de ésta pueden tomar cualquier valor, incluyendo el cero.

Definición 2 Matriz triangular inferior
Una matriz triangular inferior, $L = (l_{ij})$ tiene, para toda $j = 1, 2, \ldots, n$, los elementos [6]

$$l_{ij} = \begin{cases} 0 & si\ i < j \\ Cualquier\ valor & i \geq j \end{cases}$$

Un ejemplo de este tipo de matrices es:

$$\mathrm{L} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 7 & 3 & 0 & 0 & 0 \\ -3 & 4 & 5 & 0 & 0 \\ 5 & 0 & 4 & -7 & 0 \\ 9 & 11 & 0 & 6 & 11 \end{bmatrix}$$

Nótese que todos los elementos por encima de la diagonal son ceros y que los elementos que est[an en la diagonal o por debajo de ésta pueden tomar cualquier valor, incluyendo el cero.

Definición 3 Matriz transpuesta
La transpuesta de una matriz $A = (a_{ij})$ de $n \times m$ es una matriz A^t, donde para cada i, los elementos de la i-ésima columna de A^t son los mismos que los de la i-ésima fila de A, es decir, $A^t = (a_{ji})$.

A continuación se presenta un ejemplo de una matriz y su transpuesta:

$$A = \begin{bmatrix} 1 & 0 & 4 & 13 & 3 \\ 7 & 3 & 11 & 4 & 4 \\ 3 & 4 & 5 & 23 & 15 \\ 5 & 13 & 4 & 7 & 2 \\ 9 & 11 & 14 & 6 & 11 \end{bmatrix} A^T = \begin{bmatrix} 1 & 7 & 3 & 5 & 9 \\ 0 & 3 & 4 & 13 & 11 \\ 4 & 11 & 5 & 4 & 14 \\ 13 & 4 & 23 & 7 & 2 \\ 3 & 4 & 15 & 2 & 11 \end{bmatrix}$$

Nótese que las filas de A son las columnas de A^T.

Definición 4 Matriz estrictamente diagonal dominante
Una matriz A de $n \times n$ es estrictamente diagonal dominante cuando

$$|a_{ii}| > \sum_{j=1, j \neq i}^{n} |a_{ij}|$$

El siguiente es un ejemplo de este tipo de matrices

$$\mathrm{A} = \begin{bmatrix} 30 & 1 & 1 & 2 & 3 \\ 0 & 13 & 4 & 3 & -6 \\ 4 & 3 & -15 & 4 & 0 \\ -13 & 4 & 3 & 27 & 2 \\ 3 & 4 & 1 & 2 & 1l \end{bmatrix}$$

Note que los elementos de la diagonal en valor absoluto, son mayores que la suma de los elementos (en valor absoluto) de la respectiva fila y columna.

Definición 5 Matriz definida positiva
Una matriz A es definida positiva si es simétrica y, $x^tAx > 0$, para todo vector columna n dimensional x [1].

Estas matrices cumplen ciertas características que se presentan en el Teorema 1.

Teorema 1 *Si A es una matriz definida positiva $n \times n$, entonces:*

- *A es no singular*
- $a_{ii} > 0$, *para cada* $i = 1, 2, \ldots, n$
- $max_{1 \leq k,j \leq n} |a_{kj}| \leq max_{1 \leq i \leq n} |a_{ii}|$
- $(a_{ij})^2 < a_{ii} ajj$ *para* $i \neq j$

.

A continuación se presenta un ejemplo de una matriz definida positiva tomado de [1],

$$\mathrm{A} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

Veamos que $x^tAx > 0$. Sea x un vector columna de dimensión 3×1.

$$x^tAx = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Realizando las operaciones y rearreglando los términos obtenemos:

$$\begin{aligned} x^tAx &= 2x_1^2 - 2x_1x_2 + 2x_2^2 - 2x_2x_3 + 2x_3^2 \\ &= x_1^2 + (x_1^2 - 2x_1x_2 + x_2^2) + (x_2^2 - 2x_2x_3 + x_3^2) + x_3^2 \\ &= x_1^2 + (x_1 - x_2)^2 + (x_2 - x_3)^2 + x_3^2 \end{aligned}$$

Como todos los términos están elevados al cuadrado, se tiene que $x^tAx > 0$ en todos los casos, excepto cuando $x_1 = x_2 = x_3 = 0$.

Definición 6 Matriz banda
Una matriz de $n \times n$ recibe el nombre de matriz banda si existen los enteros p y q con $1 < p, q < n$, que tienen la propiedad que $a_{ij} = 0$ siempre que $i + p \leq j$ o $j + q \leq i$. El ancho de banda de este tipo de matrices se define como $w = p + q - 1$.

A continuación se presenta un ejemplo de una matriz de 8×8 con ancho de banda 4, es decir, $p = 2$ y $q = 3$

$$\mathrm{A} = \begin{bmatrix} 5 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 13 & 4 & 3 & 0 & 0 & 0 & 0 \\ 0 & 3 & 15 & 4 & 7 & 0 & 0 & 0 \\ 0 & 0 & 8 & 27 & 2 & 6 & 0 & 0 \\ 0 & 0 & 0 & 2 & 11 & 5 & 12 & 0 \\ 0 & 0 & 0 & 0 & 4 & 12 & 2 & 4 \\ 0 & 0 & 0 & 0 & 0 & 3 & 5 & 21 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 13 \end{bmatrix}$$

En esta matriz, se ve claramente que los elementos por debajo de la primera subdiagonal y por encima de la segunda superdaigonal, son iguales a cero.

Al factorizar este tipo de matrices, se conservan características de su estructura, como enuncia el Teorema 2.

Teorema 2 *Supongamos que $A \in R^{n \times n}$ tiene una factorización LU. Si A tiene un ancho de banda inferior p y un ancho de banda superior q , entonces U es una matriz banda triangular superior con ancho de banda q y L una triangular inferior con ancho de banda p [3].*

Se presenta una matriz banda A y su respectiva factorización en dos matrices, L y U que son triangular inferior y superior respectivamente.

$$\begin{bmatrix} 4 & 1 & 0 & 0 & 0 & 0 \\ 1 & 5 & 1 & 0 & 0 & 0 \\ 1 & 1 & 6 & 1 & 0 & 0 \\ 0 & 1 & 1 & 7 & 1 & 0 \\ 0 & 0 & 1 & 1 & 8 & 1 \end{bmatrix} = \begin{bmatrix} 4{,}0 & 0 & 0 & 0 & 0 \\ 1{,}0 & 4{,}7 & 0 & 0 & 0 \\ 1{,}0 & 0{,}7 & 5{,}8 & 0 & 0 \\ 0 & 1{,}0 & 0{,}7 & 6{,}8 & 0 \\ 0 & 0 & 1{,}0 & 0{,}8 & 7{,}8 \end{bmatrix} \begin{bmatrix} 1{,}0 & 0{,}2 & 0 & 0 & 0 \\ 0 & 1{,}0 & 0{,}2 & 0 & 0 \\ 0 & 0 & 1{,}0 & 0{,}1 & 0 \\ 0 & 0 & 0 & 1{,}0 & 0{,}1 \\ 0 & 0 & 0 & 0 & 1{,}0 \end{bmatrix}$$

Note que la matriz L sólo tiene elementos diferentes de cero en la diagonal y en la subdiagonal y la matriz U, en la diagonal y en dos superdiagonales, conservando así, la estructura de la matriz A.

Definición 7 Matrices dispersas estructuradas y no estructuradas
Las matrices dispersas estructuradas son aquellas que tienen las entradas diferentes de cero formando un patrón regular, permitiendo así, crear un algoritmo que permita determinar donde están sus elementos. Por otro lado, las matrices no estructuras, son las que no presentan ningún patrón definido en sus entradas no nulas [4].

Sistemas de gran dimensión
Se considera que un sistema es de gran dimensión, cuando la dimensión de la matriz asociada al sistema de ecuaciones es muy grande y los costos de computación son elevados [4].

Como los datos obtenidos en los diferentes experimentos realizados, tienen mucho riudo, se utiliza el Filtro de Hodrick y Prescott buscando suavizarlos. A continuación se presenta una descripción de éste.

Filtro de Hodrick y Prescott

Filtro con origen en el método de Ẅhittaker-Henderson de tipo A". La idea de este filtro es extraer el componente tendencial de una serie. Hodrick y Prescott proponen que éste componente es el que minimiza:

$$\sum_{t=1}^{T}(Y_t - \tau_t)^2 + \lambda \sum_{t=2}^{T-1}[(\tau_{t+1} - \tau t) - (\tau_t - \tau_{t-1})]^2 \quad (1)$$

Donde el primer término hace referencia a la suma de las desviaciones de la serie con relación a la tendencia al cuadrado, y mide el grado de ajuste. El segundo término es la suma de cuadrados de las segundas diferencias de los componentes de tendencia y mide el grado de suavidad. Lo que se logra con este modelo es el cambio suave de la tendencia en el tiempo [8].

