Andrés Felipe Parada Valle (aparadav@unal.edu.co)
Sebastián Hernández Cerón (sehernandezce@unal.edu.co)
Andrés Nicolás Figueredo Urrea (afigueredou@unal.edu.co)
Christian David Rodríguez Castiblanco (chdrodriguezca@unal.edu.co)

Universidad Nacional de Colombia – Sede Bogotá
Mayo 25 de 2017 – Bogotá D.C.

INTRODUCCIÓN

El presente documento pretende presentar al lector el proceso de construcción de un código escrito en el lenguaje C++ cuyo objetivo es utilizar el proceso de factorización LU para solucionar un sistema de ecuaciones ingresado por el usuario o generado por el computador. La estructura del presente documento consiste de tres partes, la primera siendo un marco teórico para poner al lector en contexto con las temáticas aquí trabajadas, la segunda es un listado de funciones creadas por los autores del documento con el objetivo de desarrollar de manera más sencilla el código final, y la tercera es la presentación del código final con sus respectivos comentarios.

OBJETIVOS

El objetivo principal de este proyecto es el de escribir un código en C++ que permita al usuario de una manera sencilla y comprensible realizar el cálculo de la solución de un sistema de ecuaciones con “n” número de ecuaciones y “n” números de incógnitas usando como una herramienta el proceso de factorización LU.

Como objetivos secundarios se tiene la descomposición del código en funciones que hagan tareas específicas, esto con aras a mejorar la comprensión del código y tener más orden en su estructura. Aparte de esto, se pretende realizar un trabajo de consulta y aprendizaje acerca del proceso de factorización LU, haciendo énfasis en los tres tipos de factorización, los cuales son Crout, Doolittle y Cholesky. Por último, se quiere aplicar las nociones de estructura y sintaxis del lenguaje C++ adquiridas a lo largo del periodo académico en un código que más allá de tener un uso pedagógico, pueda tener un viable uso académico.

1. MARCO TEÓRICO:

VECTOR: Un vector es una n-túpla de n objetos, los cuales son las componentes del mismo, en este caso numéricos; dependiendo del ordenamiento de los mismos, tendremos un vector fila o un vector columna.
VECTOR FILA: El vector fila, se ordena de manera horizontal como su nombre lo indica en forma de fila así:

$$[n_{11} \quad n_{12} \quad n_{13} \quad \dots \quad n_{1n}]$$

VECTOR COLUMNA: Vector que se ordena de manera vertical formando una columna de la siguiente manera:

$$\begin{bmatrix} n_{11} \\ n_{21} \\ \vdots \\ n_{n1} \end{bmatrix}$$

MATRIZ: Es una composición entre vectores filas y vectores columna formando un arreglo de tipo rectangular. Una matriz poseerá n filas y m columnas, así que el tamaño de una matriz será dado por la expresión n x m. Un esquema de matriz es el siguiente:

$$\begin{bmatrix} n_{11} & n_{12} & \cdots & n_{13} \\ n_{21} & n_{22} & & n_{23} \\ \vdots & & \ddots & \vdots \\ n_{n1} & n_{n2} & \cdots & n_{nn} \end{bmatrix}$$

La ubicación dentro de una matriz se da por coordenadas, siendo el primer subíndice la fila y el segundo la columna, por ejemplo, la componente n_{32} está ubicada en la fila 3, en la columna 2.

En notación, las filas estarán dadas por la letra i y las columnas por la letra j así: x_{ij}

MATRIZ CUADRADA: Una matriz cuadrada es una matriz, la cual tiene igual número de filas como de columnas, en la que su diagonal estará dada por los elementos n_{ii}

$$\begin{bmatrix} n_{11} & n_{12} & \cdots & n_{13} \\ n_{21} & n_{22} & & n_{23} \\ \vdots & & \ddots & \vdots \\ n_{n1} & n_{n2} & \cdots & n_{nn} \end{bmatrix}$$

Dentro de las matrices cuadradas, encontramos unos tipos específicos, entre los cuales están:

MATRIZ TRIANGULAR SUPERIOR: En este tipo de matriz, los elementos que estén debajo de la diagonal serán ceros.

$$\begin{bmatrix} 1 & 7 & -2 \\ 0 & -3 & 4 \\ 0 & 0 & 2 \end{bmatrix}$$

MATRIZ TRIANGULAR INFERIOR: Es similar a la triangular superior, pero en este caso los ceros serán los elementos que se encuentren por encima de la diagonal principal.

$$\begin{bmatrix} 1 & 0 & 0 \\ 7 & -3 & 0 \\ -2 & 5 & 2 \end{bmatrix}$$

MATRIZ DIAGONAL: En este tipo de matriz, los únicos elementos que no serán 0 serán las componentes ubicadas en la diagonal de la matriz.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

MATRIZ IDENTIDAD: Es una matriz de tipo diagonal, pero los valores de su diagonal serán todos igual a 1.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

PRODUCTO ENTRE MATRICES: El producto entre matrices, será sumar los productos entre las componentes de la fila $_i$ de la primera matriz, por las componentes de la columna $_j$ de la segunda.

$$A \cdot B = \begin{pmatrix} 2 & 0 & 1 \\ 3 & 0 & 0 \\ 5 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix} =$$

$$= \begin{pmatrix} 2\cdot1+0\cdot1+1\cdot1 & 2\cdot0+0\cdot2+1\cdot1 & 2\cdot1+0\cdot1+1\cdot0 \\ 3\cdot1+0\cdot1+0\cdot1 & 3\cdot0+0\cdot2+0\cdot1 & 3\cdot1+0\cdot1+0\cdot0 \\ 5\cdot1+1\cdot1+1\cdot1 & 5\cdot0+1\cdot2+1\cdot1 & 5\cdot1+1\cdot1+1\cdot0 \end{pmatrix} =$$

$$= \begin{pmatrix} 3 & 1 & 2 \\ 3 & 0 & 3 \\ 7 & 3 & 6 \end{pmatrix}$$

Figura 0: Producto entre matrices[1]

MÉTODO DE ELIMINACIÓN GAUSSIANA: Este sistema sirve para resolver sistemas de ecuaciones y funciona de la siguiente manera:

$$\begin{cases} x + 2y + 3z = 1 \\ 4x + 5y + 6z = -2 \\ 7x + 8y + 10z = 5 \end{cases}$$

El primer paso es convertir el sistema de ecuaciones en forma de matriz, para lo cual tomaremos el índice de cada una de las variables. Es decir, generar una matriz aumentada.

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & -2 \\ 7 & 8 & 10 & 5 \end{array}\right)$$

En este caso generaremos la matriz A con los índices de las variables, y la matriz B con los términos independientes.

El siguiente paso será volver los elementos debajo de la componente de la diagonal en valores nulos en este caso, volver el 4 y el 7 en 0.

Para realizar esto, haremos un procedimiento en la fila en la que se encuentre el valor que queremos eliminar, en este caso, en la fila 2 y 3.