3. Almacenamiento

Cuando se realizan operaciones con matrices cuyas entradas en algunos casos son ceros, en algunos casos su resultado es predecible, y por esto no tiene sentido almacenarlos. En esta investigación, se quieren conocer los límites de almacenamiento de diferentes estructuras en MATLAB ya que a partir de estos resultados, es posible establecer las ventajas que se obtienen al utilizar métodos que permitan almacenar la información de las matrices dispersas optimizando los recursos computacionales y disminuyendo el consumo de memoria.

3.1. Almacenamiento en MATLAB

Se consideraron matrices, vectores y almacenamiento tipo Sparse en MATLAB (estructura que usa MATLAB para almacenar una matriz dispersa en vectores). En cada uno de estos casos se determinó la dimensión límite posible para almacenarlos y el tiempo que toma crear la matriz según su dimensión. A continuación se describirá con detalle cada uno de estos casos.

3.1.1. Capacidad de Almacenamiento de Matrices Dispersas

Se pretende determinar la máxima capacidad de almacenamiento para almacenar matrices dispersas en un formato especial según la densidad, es decir, la proporción de elementos diferentes de 0 en la matriz con respecto a los que no lo son. En la figura 1 se presentan los resultados.

Los resultados obtenidos en la Figura 1, indican que a mayor densidad, es menor la dimensión máxima de la matriz que puede almacenarse y es mayor el tiempo requerido para crearla. Esto tiene sentido, ya que a menor densidad, menor es la dimensión de los vectores requerida para almacenar los elementos diferentes de cero.

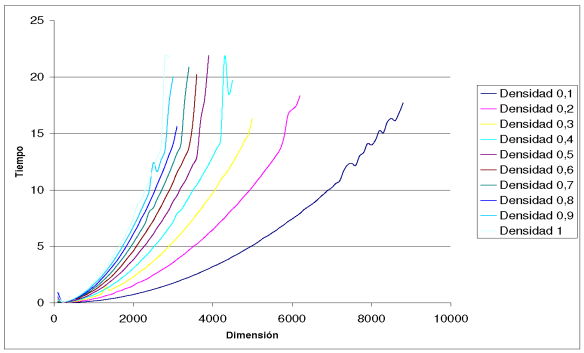

Figura 1: Almacenamiento de matrices dispersas

3.1.2. Capacidad de Almacenamiento de vectores

Para determinar la capacidad de almacenamiento de vectores, se crearon tres tipos diferentes: un vector lleno de ceros, un vector lleno de unos y uno con elementos aleatorios. Los tiempos obtenidos para la creación de cada matriz se presentan en la figura 2.

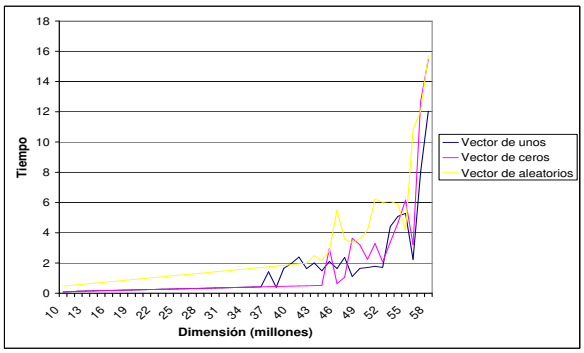

Figura 2: Almacenamiento de vectores

Los resultados presentados en la figura 2 muestran que la dimensión máxima para los tres tipos de vectores es la misma, aproximadamente 58′000,000, pero es claro que la creación de vectores con elementos aleatorios toma más tiempo en la mayoría de los casos.

3.1.3. Capacidad de Almacenamiento de matrices

Para determinar la capacidad de almacenamiento de matrices, se crearon tres tipos diferentes: una matriz llena de ceros, una matriz llena de unos y una

con elementos aleatorios. Los tiempos obtenidos para la creación de cada matriz se presentan en la figura 3

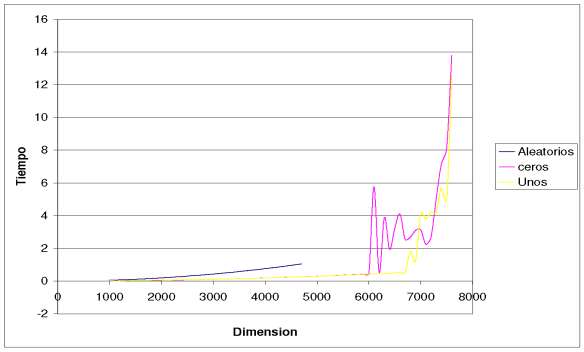

Figura 3: Almacenamiento de matrices

Como se ve en la figura 3, la dimensión máxima para las matrices con elementos aleatorios es de 4700×4700, y el tiempo para crearlas siempre es mayor que el tiempo requerido para las matrices de ceros y unos. Por otro lado, la dimensión máxima para las matrices de ceros y unos es 7600×7600 y el tiempo para crearlas no tiene una tendencia definida entre las dimensiones 6000 y 7500 aproximadamente, cuando lo que se espera, si se tiene un sólo proceso en el computador, es un tiempo con tendencia creciente.

4. Solución de Sistemas de Ecuaciones

Se busca resolver un sistema de ecuaciones de la forma $Ax = b$, donde A es una matriz dispersa, x el vector de la incógnitas y b el vector con los términos independientes. Este sistema de ecuaciones puede resolverse con métodos directos e iterativos, teniendo en cuenta que cuando se aplican métodos directos se presentan errores de redondeo y es posible que se requiera más memoria por la presencia del *fill-in* [7].

4.1. Métodos Directos

Los métodos directos son técnicas que permiten obtener un resultado en un número determinado número de iteraciones [1]. Como se dijo anteriormente, se pueden presentar inconvenientes debido a los errores de redondeo, y en el caso de las matrices dispersas por causa del *fill-in*. Se trabajará con los métodos de Cholesky, Dolytle y Crout. A continuación se dará una breve descripción de estos métodos.

4.1.1. Factorización LU

Se busca factorizar la matriz A como el producto de dos matrices L y U siendo L una matriz triangular inferior y U, triangular superior, es decir, $A = LU$. La diagonal principal de alguna de estas dos matrices consta de unos [1]. Después, buscando solucionar el sistema de ecuaciones $Ax = b$, se resuelve el siguiente problema:

$$Lz = b$$
$$Ux = z$$

Estos dos sistemas pueden resolverse con sustitución progresiva y regresiva respectivamente. Cuando la diagonal principal de la matriz L consta de unos, se conoce como factorización de Dolytle, en el caso en que ocurra esto para la matriz U, se conoce como factorización de Crout [5].

Se presenta un ejemplo de factorización de Crout de un sistema de 6×6,

Sea A la matriz que representa el sistema de ecuaciones y b el vector de términos independientes.

$$\mathrm{A} = \begin{bmatrix} 4 & 1 & 0 & 0 & 0 & 0 \\ 1 & 5 & 1 & 0 & 0 & 0 \\ 0 & 1 & 6 & 1 & 0 & 0 \\ 0 & 0 & 1 & 7 & 1 & 0 \\ 0 & 0 & 0 & 1 & 8 & 1 \\ 0 & 0 & 0 & 0 & 1 & 9 \end{bmatrix}$$

$$\mathrm{b} = \begin{bmatrix} 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{bmatrix}$$

Se obtiene la factorización

$$A = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 1 & 4{,}75 & 0 & 0 & 0 & 0 \\ 0 & 1 & 5{,}78 & 0 & 0 & 0 \\ 0 & 0 & 1 & 6{,}82 & 0 & 0 \\ 0 & 0 & 0 & 1 & 7{,}85 & 0 \\ 0 & 0 & 0 & 0 & 1 & 8{,}87 \end{bmatrix} \times \begin{bmatrix} 1 & 0{,}25 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0{,}31 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0{,}17 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0{,}14 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0{,}12 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Se resuelve el sistema $Lz = b$ de forma progresiva y se obtiene,

$$\mathrm{z} = \begin{bmatrix} 1{,}5 & 1{,}15 & 1{,}18 & 1{,}14 & 1{,}12 & 1{,}11 \end{bmatrix}^T$$

a partir de este resultado, se resuelve el sistema $Ux = z$ de forma regresiva y se obtiene la solución del sistema,

$$\mathrm{x} = \begin{bmatrix} 1{,}26 & 0{,}94 & 1 & 1 & 0{,}98 & 1{,}11 \end{bmatrix}^T$$

4.1.2. Factorización Cholesky

Se busca factorizar la matriz A como el producto de dos matrices L y U siendo L una matriz triangular inferior y U, triangular superior, estableciendo la siguiente condición:

$$\mathrm{L}(i,i) = \mathrm{U}(i,i) \qquad \text{para } i = 1, \ldots, n$$

Después, buscando solucionar el sistema de ecuaciones $Ax = b$, se resuelve el siguiente problema:

$$Lz = b$$
$$Ux = z$$

Estos dos sistemas pueden resolverse con sustitución progresiva y regresiva respectivamente. Las matrices que se deseen factorizar con este método deben ser definidas positivas [3].