[1] Tomado de: http://www.vitutor.com/algebra/matrices/producto.html

Para la fila 2, lo que haremos será multiplicar el inverso aditivo (-4) del valor que queremos eliminar (4), por el valor correspondiente a su columna en la fila 1 y lo sumamos al valor que tiene en la fila 2 de la siguiente manera:

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & -2 \\ 7 & 8 & 10 & 5 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 0 & -3 & -6 & -6 \\ 0 & -6 & -11 & -2 \end{array}\right)$$

Continuamos escalonando la matriz de tal forma que formemos el elemento nulo que falta bajo la diagonal; para esto, reescribiremos la fila 3 de la siguiente manera:

El objetivo es anular -6, por lo cual lo que se realizará será multiplicar el pívot de su columna (-3) por un número que anule el -6 (-2) y esto se realiza en toda la fila 3.

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 0 & -3 & -6 & -6 \\ 0 & -6 & -11 & -2 \end{array}\right) * -2 \sim \left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 0 & -3 & -6 & -6 \\ 0 & 0 & 1 & 10 \end{array}\right)$$

Posteriormente, se volverá ese pívot un 1, para esto, se dividirá por sí mismo a toda la fila 2, obteniendo lo que se verá a continuación.

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 0 & 1 & 2 & 2 \\ 0 & 0 & 1 & 10 \end{array}\right)$$

Por lo tanto, nuestro sistema de ecuaciones tendrá la siguiente forma:

$$\begin{cases} x + 2y + 3z = 1 \\ 0 + y + 2z = 2 \\ 0 + 0 + z = 10 \end{cases}$$

De la tercera fila tenemos que z=10, por lo tanto, z tendrá el valor de 10.

En la segunda fila, podemos reemplazar el valor que hayamos de z y nos queda Y+(2*10) =2, realizamos el despeje y nos queda que y=-18.

En la tercera fila realizamos el mismo proceso, evaluando y como -18 y z como 10, con lo cual tendremos que x+2(-18) +3(10) =1 Y al realizar el despeje tendremos que x=7

OPERACIONES ELEMENTALES ENTRE FILAS: Entre filas en matrices podemos realizar cambio de ellas, donde podemos invertir una fila con otra, así:

$$\begin{bmatrix} 4 & 5 & 8 \\ 6 & 7 & -10 \\ -1 & 2 & 3 \end{bmatrix} \sim \begin{bmatrix} 6 & 7 & -10 \\ 4 & 5 & 8 \\ -1 & 2 & 3 \end{bmatrix}$$

En la cual se invierten la fila 1 y 2.

También encontramos la multiplicación de una fila por una constante, en la cual la constante que se aplique afectará a todas las componentes de la fila.

En el siguiente ejemplo se multiplica toda la f_2 por 2

$$\begin{bmatrix} 4 & 5 & 8 \\ 6 & 7 & -10 \\ -1 & 2 & 3 \end{bmatrix} * 2 \sim \begin{bmatrix} 6 & 7 & -10 \\ 12 & 14 & -20 \\ -1 & 2 & 3 \end{bmatrix}$$

Existe otro tipo de operación, el cual es la suma, en este caso podemos sumar una fila a otra.

FACTORIZACIÓN LU:

La factorización LU, deriva sus nombres de L= Lower y U= Upper, y es un método de factorización de una matriz en la cual interviene un producto entre dos matrices, las cuales serán una triangular inferior y una triangular superior. Para poderla realizar, los elementos de la diagonal deben ser diferentes de 0, en caso de no ser así, se pre-multiplicará por permutación tantas veces sea necesario.

Es un método en el cual podemos aplicar el sistema de eliminación gaussiano en una matriz.

Sus utilidades más grandes, son que, en términos de memoria, esta tiene menos requerimientos, y en términos de efectividad, resolver esta matriz cuadrada pasa a ser la resolución entre los dos sistemas triangulares L y U respectivamente.

I. MÉTODO DE CHOLESKY:

En el caso en el que una matriz M cumpla las siguientes condiciones:

- Sea A cuadrada.
- Sea A simétrica.
- Sea A definida positiva.

Entonces es posible aplicar a A un método de factorización LU especial, donde la matriz triangular superior U es la transpuesta de la matriz triangular inferior L, es decir:

$$A = L \cdot U = L \cdot Lt$$

Antes de presentar formalmente el método, se dará una breve reseña histórica del mismo. La factorización Choelsky toma el nombre del matemático André Louis-Choelsky, quien encontró que una matriz simétrica definida positiva puede ser descompuesta como el producto de una matriz triangular inferior y la transpuesta de la matriz triangular inferior.[2] El método de Choelsky, como los métodos de Doolitle y Crout, es utilizado para resolver sistemas de ecuaciones lineales, con la diferencia de que cuando el método es aplicable, es dos veces más eficiente que los otros dos métodos mencionados anteriormente, debido a que solo es necesario calcular una matriz.

[2] Es.wikipedia.org. (2017). Factorización de Cholesky. [en línea] Disponible en: https://es.wikipedia.org/wiki/Factorización_de_Cholesky [Revisado 8 mayo 2017].

$$L.L^t = \begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix} \cdot \begin{pmatrix} l_{11} & l_{21} & l_{31} & l_{41} \\ 0 & l_{22} & l_{32} & l_{42} \\ 0 & 0 & l_{33} & l_{43} \\ 0 & 0 & 0 & l_{44} \end{pmatrix} == \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Figura 1. Factorización LU: Método Cholesky[3]

Para determinar el método que se debe utilizar al momento de determinar la matriz L primero se consideran los elementos de sus diagonales, en la figura 1 se puede observar que los elementos de la diagonal de A, que son de la forma *ann,* corresponden a los resultados de la n-ésima fila de la matriz L por la n-ésima columna de la matriz Lt, quedando los elementos de la diagonal de la matriz A en la figura 1 de la siguiente manera:

$$l_{11}^2 = a_{11}$$
$$l_{21}^2 + l_{22}^2 = a_{22}$$
$$l_{31}^2 + l_{32}^2 + l_{33}^2 = a_{33}$$
$$l_{41}^2 + l_{42}^2 + l_{43}^2 + l_{44}^2 = a_{44}$$

Se puede observar que los elementos de la diagonal de A siguen la siguiente forma:

$$l_{n1}^2 + l_{n2}^2 + \ldots + l_{n\,n-1}^2 + l_{nn}^2 = a_{nn} \quad (2)$$

Sin embargo, lo que nos interesa es encontrar los elementos de L, así que se procede a despejar el elemento l_{nn}de la ecuación (2):

$$l_{n1}^2 + l_{n2}^2 + \ldots + l_{n\,n-1}^2 + l_{nn}^2 = a_{nn}$$
$$l_{nn}^2 = a_{nn} - l_{n1}^2 - l_{n2}^2 - \ldots - l_{n\,n-1}^2$$
$$l_{nn}^2 = a_{nn} - (l_{n1}^2 + l_{n2}^2 + \ldots + l_{n\,n-1}^2)$$
$$l_{nn}^2 = a_{nn} - \left(\sum_{j=1}^{n-1} l_{nj}^2\right)$$
$$l_{nn} = \sqrt{a_{nn} - \left(\sum_{j=1}^{n-1} l_{nj}^2\right)} \quad (3)$$

Se tiene así, que la ecuación (3) es válida para calcular los elementos de la diagonal de la matriz L.

Ahora se procede a considerar los elementos por fuera de la diagonal. En el punto anterior se partió de la idea de deducir una expresión para los elementos de la matriz A en términos de la matriz L y luego despejar el término de la matriz L deseado. Como la matriz A es simétrica, los elementos de A de la forma a_{ij} son iguales a los elementos de la forma a_{ji}, por lo que solo es necesario analizar o el triángulo superior de la matriz, o el inferior. Se procederá a analizar a triangulo inferior.

Las expresiones para los elementos del triángulo inferior de la matriz son:

[3] Imagen tomada de: http://www.ehu.eus/juancarlos.gorostizaga/mn11b/temas/factorizac_LU.pdf

$$a_{21} = l_{21} \cdot l_{11}$$
$$a_{31} = l_{31} \cdot l_{11} \qquad a_{32} = l_{31} \cdot l_{21} + l_{32} \cdot l_{22}$$
$$a_{41} = l_{41} \cdot l_{11} \qquad a_{42} = l_{31} \cdot l_{21} + l_{32} \cdot l_{22} \qquad a_{43} = l_{41} \cdot l_{31} + l_{42} \cdot l_{32} + l_{43} \cdot l_{33}$$

En las expresiones anteriores se puede observar una tendencia en las columnas, para ser más específicos, una tendencia en cuanto al número de factores en la ecuación. Concretamente se puede ver que los elementos de la primera columna tienen un factor, los de la segunda tienen dos y los de la tercera tienen tres. Esto se debe a la forma como se multiplican las matrices, en la figura 1 se puede ver que para encontrar el último elemento del triángulo inferior de la matriz A se debe realizar el producto punto entre la última fila de la matriz L, y la penúltima columna de la matriz Lt, pero como esta última columna tiene un cero en la posición l_{34}, la expresión para el último elemento del triángulo inferior de la matriz A tiene n-1 factores, siendo n el número de filas y columnas de la matriz cuadrada.

$$L.L^t = \begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix} \cdot \begin{pmatrix} l_{11} & l_{21} & l_{31} & l_{41} \\ 0 & l_{22} & l_{32} & l_{42} \\ 0 & 0 & l_{33} & l_{43} \\ 0 & 0 & 0 & l_{44} \end{pmatrix} == \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Figura 2. El producto punto entre la fila y columna subrayadas tiene n-1 factores, esto debido al 0 en la columna que cancela el n-ésimo factor.[4]

En las expresiones para los elementos a_{ij} de la matriz se puede observar otro hecho importante: todas están en términos de elementos L que no se pueden calcular con lo planteado hasta ahora, a excepción del término l_{11}, el cual se puede hallar con la ecuación (3), esto es de gran utilidad, ya que con este término es posible calcular las entradas de la primera columna de la matriz L, con estas entradas es posible calcular el término l_{22}y este a la vez facilita los cálculos de la segunda columna:

$$l_{21} = \frac{a_{21}}{l_{11}}$$
$$l_{31} = \frac{a_{31}}{l_{11}} \qquad l_{32} = \frac{a_{32} - (l_{31} \cdot l_{21})}{l_{22}}$$
$$l_{41} = \frac{a_{41}}{l_{11}} \qquad l_{42} = \frac{a_{42} - (l_{41} \cdot l_{21})}{l_{22}} \qquad l_{43} = \frac{a_{43} - (l_{41} \cdot l_{31} + l_{42} \cdot l_{32})}{l_{33}}$$

La expresión que se encuentra entre paréntesis presenta una relación interesante con el término a calcular, se observa que en cada factor $(l_{ik} \cdot l_{jk})$ de la expresión dentro de los paréntesis, la posición fila (i) del primer término y la posición columna (j) del segundo término corresponde a la posición fila-columna (ij) del término que se quiere calcular, mientras que la posición faltante de cada término corresponde a un número igual para los dos términos del factor y diferente en cada uno de los factores. Se puede también observar que los elementos de la primera columna no tienen factores asociados, los de la segunda tienen uno y los de la tercera tienen dos, entonces, el número faltante en discusión se le puede atribuir al número del factor en la expresión, siendo el 1 para el primer factor, el 2 para el segundo factor y el n para el n-ésimo factor. Por último, se puede ver que las

[4] Imagen tomada de: http://www.ehu.eus/juancarlos.gorostizaga/mn11b/temas/factorizac_LU.pdf

expresiones tienen $j-1$ factores. Toda la información anterior se puede sintetizar en la siguiente fórmula:

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot l_{jk}}{l_{jj}} \quad (4)$$

Ahora ya se tienen las expresiones para hallar los términos de la matriz L, y ya que la matriz U es igual a Lt, sólo son necesarias las expresiones (3) y (4) para realizar una factorización LU de una matriz por el método Cholesky, cuando esto sea posible.

II. MÉTODO DE CROUT:

Sea A una matriz cuadrada (n x n), entonces tenemos A= LU en donde la matriz L es una matriz triangular inferior (n x n) y U una matriz triangular superior con diagonal unitaria (n x n).

A = L * U

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ l_{31} & l_{32} & l_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn} \end{bmatrix} * \begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & 1 & u_{23} & \cdots & u_{2n} \\ 0 & 0 & 1 & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

Figura 3. Factorización LU: Método de Crout[5]

Cuando se indique $a_{ij} = x$, significa que en el renglón i y columna j se encuentra el valor x. Antes de determinar el método general que nos permita encontrar los valores de la matriz L y U, observemos que en la figura 1:

- Los elementos de la matriz L en la primera columna son iguales a los elementos de la matriz A.

$$\begin{aligned} l_{11} * 1 &= a_{11} \\ l_{21} * 1 &= a_{21} \\ l_{31} * 1 &= a_{31} \\ &\cdots \\ l_{n1} * 1 &= a_{n1} \end{aligned}$$

  Despejando tenemos que $l_{i1} = \boldsymbol{a_{i1}}$.

- Los elementos en la columna j=2 de la matriz A se obtiene de la siguiente forma:

$$\begin{aligned} l_{11} * u_{12} &= a_{12} \\ l_{21} * u_{12} + l_{22} &= a_{22} \\ l_{31} * u_{12} + l_{32} &= a_{32} \\ &\cdots \\ l_{n1} * u_{12} + l_{n2} &= a_{n2} \end{aligned}$$

[3] imagen tomada de: http://www.monografias.com/trabajos92/factorizacion-matrices/factorizacion-matrices.shtml

A partir de esto podemos encontrar el valor en l_{i2} y en u_{12}:

Sabemos que $l_{i1} = a_{i1}$ y despejando tenemos que:

$$u_{12} = \frac{a_{12}}{l_{11}} \qquad l_{i2} = a_{i2} - l_{i1} * u_{12}$$

Con el proceso anterior y la figura 1 se puede deducir que $u_{1j} = \frac{a_{1j}}{l_{11}}$ debido a que, en la matriz L, los elementos siguientes a l_{11} son 0.