Se presenta un ejemplo de este tipo de factorización de un sistema de 5×5,

Sea A la matriz que representa el sistema de ecuaciones y b el vector de términos independientes.

$$\mathrm{A} = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

$$\mathrm{b} = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{bmatrix}$$

Se obtiene la factorización

$$A = \begin{bmatrix} 1{,}41 & 0 & 0 & 0 & 0 \\ -0{,}70 & 1{,}22 & 0 & 0 & 0 \\ 0 & -0{,}81 & 1{,}15 & 0 & 0 \\ 0 & 0 & -0{,}89 & 1{,}11 & 0 \\ 0 & 0 & 0 & -0{,}89 & 1{,}09 \end{bmatrix} \times \begin{bmatrix} 1{,}41 & -0{,}70 & 0 & 0 & 0 \\ 0 & 1{,}22 & -0{,}81 & 0 & 0 \\ 0 & 0 & 1{,}15 & -0{,}89 & 0 \\ 0 & 0 & 0 & 1{,}11 & -0{,}89 \\ 0 & 0 & 0 & 0 & 1{,}09 \end{bmatrix}$$

Se resuelve el sistema $Lz = b$ de forma progresiva y se obtiene,

$$\mathrm{z} = \begin{bmatrix} 3{,}53 & 6{,}94 & 10{,}96 & 15{,}65 & 20{,}99 \end{bmatrix}^T$$

a partir de este resultado, se encuentra la solución del sistema resolviendo $Ux = z$ de forma regresiva y se obtiene

$$\mathrm{x} = \begin{bmatrix} 15{,}83 & 26{,}66 & 31{,}50 & 29{,}33 & 19{,}16 \end{bmatrix}^T$$

4.1.3. Factorización LDM^T y LDL^T

Se busca factorizar la matriz A como el producto de tres matrices LDM^T, siendo L y M matrices triangulares inferiores y D una matriz diagonal con elementos positivos [1]. A partir de esta factorización se resuelve el sistema $Ly = b$ por sustitución progresiva, el sistema $Dz = y$ de forma inmediata (dado que D es una matriz diagonal) y por último $M^T x = z$ por sustitución regresiva. Cuando la estructura de la matriz A es simétrica, se obtiene que $L = M$ y de ahí la factorización LDL^T para matrices simétricas.

4.2. Métodos iterativos

La idea de los métodos iterativos es partir de una solución inicial del sistema de ecuaciones y encontrar una sucesión de soluciones que se busca converjan a la solución del sistema[9]. Estos métodos tienen una gran ventaja con relación a los métodos directos pues no presentan errores de redondeo ni se da el efecto *fill-in*, pero cuando las matrices son mal condicionadas se pueden presentar dificultades para aplicarlos [2]. En este proyecto se trabajará con el método de Jacobi y el método de Gauss Seidel. A continuación se presenta una breve descripción de éstos.

4.2.1. Método de Jabobi

El método de Jacobi es uno de los métodos más sencillos. Está definido para matrices que no tienen ceros en su diagonal. Este método consiste en resolver la i-ésima ecuación en $Ax = b$ buscando obtener

$$x_i = \sum_{j=1, j\neq i}^{n} -(\frac{a_{ij}x_j}{a_{ii}}) + \frac{b_i}{a_{ii}} \tag{2}$$

y poder encontrar cada $x_i^{(k+1)}$ usando la siguiente expresión para cada i desde 1 hasta n [1]:

$$x_i^{(k=1)} = \sum_{j=1, j\neq i}^{n} -(\frac{a_{ij}x_j}{a_{ii}}) + \frac{b_i}{a_{ii}} \tag{3}$$

A continuación se ilustra este método con un ejemplo de un sistema de 4×4 Sea A la matriz que representa el sistema de ecuaciones y b el vector de términos independientes.

$$\mathrm{A} = \begin{bmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{bmatrix}$$

$$\mathrm{b} = \begin{bmatrix} 6 \\ 25 \\ -11 \\ 15 \end{bmatrix}$$

En la Tabla 1 se presentan los resultados de cada iteración y el error respectivo, en este caso, se definió 1000 como el número máximo de iteraciones y 10^{-7} como el error deseado.

Tabla 1: Ejemplo método de Jacobi

iteración	x_1	x_2	x_3	x_4	error
1	0.6000	2.2727	-1.1000	1.8750	3.2017
2	1.0473	1.7159	-0.8052	0.8852	0.8622
3	0.9326	2.0533	-1.0493	1.1309	0.3929
4	1.0152	1.9537	-0.9681	0.9738	0.1375
5	0.9890	2.0114	-1.0103	1.0214	0.0674
6	1.0032	1.9922	-0.9945	0.9944	0.0253
7	0.9981	2.0023	-1.0020	1.0036	0.0120
8	1.0006	1.9987	-0.9990	0.9989	0.0047
9	0.9997	2.0004	-1.0004	1.0006	0.0021
10	1.0001	1.9998	-0.9998	0.9998	0.0009
11	0.9999	2.0001	-1.0001	1.0001	0.0004
12	1.0000	2.0000	-1.0000	1.0000	0.0002
13	1.0000	2.0000	-1.0000	1.0000	0.0001
14	1.0000	2.0000	-1.0000	1.0000	0.0000
15	1.0000	2.0000	-1.0000	1.0000	0.0000
16	1.0000	2.0000	-1.0000	1.0000	0.0000
17	1.0000	2.0000	-1.0000	1.0000	0.0000
18	1.0000	2.0000	-1.0000	1.0000	0.0000
19	1.0000	2.0000	-1.0000	1.0000	0.0000
20	1.0000	2.0000	-1.0000	1.0000	0.0000
21	1.0000	2.0000	-1.0000	1.0000	0.0000

4.2.2. Método de Gauss-Seidel

Cuando se usa el método de Jacobi, en cada iteración, no se utilizan los valores de x que se van encontrando, es decir, para hallar $x_2^{(k+1)}$, se utiliza $x_1^{(k)}$ en lugar de $x_1^{(k+1)}$ que ya se ha calculado. Si se hace este cambio en el algoritmo, buscando siempre utilizar los valores que se han obtenido más recientemente, se obtiene el método de Gauss-Seidel [3]. Aunque en la mayoría de los casos se obtienen mejores soluciones con este método que con el de Jacobi, se debe tener presente que con algunos sistemas de ecuaciones, Jacobi converge y Gauss-Seidel no [1].

Se presenta el mismo ejemplo que se realizó con Jacobi de un sistema de 4×4 para observar las diferencias en los resultados. En la Tabla 2 se presentan los resultados de cada iteración y el error correspondiente.

Tabla 2: Ejemplo método de Gauss-Seidel

iteración	x_1	x_2	x_3	x_4	error
1	0.6000	2.3273	-0.9873	0.8789	2.7429
2	1.0302	2.0369	-1.0145	0.9843	0.0580
3	1.0066	2.0036	-1.0025	0.9984	0.0336
4	1.0009	2.0003	-1.0003	0.9998	0.0049
5	1.0001	2.0000	-1.0000	1.0000	0.0006
6	1.0000	2.0000	-1.0000	1.0000	0.0001
7	1.0000	2.0000	-1.0000	1.0000	0.0000
8	1.0000	2.0000	-1.0000	1.0000	0.0000
9	1.0000	2.0000	-1.0000	1.0000	0.0000

En este caso, con el método de Gauss-Seidel se obtuvo una solución en 9 iteraciones, mientras que con el método de Jacobi, se necesitaron 21.

4.3. Métodos para matrices tridiagonales y pentadiagonales

Dado que las matrices tridiagonales y pentadiagonales tienen una gran cantidad de elementos iguales a cero, es de gran utilidad pensar en métodos que almacenen únicamente los elementos diferentes de cero y así, se pueda lograr el almacenamiento de una mayor cantidad de datos y la reducción del tiempo computacional. Se implementaron los métodos de Dolytle, Crout, Cholesky, Jacobi y Gauss-Seidel aprovechando las propiedades de estas matrices, es decir, almacenando únicamente los elementos diferentes de cero buscando un menor tiempo computacional y una reducción en el uso de la memoria.

En el Algoritmo 1 se presenta el algoritmo de Dolytle para factorizar matrices tridiagonales, las entradas de este algoritmo son un vector a con la diagonal, un vector b con la superdiagonal y un vector c con la subdiagonal. Las salidas son cuatro vectores con la información de las diagonales de U y L, la subdiagonal de L y la superdiagonal de U. El algoritmo de Crout es similar a éste, teniendo en cuenta que en Crout la matriz que presenta unos en la diagonal es U en lugar de L.

Algoritmo 1 Dolytle para matrices tridiagonales

```
function[L1, L2, U1, U2] = Cholesky(a, b, c)
n = length(a);
L1 = ones(1, n);
L2 = zeros(1, n − 1);
U1 = zeros(1, n);
U2 = zeros(1, n − 1);
if a(1) ≠ 0 then
  U2 = b;
  U1(1) = a(1);
  L2(1) = c(1)/U1(1);
  for k = 2 : n do
    U1(k) = a(k) − L2(k − 1) ∗ U2(k − 1);
    if k < n ∧ U1(k) ≠ 0 then
      L2(k) = c(k)/U1(k);
    else if U1(k) == 0 then
      fprintf('División por cero')
    end if
  end for
else
  fprintf('División por cero')
end if
```

Dada la estructura de la factorización de Cholesky, cuando la matriz tridiagonal es simétrica, la factorización obtenida sólo tiene que almacenarse en dos vectores, uno que contenga las diagonales, y otro, la subdiagonal de L y la superdiagonal de U (que son iguales). La estructura de este método se presenta en el Algoritmo 2.

Algoritmo 2 Cholesky para matrices tridiagonales

```
function[L1, L2, U1, U2] = Cholesky(a, b, c)
n = length(a);
Diagonales = zeros(1, n);
Sub = zeros(1, n − 1);
if a(1) > 0 then
    Diagonales(1) = sqrt(a(1));
    Sub(1) = b(1)/Diagonales(1);
    k = 2;
    fin = 0;
    while k <= n ∧ fin == 0 do
        if a(k) − Sub(k − 1)^2 > 0 then
            Diagonales(k) = sqrt(a(k) − Sub(k − 1)^2);
            if Diagonales(k) ≠ 0 ∧ k < n then
                Sub(k) = b(k)/Diagonales(k);
            else if Diagonales(k) == 0 then
                fprintf('División por cero')
                fin = 1;
            end if
        else
            fprintf('División por cero')
            fin = 1;
        end if
        k = k + 1
    end while
    fprintf('Valor negativo en la posicion (1), no se puede ejecutar el método')
end if
```

El trato de matrices pentadiagonales es análogo al de las matrices tridiagonales, agregando dos vectores a la entrada con la información de la nuevas filas que tienen valores diferentes de cero, y la salida del algoritmo, tiene dos vectores más, uno con una subdiagonal más de L, y otro, con una superdiagonal más de U. En el Algoritmo 3 se presenta el método de Crout para factorizar matrices pentadiagonales. El algoritmo de Dolytle para este tipo de matrices es muy similar a éste.

Algoritmo 3 Crout para matrices pentadiagonales

$function[L1, L2, U1, U2] = Crout(a, b, c, d, e)$
$n = length(a)$;
$L1 = zeros(1, n)$;
$L2 = zeros(1, n-1)$;
$L3 = zeros(1, n-2)$;
$U1 = ones(1, n)$;
$U2 = zeros(1, n-1)$;
$U3 = zeros(1, n-2)$;
if $a(1) \neq 0$ **then**
  $L3 = e$;
  $L1(1) = a(1)$;
  $L2(1) = d(1)$;
  $U2(1) = b(1)/L1(1)$;
  $L1(2) = a(2) - (U2(1) * L2(1))$;
  $L2(2) = d(2) - (U2(1) \times L3(1))$
  for $k = 3 : n$ **do**
    $U3(k-2) = c(k-2)/L1(k-2)$;
    $U2(k-1) = (b(k-1) - L2(k-2) \times U3(k-2))/L1(k-1)$;
    $L1(k) = a(k) - L3(k-2) \times U3(k-2) - L2(k-1) \times U2(k-1)$;
    if $k < n$ **then**
      $L2(k) = d(k) - L3(k-1) \times U2(k-1)$;
    end if
  end for
else
  fprintf('División por cero')
end if

Como en el algoritmo de Cholesky se presentan diferencias significativas con el algoritmo de Crout y Dolytle, en el Algoritmo 4 se presenta su estructura.

Algoritmo 4 Cholesky para matrices pentadiagonales

```
function[L1, L2, L3, U1, U2, U3] = Cholesky(a, b, c, d, e)
n = length(a);
L2 = zeros(1, n − 1);
L3 = zeros(1, n − 2);
Diagonales = zeros(1, n);
U2 = zeros(1, n − 1);
U3 = zeros(1, n − 2);
if a(1) ≠ 0 then
  Diagonales(1) = sqrt(a(1));
  U2(1) = b(1)/Diagonales(1);
  U3(1) = c(1)/Diagonales(1);
  L2(1) = d(1)/Diagonales(1);
  Diagonales(2) = sqrt(a(2) − L2(1) × U2(1));
  U2(2) = (b(2) − (L2(1) ∗ U3(1)))/Diagonales(2);
  U3(2) = c(2)/Diagonales(2);
  for k = 3 : n do
    L3(k − 2) = e(k − 2)/Diagonales(k − 2);
    L2(k − 1) = (d(k − 1) − L3(k − 2) × U2(k − 2))/Diagonales(k − 1);
    Diagonales(k) = sqrt(a(k)−L3(k−2)∗U3(k−2)−L2(k−1)∗U2(k−1));
    if k < n − 1 then
      U3(k) = c(k)/Diagonales(k);
    end if
  end for
else
  fprintf('División por cero')
end if
```

Buscando mejorar el tiempo computacional obtenido con los métodos directos, se implementaron los métodos iterativos Jacobi y Gauss-Seidel de la forma tradicional y aprovechando el almacenamiento vectorial para las matrices tridiagonales y pentadiagonales. En el Algoritmo 5 se presenta el método de Jacobi para matrices tridiagonales, este es similar al de Gauss-Seidel, que, como se mencionó anteriormente, difieren en el uso de los valores recientes obtenidos. La entrada de este algoritmo es un vector a con la diagonal de la matriz, los vectores d y e con las superdiagonal y la subdiagonal respectivamente, y un vector b con los términos independientes. La salida es un vector x_f con los valores de las incógnitas, y el número de iteraciones realizadas para alcanzar el error deseado.

Algoritmo 5 Jacobi para matrices tridiagonales

$function[x_f, k] = JacobiTri(a, d, e, b)$
$n = length(a)$;
$x_i = zeros(1, n)$;
$tol = 10^{-7}$;
$x_f = zeros(1, n)$;
$N = 1000$;
$error = tol + 1$;
$k = 1$;
while $K < n \wedge error > tol$ **do**
  for $i = 1 : n$ **do**
    if $i = 1$ **then**
      $x_f(i) = (b(i) - d(i) * x_i(i + 1))/a(i)$;
    else if $i == n$ **then**
      $x_f(i) = (b(i) - e(i - 1) * x_i(i - 1))/a(i)$;
    else
      $x_f(i) = (b(i) - e(i - 1) * x_i(i - 1) - d(i) * x_i(i + 1))/a(i)$;
    end if
    $error = abs(norm(x_f) - norm(x_i))$;
    $x_f = x_i$;
    $k = k + 1$;
  end for
end while

De manera análoga a los algoritmos implementados para resolver matrices pentadiagonales, se implementaron los métodos de Jacobi y Gauss-Seidel, en el Algoritmo 6, se presenta el método de Gauss-Seidel, el cual es similar a la de Jacobi.

Algoritmo 6 Gauss-Seidel para matrices pentadiagonales

$function[x_f, k] = SeidelPenta(a, d1, d2, e1, e2, b)$
$n = length(a)$;
$x_i = zeros(1, n)$;
$tol = 10^{-7}$;
$x_f = zeros(1, n)$;
$N = 1000$;
$error = tol + 1$;
$k = 1$;
while $K < n \wedge error > tol$ **do**
  for $i = 1 : n$ **do**
    if $i = 1$ **then**
      $x_f(i) = (b(i) - d1(i) * x_i(i + 1) - d2(i) * x_i(i + 2))/a(i)$;
    else if $i == 2$ **then**
      $x_f(i) = (b(i) - e1(i-1) * x_f(i-1) - d1(i) * x_i(i+1) - d2(i) * x_i(i+2))/a(i)$;
    else if $i <= n - 1$ **then**
      $x_f(i) = (b(i) - e1(i-1) * x_f(i-1) - e2(i-1) * x_f(i-2) - d1(i) * x_i(i+1) - d2(i) * x_i(i+2))/a(i)$;
    else if $i == n - 1$ **then**
      $x_f(i) = (b(i) - e1(i-1) * x_f(i-1) - e2(i-1) * x_f(i-2) - d1(i) * x_i(i+1))/a(i)$;
    else
      $x_f(i) = (b(i) - e1(i-1) * x_f(i-1) - e2(i-1) * x_f(i-2))/a(i)$;
    else

    end if
    $error = abs(norm(x_f) - norm(x_i))$;
    $x_f = x_i$;
    $k = k + 1$;
  end for
end while

5. Resultados

Debido a la distribución de los procesos en el computador, los resultados obtenidos en los experimentos generan una curva con muchos picos, cuando se esperaba una curva suave, pues a medida que aumenta la dimensión ya sea de un vector o de una matriz, se espera que la ejecución del método tome más tiempo. Por esto, a los resultados obtenidos, se les aplicó el filtro de Hodrick y Prescott, para obtener, a partir de la serie de datos originales, una serie donde sólo se observe la tendencia.