Ahora es posible encontrar los valores de la fila i=1 de la matriz U
y los valores de la columna j=2 de la matriz L.

- Los elementos de la columna n de la Matriz A se obtienen de la siguiente forma:

$$l_{11} * u_{2n} = a_{1n}$$
$$l_{21} * u_{1n} + l_{22} * u_{2n} = a_{2n}$$
$$l_{31} * u_{1n} + l_{32} * u_{2n} + l_{33} * u_{3n} = a_{3n}$$
$$\dots$$
$$l_{n1} * u_{1n} + l_{n2} * u_{2n} + l_{n3} * u_{3n} + \dots + l_{nn} = a_{nn}$$

Para determinar $l_{nn} = a_{nn} - (l_{n1} * u_{1n} + l_{n2} * u_{2n} + l_{n3} * u_{3n} + \dots + l_{n(n-1)} * u_{(n-1)n})$ debemos conocer primero el valor de l_{n2}, l_{n3},..., $l_{n(n-1)}$ y u_{2n},u_{3n},..,$u_{(n-1)n}$, para ello se debe realizar el proceso descrito para obtener los elementos en la columna j=2 de la matriz A, luego el mismo proceso pero para j=3, se despejará la l_{i3} y u_{23}, lo que nos permitirá conocer los elementos de la Matriz L en la columna 3 y los elementos de la matriz U en el renglón 2:

Veamos qué sucede:

$$l_{11} * u_{13} = a_{13}$$
$$l_{21} * u_{13} + l_{22} * u_{23} = a_{23}$$
$$l_{31} * u_{13} + l_{32} * u_{23} + l_{33} = a_{33}$$
$$\dots$$
$$l_{n1} * u_{13} + l_{n2} * u_{23} + l_{n3} = a_{n3}$$

Despejando tenemos:

$$u_{23} = \frac{a_{23} - l_{21} * u_{13}}{l_{22}} \qquad l_{i3} = a_{n3} - (l_{i1} * u_{13} + l_{i2} * u_{23})$$

Con el proceso anterior y la figura 1 se puede deducir que $u_{2j} = \frac{a_{2j} - l_{21} * u_{1j}}{l_{22}}$ debido a que, en la matriz L, los elementos siguientes a l_{22} son iguales a 0.

Claramente podemos saberl_{i1},l_{i2},u_{13}, ya que sabemos que:

$$l_{i1} = a_{i1} \qquad u_{1j} = \frac{a_{1j}}{l_{11}} \qquad l_{i2} = a_{i2} - l_{i1} * u_{12}$$

Observe que para hallar el valor u_{ij}y l_{ij}se requiere haber hallado los valores de la columna de la matriz L y los valores del renglón de la matriz U anterior donde se halla el valor que buscamos.
Por ello se debe repetir este proceso hasta encontrar el valor del_{n2}, l_{n3},...,$l_{n(n-1)}$ y u_{2n},u_{3n},..,$u_{(n-1)n}$, pero claro, esto no es tan complicado si la matriz A es 3x3, pero qué ocurriría si la matriz A es de mayor tamaño. Aquí terminan las observaciones de la figura 1.

Teniendo en cuenta las observaciones anteriores es posible deducir lo siguiente:

- Los valores de l_{ij} serán iguales a $a_{ij} - (l_{i1} * u_{1j} + l_{i2} * u_{2j} + l_{i3} * u_{3j} + \ldots + l_{i(j-1)} * u_{(j-1)j})$
- Los valores de u_{ji} serán iguales a $\frac{a_{ji} - (l_{j1} * u_{1i} + l_{j2} * u_{2i} + \ldots + l_{k(j-1)} * u_{(j-1)i})}{l_{jj}}$

Es decir,

$$1 \leq j < n$$

$$l(i,j) = a_{ij} - \sum_{k=1}^{j-1} l_{ik} * u_{kj} \; ; \quad j \leq i \leq n$$

$$u(j,i) = \left(a_{ji} - \sum_{k=1}^{j-1} l_{jk} * u_{ki} \right) / \, l_{jj} \; ; \quad j+1 \leq i \leq n$$

Figura 4. Fórmulas generales de factorización LU método Crout.[6]

III. MÉTODO DOLITTLE:

Sea A una matriz cuadrada (n x n), se asume que la descomposición A=LU existe, en donde la matriz L es una matriz triangular inferior con diagonal unitaria (n x n) y U una matriz triangular superior (n x n).

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ m_{21} & 1 & 0 \\ m_{31} & m_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} = LU$$

Si multiplicamos L y U por la primera fila de A inmediatamente obtenemos que:

$$a_{11} = u_{11} \quad , \quad a_{12} = u_{12} \quad , \quad a_{13} = u_{13}$$

4. Imagen tomada de: http://www.monografias.com/trabajos92/factorizacion-matrices/factorizacion-matrices.shtml

Luego multiplicamos L y U por la segunda fila de A y tenemos que:

$$a_{21} = m_{21}u_{11} \quad , \quad a_{22} = m_{21}u_{12} + u_{22} \quad , \quad a_{23} = m_{21}u_{13} + u_{23}$$

A partir de estas ecuaciones, podemos resolver para m_{21}, u_{22} y u_{23}. Por último, multiplicamos L y U por la tercera fila de A y tenemos que:

$$a_{31} = m_{31}u_{11} \quad , \quad a_{32} = m_{31}u_{12} + m_{32}u_{22} \quad , \quad a_{33} = m_{31}u_{13} + m_{32}u_{23} + u_{33}$$

Para un ejemplo menos general, supongamos que queremos encontrar una factorización LU a la matriz A

$$A = \begin{bmatrix} 2 & 3 & 2 \\ 1 & 3 & 2 \\ 3 & 4 & 1 \end{bmatrix}$$

En primer lugar, suponemos que existe una factorización LU. Entonces tenemos:

$$A = \begin{bmatrix} 2 & 3 & 2 \\ 1 & 3 & 2 \\ 3 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ m_{21} & 1 & 0 \\ m_{31} & m_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} = LU$$

Deduciendo los valores correspondientes a la primera fila obtenemos:

$$2 = u_{11} \quad , \quad 3 = u_{12} \quad , \quad 2 = u_{13}$$

Situándonos en la segunda fila de A obtenemos que:

$$1 = m_{21}(2) \quad , \quad 3 = m_{21}(3) + u_{22} \quad , \quad 2 = m_{21}(2) + u_{23}$$

De la primera ecuación vemos que $m_{21} = \frac{1}{2}$. Conectando esto a la segunda ecuación tenemos que $u_{22} = \frac{3}{2}$. Conectando m_{21} en la tercera ecuación tenemos que $u_{23} = 1$.

Pasando a la tercera fila de A y tenemos que:

$$3 = m_{31}(2) \quad , \quad 4 = m_{31}(3) + m_{32}\left(\frac{3}{2}\right) \quad , \quad 1 = m_{31}(2) + m_{32}(1) + u_{33}$$

De la primera ecuación vemos que $m_{31} = \frac{3}{2}$ .Conectando esto a la segunda ecuación tenemos que $m_{32} = -\frac{1}{3}$ Conectando ambos en la tercera ecuación tenemos que $u_{33} = -\frac{5}{3}$. De esta manera tenemos todas las entradas para L y para U obteniendo la expresión:

$$A = \begin{bmatrix} 2 & 3 & 2 \\ 1 & 3 & 2 \\ 3 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{3}{2} & -\frac{1}{3} & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 & 2 \\ 0 & \frac{3}{2} & 1 \\ 0 & 0 & -\frac{5}{3} \end{bmatrix} = LU$$

2. CODIGO

VERSIONES DE CÓDIGO

Sabemos que la factorización LU es un tema matemático con ecuaciones ya definidas, por ello primero se planteó la teoría con el fin de definir en que consiste cada tema usado, luego a partir de ejercicios hechos a mano se observó cómo se podría implementar en C++ una función que permita resolver sistema lineal a partir de la factorización LU. Por lo anterior no se poseen distintos tipos de versiones de código del programa.

Ahora, se explicarán la mayoría de funciones usadas en el programa final.

FUNCIONES PRINCIPALES

Estas funciones permiten crear, llenar e imprimir un vector o matriz.

Función Crear_Matriz_Manual:
Esta función es tipo “double **”, con el fin de retornar matrices. Los argumentos que requiere son entradas de números enteros (*int Tamaño*). Permite crear una matriz de tamaño (*int Tamaño*) de forma dinámica a partir de punteros, y llenar la matriz con valores determinados por el usuario. El código de la función es el siguiente:

```
double **Crear_Matriz_Manual (int Tamaño)
{
  // 1. Se define la nueva matriz.
  double **Matriz = new double*[Tamaño];
  for (int i=0; i<Tamaño;i++)
  {
     Matriz[i]=new double[Tamaño];
  }

  // 2. Se le pide al usuario que llene la matriz.
  for (int i=0; i<Tamaño;i++)
  {
     for (int j=0;j<Tamaño;j++)
     {
        cout<<"Ingrese posición ["<<i<<"]["<<j<<"]:";
        cin>>*(*(Matriz+i)+j); //Matriz [i][j]

     }
  }
  return Matriz;
}
```

A continuación, se explica cada numeral del código:

1. Declaramos una matriz de entradas reales, a partir de punteros, de tamaño ingresado por el usuario.
2. A partir de un *for* anidado recorremos la matriz desde la posición [0][0] permitiéndole al usuario digitar los valores que tendrá la matriz.

Función Crear_Matriz_Ale:

Esta función es tipo “double **”, con el fin de retornar matrices, similar a la función anterior, permite crear una matriz de tamaño (*int Tamaño*) de forma dinámica a partir de

punteros, pero en esta ocasión se llenará la matriz a partir de números aleatorios generados por una semilla. Los argumentos que requiere son entradas de números enteros (*int Tamaño*). El código de la función es el siguiente:

```
double **Crear_Matriz_Ale (int Tamaño){

  // 1. Se define la nueva matriz.
  double **Matriz;
  Matriz= new double*[Tamaño];
  for (int i=0;i<Tamaño;i++)
  {
     Matriz[i]=new double[Tamaño];
  }

  // 2. Se llena la matriz de números aleatorios del 0 al 30.
  for (int i=0;i<Tamaño;i++)
  {
     for (int j=0;j<Tamaño;j++)
     {
        *(*(Matriz+i)+j)=rand()%31;
     }
  }
  return Matriz;
}
```

A continuación, se explica cada numeral del código:

1. Declaramos una matriz de entradas reales, a partir de punteros, de tamaño ingresado por el usuario.
2. A partir de un *for* anidado recorremos la matriz desde la posición [0][0] llenando la matriz con valores dados por la semilla *srand(time(NULL));* declarada en el *main().* Además, se ha definido un rango 0≤ x <31, a partir de una función módulo -> "rand%31", es decir, dentro de la matriz solo se guardarán valores que estén dentro del rango.

Función Crear_Vector_Manual:

Esta función es tipo "double *", con el fin de retornar vectores, permite crear un vector de tamaño (*int Tamaño*) de forma dinámica a partir de punteros. Llenará el vector con valores dados por el usuario. El argumento que requiere es un número entero (*int Tamaño*). El código de la función es el siguiente:

```
double *Crear_Vector_Manual (int Tamaño){
  //1. Define el nuevo vector.
  double *Vector = new double[Tamaño];

  //2. Le pide al usuario que llene el vector.
  for (int i=0; i<Tamaño; i++){
     cout<<"Ingrese la posición ["<<i<<"]: ";
     cin>>Vector[i];
  }

  return Vector;
}
```

A continuación, se explica cada numeral del código:

1. Declaramos un vector de entradas reales, a partir de punteros, con tamaño igual al valor de la variable ingresada en el parámetro.
2. A partir de un *for* cambiamos de posición en el vector, permitiéndole al usuario digitar el valor que tendrá el vector en cada posición.

Función Crear_Vector_Ale:

Esta función es tipo "double *", con el fin de retornar vectores, similar a la anterior permite crear un vector de tamaño (*int Tamaño*) de forma dinámica a partir de punteros. Llenará el vector con valores aleatorios. El argumento que requiere es un número entero (*int Tamaño*). El código de la función es el siguiente:

```
double *Crear_Vector_Ale (int Tamaño){
  //1. Define el nuevo vector.
  double *Vector = new double[Tamaño];

  //1. Llena el vector de números aleatorios del 0 hasta el 30.
  for (int i=0; i<Tamaño; i++){
     Vector[i]=rand()%31;
  }

  return Vector;
}
```

A continuación, se explica cada numeral del código:

1. Define el vector de tamaño, dado por el usuario.
2. Recorre el vector usando un *for.* En cada posición guardará un valor que esté dentro del intervalo $0 \leq x < 11$. La computadora a partir de una semilla con una función aleatoria dará un valor que se será dividido entre 11 y el residuo de esta división será el que se tomará.

Función Imprimir_Matriz:

Esta función es tipo "*void*" puesto que no es necesario retornar ningún dato. Los argumentos que requiere es una matriz con entradas reales *(double **Matriz*) y el tamaño de dicha matriz (*int Tamaño*). Permite mostrar en pantalla los dígitos guardados en la matriz. El código de la función es el siguiente:

```
void Imprimir_Matriz(double **Matriz, int Tamaño)
{
  //1. Imprime la matriz.
  for (int i=0;i<Tamaño;i++)
  {
     for (int j=0;j<Tamaño;j++)
     {
        cout<<'\t';
        cout<<*(*(Matriz+i)+j);//Matriz[i][j]
        cout<<'\t';
     }
     cout<<endl;
  }
}
```

A continuación, se explica cada numeral del código:

1. A partir de un *for* anidado recorremos la matriz. Con el segundo *for*, se desplaza la matriz de columna en columna imprimiendo los valores de la fila dejando un espacio entre términos, luego con el primer *for* se cambia de fila dando un salto de línea.