5.1. Métodos almacenamiento matricial

Con el sistema de ecuaciones almacenado en una matriz, se implementaron los métodos Crout, Dolytle, Cholesky, Jacobi y Gauss-Seidel y se realizaron experimentos para determinar el tiempo que tarda en ejecutarse cada uno de estos métodos cuando va creciendo el tamaño del sistema de ecuaciones, es decir, al aumentar el tamaño de la matriz. Los resultados que se obtuvieron se presentan en la figura 4 y en la 5 (donde en el eje x se tiene la dimensión de la matriz y en el eje y el tiempo que tarda en ejecutarse cada método) pues al crecer la dimensión de la matriz, los tiempo obtenidos con los métodos iterativos son mucho menores que los obtenidos con los métodos directos y no pueden compararse en la misma gráfica.

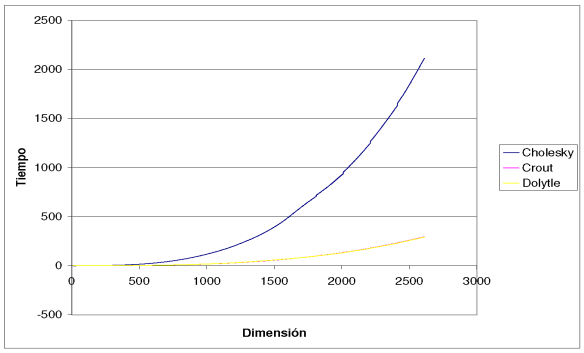

Figura 4: Resultados Métodos directos

Como se puede ver en la figura 4, comparando los métodos directos, con los algoritmos de Crout y Dolytle se obtienen mejores tiempos computacionales que con el de Cholesky, esto es debido al cálculo de raíces presente en este último. También es claro que el tiempo obtenido con los métodos de Crout y Dolytle son muy similares, lo cual se esperaba, ya que el funcionamiento de estos métodos es muy similar.

Como se ve en la figura 5, donde se presentan los resultados obtenidos con los métodos iterativos, se obtienen mejores tiempos con el método de Gauss-Seidel

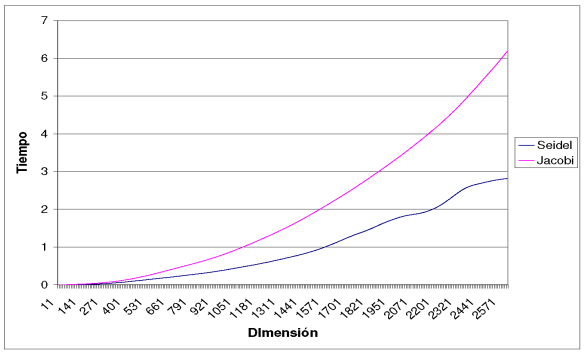

Figura 5: Resultados Métodos iterativos

en todos los casos.

Para evaluar mejor el comportamiento de los métodos cuando la dimensión de la matriz es pequeña (menores de 55×55), se compararon los resultados obtenidos hasta este punto. Los resultados se presentan en la Figura 6.

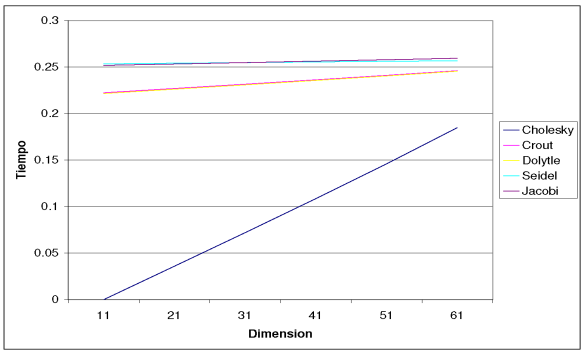

Figura 6: Tiempos de ejecución para matrices pequeñas

Como es claro en la Figura 6, los mejores tiempos se obtienen con el algoritmo de Cholesky, mientras que los algoritmos iterativos presentan el mayor tiempo de ejecución. Estos resultados son contrarios a lo que se obtuvo en los experimentos donde se involucraban matrices de dimensiones mayores. De lo anterior, queda claro que cuando se trabaja con matrices pequeñas, se obtienen soluciones de manera más rápida con los métodos directos.

Para observar con mayor claridad la relación entre los tiempos de cómputo obtenidos, se realizaron comparaciones entre los tiempos de ejecución de Cholesky y Crout, y de Crout y Dolytle para matrices tridiagonales. Los resultados se presentan en la figura 7.

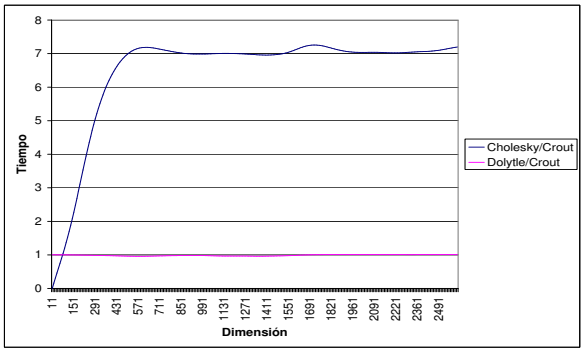

Figura 7: Relación tiempos

En la Figura 7, se ve claramente que la relación entre el tiempo de Crout y el de Dolytle siempre está alrededor de uno, lo cual es coherente con los resultados anteriores, donde se ha visto que estos dos métodos funcionan de forma similar y obtienen soluciones en tiempos muy parecidos. Por otro lado, la relación entre el tiempo de ejecución de Cholesky y Crout cuando la dimensión de la matriz es pequeña, es menor que uno, esto es coherente con los resultados presentados en la Figura 6, pero cuando la dimensión de la matriz empieza a crecer (alrededor de 100×100) esta relación es creciente, es decir, crece más rápido el tiempo de ejecución de Cholesky que el de Crout, finalmente, cuando la dimensión de la matriz es mayor a 500×500 la relación entre los tiempos de ejecución de Cholesky y Crout se estabiliza, lo que indica que la velocidad de crecimiento de ambos métodos se vuelve similar.

Las comparaciones de los tiempos de cómputo entre los métodos directos y el método iterativo Gauss-Seidel, se presentan el la Figura 8.

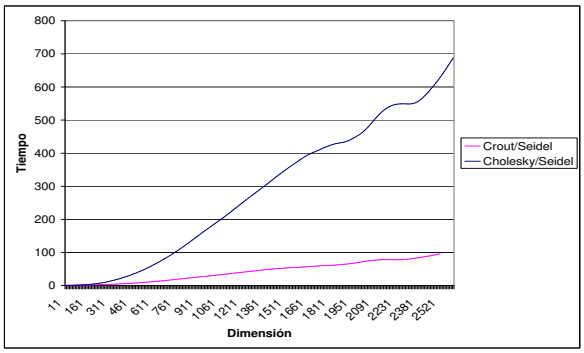

Figura 8: Relación tiempos

Al comparar los tiempos de ejecución entre los métodos directos y Gauss-Seidel, se observa que a medida que crece la dimensión de la matriz, la velocidad de crecimiento de los tiempos de ejecución de los métodos directos es mucho mayor que el tiempo obtenido con el método de Gauss-Seidel.

5.2. Métodos almacenamiento en vectores

Con la misma matriz utilizada en el experimento pasado, pero almacenando los elementos de la matriz en tres vectores y utilizando los algoritmos implementados para este tipo de almacenamiento se obtuvieron los siguientes resultados:

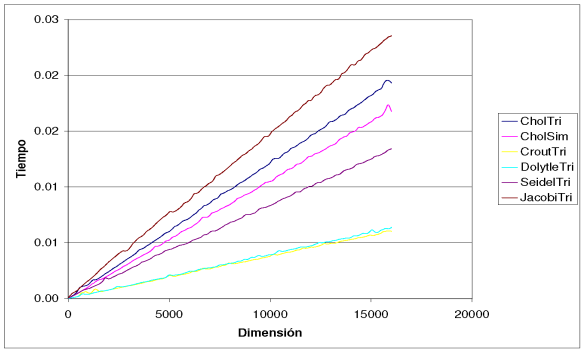

Figura 9: Resultados métodos para matrices tridiagonales

De la figura 9, es claro con los algoritmos para matrices tridiagonales se obtienen mejores tiempos de cómputo con los métodos de Crout y Dolytle que con los métodos iterativos o con Cholesky.