Función Imprimir_Vector:

Esta función es tipo "*void*" puesto que no es necesario retornar ningún dato. Los argumentos que requiere son un vector de entradas reales *(double **Vector*) y el tamaño de dicho vector (*int Tamaño*). Permite mostrar en pantalla los dígitos guardados en el vector. El código de la función es el siguiente:

```
void Imprimir_Vector (double *Vector, int Tamaño)
{
  // 1. Imprime el vector.
  cout<<endl;
  for (int i=0; i<Tamaño; i++){
     cout<<Vector[i]<<endl;
  }
}
```

A continuación, se explica cada numeral del código:

1. Recorre el vector e imprime cada dato que se haya guardado en este.

FUNCIONES FACTORIZACION LU

Función Doolittle

La función toma como parámetros la matriz de coeficientes del sistema (*double **Matriz*), el tamaño de la matriz (*int Tamano*), el arreglo en donde se almacena la matriz L (*double **AuxL*) y el arreglo en donde se almacena la matriz U (*double **AuxU*). El propósito de la función es el de almacenar en las matrices auxiliares la factorización LU de la matriz de coeficientes.

```
void Doolittle(double **Matriz, int Tamano,double **AuxL, double **AuxU){//Factorización de A=LU donde L posee la diagonal de unos

   int k=0;
   float suma=0;
   for(int i=0; i<Tamano; i++){
   // 3. Llena las matrices. Llena una fila de la matriz U y luego una columna de la matriz L.
      k=0,suma=0;
      while (k<Tamano){//Llena las filas de la matriz U
         if(k<i){// Pone los ceros, da la forma de triangular Superior
            AuxU[i][k]=0;
         }
         else{
            for(int j=0; j<i; j++){// 3.1 Multiplica la fila de la matriz L con la columna de la matriz U.
               suma=suma+AuxL[i][j]*AuxU[j][k];
            }
            AuxU[i][k]=Matriz[i][k]-suma;//3.2 Toma el resultado de la multiplicación y le resta a matriz A.
         }
         k++;
         suma=0;
      }
      k=0,suma=0;
      while (k<Tamano){//Llena las columnas de la matriz L.
         if(k<i){// Pone los ceros, da la forma de triangular Inferior
            AuxL[k][i]=0;
         }
         else if (k==i){// Pone los unos de la diagonal.
            AuxL[k][k]=1;
         }
         else{
            for(int j=0; j<i; j++){//3.3  Multiplica la fila de la matriz L con la columna de la matriz U.
               suma=suma+AuxL[k][j]*AuxU[j][i];
            }
            AuxL[k][i]=(Matriz[k][i]-suma)/AuxU[i][i];//3.4 Toma el resultado de la multiplicación y le resta a matriz A, luego divide
         }                                    // entre una posición dada de la matriz U.
         k++;
         suma=0;
      }

   }

}
```

A continuación, se explica cada numeral del código:

1. Se declara la matriz L, definida en el tamaño de A para proceder con la factorización.

2. Se declara la matriz U, definida en el tamaño de A para proceder con la factorización.

3. El comando *for* define los valores la primera fila de la matriz triangular superior U y la primera columna de la matriz triangular inferior L tomados de la matriz A inicial.

3.1 Toma los valores obtenidos de la primera columna de L y la primera fila de U y los multiplica para obtener el valor que posteriormente se le restará a matriz A.

3.2 Toma el valor de la multiplicación anterior y se la resta a la matriz A inicial. Generando los valores correspondientes para cada fila bajo la condición de que tenga el tamaño adecuado, y definiendo la diagonal de unos (números 1). También definiendo al tiempo los ceros (0) que irán por debajo de la diagonal.

3.3 Multiplica la fila de la matriz L con la columna de la matriz U si la condición de tamaño no se cumple para la matriz.

3.4 Toma el resultado de la multiplicación y le resta a matriz A, luego divide entre una posición dada en la posición U

Función Crout:
La función toma como parámetros la matriz de coeficientes del sistema (*double **Matriz*), el tamaño de la matriz (*int Tamano*), el arreglo en donde se almacena la matriz L (*double **AuxL*) y el arreglo en donde se almacena la matriz U (*double **AuxU*). El propósito de la función es el de almacenar en las matrices auxiliares la factorización LU de la matriz de coeficientes.

```
void Crout (double **Matriz, int Tamano, double **AuxL, double **AuxU){
   int k=0;
   float suma=0;
   for(int i=0; i<Tamano; i++){// 3. Llena las matrices. Llena una columna de la matriz L y luego una fila de la matriz U.
      k=0,suma=0;
      while (k<Tamano){
         for(int j=0; j<i; j++){// 3.1 Multiplica la fila de la matriz L con la columna de la matriz U.
            suma=suma+AuxL[k][j]*AuxU[j][i];
         }
         AuxL[k][i]=Matriz[k][i]-suma;//3.2 Toma el resultado de la multiplicación y le resta a matriz A.
         k++;
         suma=0;
      }//Llena las columnas de la matriz L.
      k=0,suma=0;
      while (k<Tamano){//3.3  Multiplica la fila de la matriz L con la columna de la matriz U.
         for(int j=0; j<i; j++){
            suma=suma+AuxL[i][j]*AuxU[j][k];//3.4 Toma el resultado de la multiplicación y le resta a matriz A, luego divide
            // entre una posición dada de la matriz L.
         }
         AuxU[i][k]=(Matriz[i][k]-suma)/AuxL[i][i];
         k++;
         suma=0;
      }//Llena las filas de la matriz U.
   }
}
```

A continuación, se explica cada numeral del código:

1. Se declara la matriz L, definida en el tamaño de A para proceder con la factorización.

2. Se declara la matriz U, definida en el tamaño de A para proceder con la factorización.

3. El comando *for* define los valores la primera fila de la matriz triangular superior U y la primera columna de la matriz triangular inferior L tomados de la matriz A inicial.

3.1 Toma los valores obtenidos de la primera columna de L y la primera fila de U y los multiplica para obtener el valor que posteriormente se le restará a matriz A.

3.2 Toma el valor de la multiplicación anterior y se la resta a la matriz A inicial. Generando los valores correspondientes para cada fila bajo la condición de que tenga el tamaño adecuado, y definiendo la diagonal de unos (números 1). También definiendo al tiempo los ceros (0) que irán por debajo de la diagonal.

3.3 Multiplica la fila de la matriz L con la columna de la matriz U si la condición de tamaño no se cumple para la matriz.

3.4 Toma el resultado de la multiplicación y le resta a matriz A, luego divide entre una posición dada en la posición U

Función Cholesky:
La función es de retorno tipo matriz con entradas *double*, los argumentos que requiere son una matriz con entradas reales *(double **Matriz*) y el tamaño de dicha matriz (*int Tamaño*). La función, en términos muy generales, toma la matriz del argumento, le realiza el proceso de factorización Cholesky y devuelve la matriz *L* obtenida.

La función es la siguiente:

```
double **Cholesky (double **Matriz, int Tamaño){
  //1. Se crea la matriz L y se inicializa con entradas aleatorias.
  double **matrizl=Crear_Matriz_Ale(Tamaño);

  //2. Se hace un recorrido primero por columnas y luego filas.
  for (int j=0; j<Tamaño; j++){
     for(int i=0; i<Tamaño; i++){

        //3. Se plantea el proceso para los elementos en la diagonal principal.
        if (i==j){
           int Contador1=0;
           for(int k=0;k<i;k++)
           {
              Contador1+=pow(matrizl[i][k], 2);
           }
           matrizl[i][j]=sqrt(Matriz[i][j]-Contador1);
           Contador1=0;
        }

        //4. Se plantea el proceso para los elementos en la diagonal principal.
        else if (i>j){
           double Contador2=0;
           for (int k=0; k<j; k++){
              Contador2+=(matrizl[i][k]*matrizl[j][k]);
           }
           matrizl[i][j] = Matriz[i][j] - Contador2;
           matrizl[i][j]/= matrizl[j][j];
           Contador2=0;
        }

        //5. Se llena el triángulo superior de la matriz de ceros.
        else{
           matrizl[i][j]=0;
        }
     }
  }

  //6. La función retorna la matriz L.
  return matrizl;
}
```

A continuación, se explica cada numeral del código:

4. Se declara la matriz *L* y luego se llama a la función *Crear_Matriz_Ale* para llenarla de números aleatorios.

5. Se va recorriendo la matriz revisando primero las columnas y luego las filas. En cada posición en la que se paran los *for* se calcula la entrada de la posición análoga en la matriz *L.*

6. Este *if* evalúa el caso en el que el elemento de la matriz ingresada pertenece a la diagonal principal. Se declara la variable *Contador1* y se crea el ciclo *for* con el propósito de realizar la sumatoria presente en la ecuación (3). La variable *Contador1* se crea con el objetivo de almacenar la suma acumulada por el *for,* y por último se escribe la expresión restante de la ecuación (3).

7. Este *else if* evalúa el caso en el que el elemento se encuentra en el triángulo inferior de la matriz ingresada, Se declara la variable *Contador2*, cuya función es análoga a la de *Contador1*, se crea el ciclo *for* para realizar la sumatoria, y por último se completa la expresión de la ecuación (4).

8. Como la matriz *L* fue inicializada con números aleatorios, es necesario asegurarse que los elementos del triángulo superior sean cero.

9. Se retorna la matriz *L*.

Al momento de calcular la matriz *L* se tiene un problema de aproximación, ya que las últimas entradas de la matriz *L* tienen una discrepancia con su valor real de al rededor 0.96,

el cual es un valor que no permite que al momento de multiplicar a L por su traspuesta se obtenga como resultado la matriz de la cual se partió.

FUNCIONES PARA RESOLVER SISTEMAS LINEALES

Función Sustitucion_Regresiva:

Esta función es tipo “double *”, con el fin de retornar vectores. Los argumentos que requiere son entradas de números enteros (*int Tamaño*), entrada real tipo matriz y vector. Permite realizar la sustitución regresiva a partir de una matriz triangular superior, ->Sea U una matriz triangular superior de tamaño (nxn) y “x” y “y” sean vectores de tamaño (n), Ux=y. El código de la función es el siguiente:

```
double *Sustitucion_Regresiva (double **Matriz, double *Vector, int Tamaño)
{
  // 1. Define el vector solución de retorno de la función.
  double *Vector_X = new double[Tamaño];

  //2. Recorre la matriz por filas de abajo para arriba.
  for (int j=Tamaño-1; j>-1; j--){

     //2.1. Paso base de la sustitución.
     if (j==Tamaño-1){
        Vector_X[j] = (Vector[j]/Matriz[j][j]);
     }

     //2.2. Paso general de la sustitución.
     else{
        int suma=0;
        //3. Realiza la suma de los productos coeficientes-incógnita de las incógnitas ya calculadas.
        for ( int k=Tamaño-1; k>j; k--){
           suma+= Matriz[j][k]*Vector_X[k];
        }
        //6. Resta al termino independiente la suma obtenida y la divide por el coeficiente correspondiente.
        Vector_X[j] = Vector[j] - suma;
        Vector_X[j]/= Matriz[j][j];
     }

  }

  return Vector_X;
}
```

A continuación, se explica cada numeral del código:

1. Declaramos el vector donde se guardará los valores encontrado, de tamaño igual al *número* de filas de la matriz
2. Recorre la matriz despejando la variable respectiva, y guardando los valores en el Vector_X. A partir de un *for* se cambia de fila en la matriz U y en el vector Y, iniciando en la última posición de esta matriz y vector.
3. A partir de *if* comprobamos si es el primer despeje. Siendo esto verdad, despejamos la variable. Para ello se reconoce que U_{nn} $x_n=y_n$, despejando obtenemos $x_n=y_n/ U_{nn}$.
4. Ahora, si el *if* resulta falso, a partir de un *for* resolvemos la parte de la ecuación con la variable que ya conocemos. Lo anterior, nos permite quedar tan solo con una variable. Luego, despejamos la variable que nos queda, pasamos a restar el valor que nos dio con el anterior *for* y luego dividimos similar a lo que se hizo en el paso anterior y se guarda este valor en el vector X, en su respectiva posición.

Función Sustitucion_Progresiva:

Esta función es tipo “double *”, con el fin de retornar vectores. Los argumentos que requiere son entradas de números enteros (*int Tamaño*), entrada real tipo matriz y vector. Permite realizar la sustitución regresiva a partir de una matriz triangular inferior ->Sea L una matriz triangular inferior de tamaño (nxn) y “y” y “b” sean vectores de tamaño (n), Ly=b. El código de la función es el siguiente:

```
double *Sustitucion_Progresiva (double **Matriz, double *Vector, int Tamaño)
{
  // 1. Define el vector solución de retorno de la función.
  double *Vector_Y = new double[Tamaño];

  //2. Recorre la matriz por filas de arriba para abajo.
  for (int j=0; j<Tamaño; j++){

     //3. Paso base de la sustitución.
     if (j==0){
        Vector_Y[j] = (Vector[j]/Matriz[j][j]);
     }

     //4. Paso general de la sustitución.
     else{
        int suma=0;
        for ( int k=0; k<j; k++){
           //5. Realiza la suma de los productos coeficientes-incógnita de las incógnitas ya calculadas.
           suma+= Matriz[j][k]*Vector_Y[k];
        }

        //6. Resta al termino independiente la suma obtenida y la divide por el coeficiente correspondiente.
        Vector_Y[j] = Vector[j] - suma;
        Vector_Y[j]/= Matriz[j][j];
     }

  }

  return Vector_Y;
}
```