Los resultados obtenidos al ejecutar los métodos para matrices pentadiagonales se presentan en la Figura 10. En éstos, se ve claramente que con el método de Cholesky se obtiene soluciones en tiempos mucho mayores que con los demás métodos utilizados.

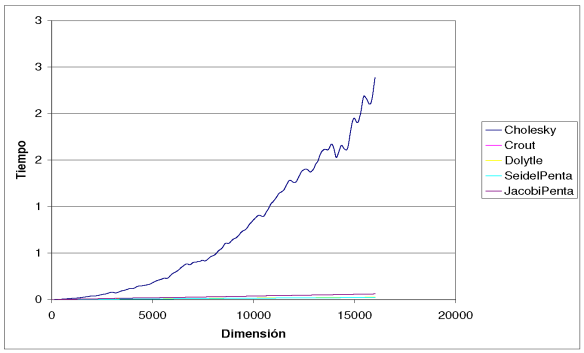

Figura 10: Resultados métodos para matrices pentadiagonales

Para analizar los tiempos que toman los demás métodos, se presentan, en la Figura 11 los resultados de todos los métodos excluyendo los de Cholesky.

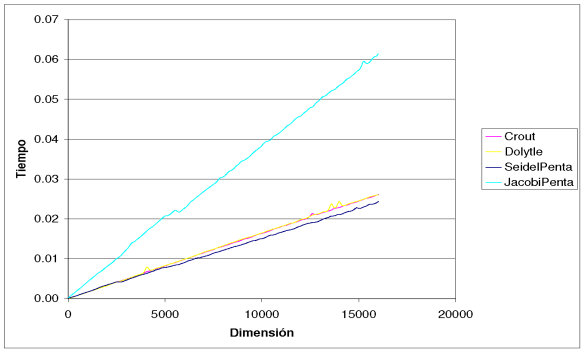

Figura 11: Resultados métodos para matrices pentadiagonales

En el caso de las matrices pentadiagonales almacenadas de forma vectorial, el método que mejores obtiene soluciones en menor tiempo es Gauss-Seidel. Este método también es el más rápido cuando se trabaja con matrices completas. Pero, los métodos de Crout y Dolytle, obtienen mejores tiempos que el método de Jacobi.

5.3. Calidad de las soluciones

Después de evaluar el tiempo que requiere cada método para encontrar una solución, se consideró de gran importancia evaluar la calidad de las soluciones obtenidas, y así, poder determinar cual método debe utilizarse en una situación determinda. Para esto se trabajó con un sistema de 16000×16000 diagonalmente dominante con la siguiente estructura para matrices tridiagonales de $n \times n$

$$\mathrm{A} = \begin{bmatrix} 4 & 1 & 0 & 0 & 0 \\ 1 & 5 & 1 & 0 & 0 \\ 0 & 1 & 6 & \ddots & 0 \\ 0 & 0 & \ddots & \ddots & 1 \\ 0 & 0 & 0 & 1 & n+3 \end{bmatrix}_{n \times n}$$

y con la siguiente, para matrices pentadiagonales.

$$\mathrm{A} = \begin{bmatrix} 4 & 1 & 1 & 0 & 0 \\ 1 & 5 & 1 & 1 & 0 \\ 1 & 1 & 6 & \ddots & \ddots \\ 0 & \ddots & \ddots & \ddots & 1 \\ 0 & 0 & 1 & 1 & n+3 \end{bmatrix}_{n \times n}$$

A partir de éstas, se determinaron los valores del vector de términos independientes para cada uno de los casos tal que la solución real fuera:

$$\mathrm{x} = \begin{bmatrix} 1 & 2 & 3 & \dots & n \end{bmatrix}^T_{1 \times n}$$

Con los resultados obtenidos con cada método, se calculó el error con la ecuación 4

$$\sqrt{\sum_{i=1}^{n} (x_{ci} - x_i)^2} \tag{4}$$

A continuación se presenta un ejemplo para una matriz tridiagonal de 5×5

$$\mathrm{A} = \begin{bmatrix} 4 & 1 & 0 & 0 & 0 \\ 1 & 5 & 1 & 0 & 0 \\ 0 & 1 & 6 & 1 & 0 \\ 0 & 0 & 1 & 7 & 1 \\ 0 & 0 & 0 & 1 & 8 \end{bmatrix}$$

El vector de términos independientes sería,

$$\mathrm{b} = \begin{bmatrix} 6 & 14 & 24 & 36 & 44 \end{bmatrix}^T$$

y la solución del sistema,

$$\mathrm{x} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}^T$$

Los resultados obtenidos utilizando las estructuras presentadas anteriormente se presentan en la Tabla 3

De la Tabla 3, se tiene que con los métodos directos, la calidad de la solución es mejor, ya que en este caso no hay error. Con estos resultados, cuando se trabaje con los métodos para matrices tridiagonales, con Crout y Dolytle es

Tabla 3: Calidad de las soluciones

Método	**Error**
Jacobi Tridiagonales	4.06048E-06
Seidel Tridiagonales	2.32588E-07
Cholesky Tridiagonales	0
Crout Tridiagonales	0
Dolytle Tridiagonales	0
Jacobi Pentadiagonales	1.1798E-05
Seidel Pentadiagonales	1.09487E-06
Cholesky Pentadiagonales	8.27181E-23
Crout Pentadiagonales	0
Dolytle Pentadiagonales	0

posible encontrar mejores soluciones en mejor tiempo. Cuando se trata de matrices pentadiagonales, hay que decidir si se desea un menor tiempo de cómputo o una sólución de mejor calidad, ya que no hay un método que presente las dos características.

Se hizo otro experimento con matrices que no fueran estrictamente diagonal dominantes, con sistemas de 16000×16000 para evaluar los resultados. La estructura utilizada para una matriz tridiagonal de $n \times n$ fue:

$$\mathrm{A} = \begin{bmatrix} 2 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & \ddots & 0 \\ 0 & 0 & \ddots & \ddots & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}_{n \times n}$$

y para matrices pentadiagonales,

$$\mathrm{A} = \begin{bmatrix} 2 & 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & \ddots & 0 \\ 1 & 1 & 2 & \ddots & 1 \\ 0 & \ddots & \ddots & \ddots & 1 \\ 0 & 0 & 1 & 1 & 2 \end{bmatrix}_{n \times n}$$

A partir de éstas, se determinaron los valores del vector de términos independientes para cada uno de los casos tal que la solución real fuera:

$$\mathrm{x} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \end{bmatrix}_{1 \times n}$$

Se presenta un ejemplo para una matriz pentadiagonal de 5×5

$$A = \begin{bmatrix} 2 & 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 1 & 0 \\ 1 & 1 & 2 & 1 & 1 \\ 0 & 1 & 1 & 2 & 1 \\ 0 & 0 & 1 & 1 & 2 \end{bmatrix}$$

El vector de términos independientes sería,

$$b = \begin{bmatrix} 4 & 6 & 6 & 6 & 4 \end{bmatrix}^T$$

y la solución del sistema,

$$x = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T$$

En la Tabla 4 se presentan los resultados obtenidos:

Tabla 4: Calidad de las soluciones

Método	**Error**
Jacobi Tridiagonales	15929.6618
Seidel Tridiagonales	0.000744807
Cholesky Tridiagonales	4.72059E-19
Crout Tridiagonales	1.66237E-17
Dolytle Tridiagonales	4.46392E-17
Jacobi Pentadiagonales	∞
Seidel Pentadiagonales	1.262E+180
Cholesky Pentadiagonales	72934719.95
Crout Pentadiagonales	∞
Dolytle Pentadiagonales	∞

En el caso de los métodos para matrices pentadiagonales, ningún método obtuvo soluciones cercanas a la real. Para los métodos de matrices tridiagonales, con los métodos directos se obtienen mejores soluciones que con los métodos iterativos, lo que confirma el resultado del experimento anterior, cuando se trabaja con métodos para matrices tridiagonales, los métodos de Crout y Dolytle obtienen soluciones más cercanas a la real en tiempos computacionales bajos.

6. Conclusiones y Trabajo Futuro

Es claro que el almacenamiento de matrices y vectores tiene un límite, según los experimentos realizados, con las matrices éste se alcanza, cuando la dimensión es mayor a 7700×7700 y con los vectores, cuando su dimensión es mayor a 58000000. Es por esto, que cuando se trabaja con matrices dispersas, el almacenamiento vectorial trae ventajas tanto en el uso de la memoria, como en el tiempo computacional requerido al solucionar un sistema de ecuaciones.

Comparando los resultados obtenidos entre los métodos en los que se considera la matriz completa y los métodos que usan almacenamiento vectorial, se evidenciaron las ventajas que traen estos últimos en cuanto al tiempo de cómputo requerido para resolver sistemas de ecuaciones y el uso de memoria.

Cuando se usan los métodos que trabajan con la matriz completa, con los métodos iterativos se pueden obtener soluciones en un tiempo mucho menor que con los métodos directos. Cuando la dimensión de la matriz es muy grande, el tiempo requerido para obtener una solución con los métodos directos puede llegar a ser 100 veces mayor que el requerido por los métodos iterativos. Por el contrario, cuando las matrices asociadas al sistema de ecuaciones son pequeñas (una dimensión menor a 50), generalmente se obtienen soluciones en un menor tiempo usando métodos directos.

Se ha encontrado que cuando se trabaja con sistema de ecuaciones que tienen una matriz tridiagonal asociada, los métodos directos de Crout y Dolytle son más eficientes, ya que con estos, se encuentra una solución de mejor calidad en menor tiempo. Por otro lado, cuando se trabaja con matrices pentadiagonales, no se pudo definir un método que obtuviera la mejor solución en el menor tiempo, pues en este caso, con los métodos iterativos se obtienen soluciones en menor tiempo, pero es con los métodos directos que se obtienen soluciones más cercanas a la solución real.

Por los resultados obtenidos en los diferentes experimentos, es claro que el método de Cholesky no presenta ventajas en ningún caso, ya que debido al tiempo que tarda en encontrar una solución, genera costos computacionales muy altos que pueden ser reducidos por otros métodos, como lo indican los experimentos realizados en este trabajo, ya sean directos, como Crout y Dolytle o iterativos como Jacobi y Gauss-Seidel.

Se pretende presentar los resultados de este trabajo en un congreso de computación o de análisis numérico.

A continuación, se plantean con detalle los trabajos futuros propuestos.

6.1. Almacenamiento de matrices dispersas en dos vectores

La idea es explorar el comportamiento de un método de almacenamiento para matrices dispersas en sólo dos vectores, uno que contenga los elementos de la matriz que son diferentes de cero y otro, la posición del elemento, enumerando cada posición en la matriz con un sólo número. La enumeración para una matriz de $n \times n$ sería como se presenta en la *matriz de enumeración M*

$$\mathrm{M} = \begin{bmatrix} 1 & n+1 & 2n+1 & \dots & (n-1)n+1 \\ 2 & n+2 & 2n+2 & \dots & (n-1)n+2 \\ 3 & \vdots & \vdots & \dots & (n-1)n+3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ n & 2n & k & 0 & n^2 \end{bmatrix}$$

Se presenta un ejemplo de una matriz de 5×5.
Sea A la matriz dispersa con la siguiente estructura:

$$\mathrm{A} = \begin{bmatrix} a & 0 & 0 & b & 0 \\ c & 0 & d & 0 & e \\ 0 & f & 0 & 0 & 0 \\ 0 & g & 0 & h & 0 \\ j & 0 & k & 0 & l \end{bmatrix}$$

Los vectores de almacenamiento serían:

$$\text{Elementos} = \begin{bmatrix} a & b & c & d & e & f & g & h & i & j & k & l \end{bmatrix}$$

$$\text{Vector indice} = \begin{bmatrix} 1 & 2 & 5 & 8 & 9 & 12 & 15 & 16 & 19 & 22 & 25 \end{bmatrix}$$

6.2. Tratamiento de matrices dispersas estructuradas y no estructuradas

Después de realizar un estudio sobre algunos casos de matrices dispersas con una estructura definida (matrices tridiagonales y pentadiagonales), se considera de gran importancia, experimentar otros casos de matrices estructuradas, como es el caso de una matriz banda que tenga diagonales internas con todos sus elementos iguales a cero.

Por ejemplo, la matriz A

$$\mathrm{A} = \begin{bmatrix} 1 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 3 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 & 3 & 0 & 0 \\ 0 & 2 & 0 & 1 & 0 & 0 & 3 & 0 \\ 0 & 0 & 2 & 0 & 1 & 0 & 0 & 3 \\ 0 & 0 & 0 & 2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 \end{bmatrix}$$

tiene una factorización LU como sigue:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 12 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1{,}4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0{,}23 & 0{,}11 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1{,}41 & 0{,}05 & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} 1 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 1 & -6 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 1 & -6 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 8{,}54 & -0{,}70 & -4{,}21 & 0{,}35 \\ 0 & 0 & 0 & 0 & 0 & 1{,}40 & -0{,}05 & -0{,}34 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1{,}41 & -0{,}02 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1{,}22 \end{bmatrix}$$

En este caso, hay cambios en la estructura al realizar la factorización y los algoritmos de factorización no podrían implementarse de la misma forma que se hizo para las matrices tridiagonales y pentadiagonales. Por esto, se propone estudiar métodos que permitan conservar la estructura de la matriz en casos como éste y que sean eficientes al aumentar el tamaño de la matriz. Otra estructura que se propone estudiar es una matriz dispersa estructurada que esté conformada por bloques(submatrices no dispersas), por ejemplo:

$$A = \begin{bmatrix} |1 & 2 & 3| & 0 & 0 & 0 & |1 & 3| \\ |6 & 4 & 5| & 0 & 0 & 0 & |2 & 1| \\ |3 & 9 & 10| & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & |8 & 3 & 2| & 0 & 0 \\ 0 & 0 & 0 & |2 & 4 & 7| & 0 & 0 \\ |1 & 3 & 11| & |5 & 11 & 12| & 0 & 0 \\ |2 & 19 & 2| & 0 & 0 & 0 & 0 & 0 \\ |8 & 9 & 15| & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Además de los diferentes tipos de matrices estructuradas, es de gran importancia realizar un estudio de los métodos para tratar matrices dispersas no estructuradas buscando realizar modificaciones pertinentes que permitan obtener resultados de buena calidad y reducir el efecto *fill-in*. En este caso sería interesante considerar matrices dispersas con estructuras especiales, por ejemplo, una matriz en bloques, donde cada uno de éstos tiene una estructura dispersa. A continuación se ilustra este caso:

$$A = \left[\begin{array}{ccc|ccc|cc} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 & 0 & 1 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 8 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ \hline 1 & 3 & 0 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

6.3. Problemas de gran dimensión que involucren matrices dispersas

Cuando se desea resolver un problema de gran dimensión, es decir, que debido a las limitaciones de la memoria, no es posible cargar toda la información para resolverlo, se debe buscar una forma alternativa para resolverlo. Esto es, encontrar la manera de cargar la información por bloques, e ir procesándola. Para esto, podría ser de gran utilidad estudiar la funcion *textscan* de MATLAB, que permite leer un archivo de texto plano por segmentos, procesar esta información y tener un indicador que marca la posición del último dato.

6.4. Fortalezas de la factorización Cholesky en matrices banda simétricas

De los resultados obtenidos en este trabajo, se concluyó que en la mayoría de los casos, con el método de Cholesky, el tiempo de cómputo requerido para obtener una solución es muy alto en comparación con los de los demás métodos. Se propone realizar un estudio con matrices definidas positivas y simétricas, para así, determinar si en estos casos, aprovechando la estructura del algoritmo de Cholesky, logran obtenerse mejores resultados que con otros métodos y lograr resaltar las ventajas que puede tener este método.

7. Apéndice

Se presentan los algoritmos utilizados durante el trabajo y que no se describieron.

Algoritmo 7 Factorización de Crout

```
function[L, U, x, t] = Crout(A, b)
[n  col] = size(a);
L = zeros(n);
U = zeros(n);
if A(1,1) ≠ 0 then
  for i = 1 : n do
    L(i,1) = A(i,1);
    U(1,i) = A(i,1)L(1,1);
  end for
  for k = 2 : n do
    for j = k : n do
      suma1 = 0
      suma2 = 0
      for l = 1 : k − 1 do
        suma1 = suma1 + L(j,l) ∗ U(l,k)
        suma2 = suma2 + L(k,l) ∗ U(l,j)
      end for
    end for
    L(j,k) = A(j,k) − suma1;
    if L(k,k) ≠ 0 then
      U(k,j) = (A(k,j) − suma2)/L(k,k);
    else
      fprintf('División por cero')
    end if
  end for
else
  fprintf('División por cero')
end if
```

Algoritmo 8 Factorización de Dolytle

```
function[L, U, x, t] = Crout(A, b)
[n  col] = size(a);
L = zeros(n);
U = zeros(n);
if A(1,1) ≠ 0 then
  for i = 1 : n do
    U(1,i) = A(1,i);
    L(i,1) = A(i,1)/U(1,1);
  end for
  for k = 2 : n do
    for j = k : n do
      suma1 = 0
      suma2 = 0
      for l = 1 : k − 1 do
        suma1 = suma1 + L(k,l) ∗ U(l,j)
        suma2 = suma2 + L(j,l) ∗ U(l,k)
      end for
    end for
    U(k,j) = A(k,j) − suma1;
    if U(k,k) ≠ 0 then
      U(k,j) = (A(j,k) − suma2)/U(k,k);
    else
      fprintf('División por cero')
    end if
  end for
else
  fprintf('División por cero')
end if
```

Algoritmo 9 Factorización de Cholesky

$function[L, U, x, t] = Crout(A, b)$
$[n \quad col] = size(a);$
$L = zeros(n);$
$U = zeros(n);$
for $i = 1 : n$ **do**
  $L(i, 1) = A(i, 1)/\sqrt{A(1, 1)};$
  $U(1, i) = A(1, i)/\sqrt{A(1, 1)};$
end for
for $k = 2 : n$ **do**
  $j = k;$
  for $j = k : n$ **do**
    $suma1 = 0$
    $suma2 = 0$
    for $l = 1 : k - 1$ **do**
      $suma1 = suma1 + L(k, l) * U(l, j)$
      $suma2 = suma2 + L(j, l) * U(l, k)$
    end for
    if $k == j$ **then**
      $U(k, k) = \sqrt{A(k, k) - suma1};$
      $L(k, k) = U(k, k);$
    else
      $U(k, j) = (A(k, j) - suma1)/U(k, k);$
      $L(j, k) = (A(j, k) - suma2)/L(k, k);$
    end if
  end for
end for

Algoritmo 10 Método de Jacobi

```
function[x, k] = SeidelPenta(A, b)
[n  col] = size(A);
x_i = zeros(1, n);
tol = 10^-7;
N = 1000;
error = tol + 1;
k = 1;
while k < n ∧ error > tol do
  for i = 1 : n do
    suma = 0;
    for j = 1 : n do
      if i ≠ j then
        suma = suma + A(i, j) * x_i(j);
      end if
    end for
    x(i) = (b(i) − suma)/A(i, i);
  end for
  error = abs(norm(x) − norm(x_i));
  x_i = x;
  k = k + 1;
end while
```

Algoritmo 11 Método de Gauss-Seidel

$function[x, k] = SeidelPenta(A, b)$
$[n \quad col] = size(A)$;
$x_i = zeros(1, n)$;
$tol = 10^{-7}$;
$N = 1000$;
$error = tol + 1$;
$k = 1$;
while $k < n \wedge error > tol$ **do**
  for $i = 1 : n$ **do**
    $suma1 = 0$;
    for $j = 1 : i - 1$ **do**
      $suma1 = suma1 + A(i, j) * x(j)$;
    end for
    $suma2 = 0$;
    for j=i+1:n **do**
      $suma2 = suma2 + A(i, j) * x_i(j)$;
    end for
    $suma = suma1 + suma2$;
    $x(i) = (b(i) - suma)/A(i, i)$;
  end for
  $error = abs(norm(x) - norm(x_i))$;
  $x_i = x$;
  $k = k + 1$;
end while

Algoritmo 12 Crout para matrices tridiagonales

```
function[L1, L2, U1, U2] = Cholesky(a, b, c)
n = length(a);
L1 = zeros(1, n);
L2 = zeros(1, n − 1);
U1 = ones(1, n);
U2 = zeros(1, n − 1);
if a(1) ≠ 0 then
  L2 = c;
  L1(1) = a(1);
  U2(1) = c(1)/L1(1);
  for k = 2 : n do
    L1(k) = a(k) − L2(k − 1) ∗ U2(k − 1);
    if k < n ∧ L1(k) ≠ 0 then
      U2(k) = c(k)/L1(k);
    else if L1(k) == 0 then
      fprintf('División por cero')
    end if
  end for
else
  fprintf('División por cero')
end if
```

Algoritmo 13 Dolytle para matrices pentadiagonales

```
function[L1, L2, U1, U2] = Crout(a, b, c, d, e)
n = length(a);
U1 = zeros(1, n);
U2 = zeros(1, n − 1);
U3 = zeros(1, n − 2);
L1 = ones(1, n);
L2 = zeros(1, n − 1);
L3 = zeros(1, n − 2);
if a(1) ≠ 0 then
  U3 = c;
  U1(1) = a(1);
  U2(1) = b(1);
  L2(1) = d(1)/U1(1);
  U1(2) = a(2) − (L2(1) ∗ U2(1));
  U2(2) = b(2) − (L2(1) × U3(1))
  for k = 3 : n do
    L3(k − 2) = e(k − 2)/U1(k − 2);
    L2(k − 1) = (d(k − 1) − U2(k − 2) × L3(k − 2))/U1(k − 1);
    U1(k) = a(k) − U3(k − 2) × L3(k − 2) − U2(k − 1) × L2(k − 1);
    if k < n then
      U2(k) = b(k) − U3(k − 1) × L2(k − 1);
    end if
  end for
else
  fprintf('División por cero')
end if
```

Algoritmo 14 Gauss-Seidel para matrices tridiagonales

$function[x_f, k] = SeidelTri(a, d, e, b)$
$n = length(a)$;
$x_i = zeros(1, n)$;
$x_f = zeros(1, n)$;
$tol = 10^{-7}$;
$N = 1000$;
$error = tol + 1$;
$k = 1$;
while $k < n \wedge error > tol$ **do**
  for $i = 1 : n$ **do**
    if $i = 1$ **then**
      $x_f(i) = (b(i) - d(i) * x_i(i + 1))/a(i)$;
    else if $i == n$ **then**
      $x_f(i) = (b(i) - e(i - 1) * x_f(i - 1))/a(i)$;
    else
      $x_f(i) = (b(i) - e(i - 1) * x_f(i - 1) - d(i) * x_i(i + 1))/a(i)$;
    end if
    $error = abs(norm(x_f) - norm(x_i))$;
    $x_i = x_f$;
    $k = k + 1$;
  end for
end while

Algoritmo 15 Jacobi para matrices pentadiagonales

$function[Resultado, k] = JacobiPenta(a, d1, d2, e1, e2, b)$
$n = length(a);$
$x_i = zeros(1, n);$
$tol = 10^{-7};$
$x_f = zeros(1, n);$
$N = 1000;$
$error = tol + 1;$
$k = 1;$
while $K < n \wedge error > tol$ **do**
  for $i = 1 : n$ **do**
    if $i = 1$ **then**
      $x_f(i) = (b(i) - d1(i) * x_i(i + 1) - d2(i) * x_i(i + 2))/a(i);$
    else if $i == 2$ **then**
      $x_f(i) = (b(i) - e1(i-1) * x_i(i-1) - d1(i) * x_i(i+1) - d2(i) * x_i(i+2))/a(i);$
    else if $i <= n - 1$ **then**
      $x_f(i) = (b(i) - e1(i-1) * x_i(i-1) - e2(i-1) * x_i(i-2) - d1(i) * x_i(i+1) - d2(i) * x_i(i+2))/a(i);$
    else if $i == n - 1$ **then**
      $x_f(i) = (b(i) - e1(i-1) * x_i(i-1) - e2(i-1) * x_i(i-2) - d1(i) * x_i(i+1))/a(i);$
    else
      $x_f(i) = (b(i) - e1(i-1) * x_i(i-1) - e2(i-1) * x_i(i-2))/a(i);$
    else

    end if
    $error = abs(norm(x_f) - norm(x_i));$
    $x_i = x_f;$
    $k = k + 1;$
  end for
end while

Referencias

[1] R. L. Burden and J. D. Faires. *Análisis numérico*. Séptima edición edition, 2002.

[2] P. Ezzatti. Mejora del desempeño de modelos numéricos del río de la plata. Master's thesis, Universidad de la República Montevideo, Uruguay, 2006.

[3] G. H. Golub and C. F. V. Loan. *Matrix Computations*. third edition, 1996.

[4] J. D. Jaramillo, A. V. Macía, and F. C. Zabala. *Métodos directos para la solución de sistemas de ecuaciones lineales simétricos, indefinidos, dispersos y de gran dimensión*. EAFIT, 2006.

[5] D. Kinkaid and W. Cheney. *Numerical Analysis: mathemathics of scientific computing*. 1991.

[6] J. H. Mathews and K. D. Fink. *Métodos Numéricos con MATLAB*. 200.

[7] G. Montero, R. Montenegro, J. M. Escobar, and E. Rodríguez. Resolución de sistemas de ecuaciones tipo *SPARSE*:la estrategia rpk. 2003.

[8] E. Muñoz and A. C. Kikut. El filtro de hodrick y prescott: una técnica para la extracción de la tendencia de una serie. Marzo 1994.

[9] F. R. Villatoro, C. M. García, and J. I. Ramos. Métodos iterativos para la resolución de ecuaciones algebraicas lineales. 2001.