A continuación, se explica cada numeral del código:

1. Declaramos el vector donde se guardará los valores encontrado, de tamaño igual al *número* de filas de la matriz.
2. Recorre la matriz despejando la variable respectiva, y guardando los valores en el Vector_Y. A partir de un *for* se cambia de fila en la matriz U y en el vector Y, iniciando en la posición inicial de esta matriz y vector.
3. A partir de *if* comprobamos si es el primer despeje. Siendo esto verdad, despejamos la variable. Para ello se reconoce que U_{00} $x_0=y_{0,}$ despejando obtenemos que $x_0=y_0/U_{00.}$
4. Ahora, si el *if* resulta falso, a partir de un *for* resolvemos la parte de la ecuación con las variables que ya conocemos. Lo anterior, nos permite quedar tan solo con una variable. Luego, despejamos la variable que nos queda, para esto pasamos a restar el valor que nos dio con el anterior *for* y luego dividimos similar a lo que se hizo en el paso anterior y se guarda este valor en el vector X, en su respectiva posición.

FUNCIONES ADICIONALES

Función Multiplicacion_Matriz:

Esta función es tipo “double **”, con el fin de retornar matrices .Esta función realiza la multiplicación de matrices cuadradas siempre y cuando sea posible. Sea D y C matrices de tamaño nxn para que sea posible multiplicar. Los argumentos que requiere son entradas de números enteros (*int ColumnasC*),(*int FilasD*) y otras de números reales tipo matriz (*double ** matrizC*),(*double ** matrizD*).
El código de la función es el siguiente:

```
double **Multiplicaion_Matriz(double ** matrizC, double **matrizD, int ColumnasC, int FilasD){

  //  1.  Verifica que sea posible multiplicar.
  if(ColumnasC==FilasD){
  double aux=0;
 //  2. Crea una matriz del tamaño de la matriz C.
   double **matrizM = Crear_Matriz_Ale(ColumnasC);
```

```
//  Multiplicación  de matrices: matriz C* matriz D = matriz M.
      for (int i=0; i<ColumnasC; i++){


         for(int j=0;j<ColumnasC; j++){
           // 3.  Realiza la multiplicación de la fila de la matriz C por la columna de la matriz D
              for (int a=0; a<ColumnasC; a++){
                aux=aux+matrizC[i][a]*matrizD[a][j];
                                                  }
             // 4.  Llena la matriz M.
             matrizM[i][j]=aux;
              aux=0;
          }
        }
     }
     else
    {
       cout<<"No es posible realizar la multiplicación"<<endl;
       return 0;
    }
    // 5. Retorna la Matriz M..
   return matrizM;
}
```

A continuación se explica cada numeral del código:

1. A partir del *if* se realiza la comparación entre el número de columnas de la matriz C y el número de filas de la matriz D, si poseen el mismo valor será posible multiplicar y proseguirá a realizar la multiplicación. Pero, si al realizar la comparación poseen diferente valor mostrará el mensaje “No es posible realizar la multiplicación.” y terminará la función.
2. Se llama la función Crear_Matriz_Ale con el fin de crear una matriz de tamaño de la Matriz C, la cual se guardará en la Matriz M. La matriz M será la matriz donde se guarde el resultado de multiplicación de matrices.
3. A partir, de un *for* anidado se recorre la matriz C y D realizando la correspondiente multiplicación.
    a. Con el primer for se cambia la posición de la fila de la matriz C y con el segundo se cambia la posición de la columna de la matriz D.
    b. Debido a los anteriores for nos hemos posicionado en una fila de la matriz C y y en una columna de la matriz D. Con un tercer for se realiza la multiplicación de dicha fila de la matriz C con la columna de la matriz D, el valor que resulte de la multiplicación será guardado en una variable auxiliar, la cual siempre poseerá el valor de 0 antes de entrar en este for.
4. Haciendo uso del mismo *for* anidado del punto tres se llena la matriz M con los valores que va tomando la variable auxiliar. El primer *for* será para cambiar de fila y el segundo *for* para cambiar de columna. Primero, llena los valores de la primera fila, luego los de la segunda fila y así sucesivamente.
5. La función devuelve la Matriz M.

Función Traspuesta:

Esta función es tipo “double ** ” con el fin de retornar matrices. Los argumentos que requiere son entradas reales *(double **Matriz*) y el tamaño de dicha matriz (*int Tamaño*). Esta función nos permite hallar la traspuesta de una matriz cuadrada A^T. El código de la función es el siguiente:

```
double **Transpuesta(double **Matriz, int Tamaño){
  //1. Se crea la matriz donde se va a guardar la traspuesta.
  double **aux=Crear_Matriz_Ale(Tamaño); ;


  //2. Se guarda la traspuesta en aux.
  for (int i=0; i<Tamaño; i++)
  {
     for (int j=0; j<Tamaño; j++)
     {
        aux[j][i]=Matriz[i][j];
     }
  }
  return aux;
```

```
}
```

A continuación se explica cada numeral del código:

1. Se crea la matriz donde se va a guardar la matriz traspuesta. Para ello se llama la función Crear_Matriz_Ale, debido a que se va a sobrescribir datos en la matriz traspuesta, no importa qué valores tenga inicialmente.
2. Se aplica la definición de Matriz traspuesta, es decir, se escribe los valores de las filas de la matriz A en las columnas de la matriz A^T. A (a_{ij}) -> A^T(a_{ji}). Para esto, se usa un *for* anidado, el primer *for* determina la fila de la Matriz A y la columna de la Matriz A^T, segundo *for* la columna de la Matriz A y la fila de la Matriz A^T.

Función Determinante:

Esta función es tipo “int”, tiene como finalidad calcular el determinante de una funcion. Esta función consiste realmente de dos funciones aparte, sin embargo una no puede funcionar sin la otra. Los argumentos que requiere la primera función (*Determinante*) son una matriz tipo “double” (*doublé **Matriz*), la segunda función (*Cofactor*) tiene com parámetros una matriz de entradas reales (*doublé **Matriz*), el tamaño de la matriz (*int Tamaño*) y la fila y columna del termino al que se le quiere calcular el cofactor (*int Fila, int Columna*). Esta función nos permite hallar la traspuesta de una matriz cuadrada A^T. El código de la función es el siguiente:

```
int Determinante (double **Matriz,int Tamaño)
{
   int determinante=0;

   //D.1 Se plantea el caso de que la matriz sea de tamaño 1x1.
   if (Tamaño==1)
   {
      determinante=Matriz[0][0];
   }

   //D.2 Se plantea el caso de que la matriz sea de tamaño 2x2.
   else if (Tamaño==2)
   {
      determinante=Matriz[0][0]*Matriz[1][1];
      determinante-=(Matriz[0][1]*Matriz[1][0]);
   }

   //D.3 Se plantea el caso de que la matriz sea de tamaño mayor a 2x2.
   else
   {

      //D.3.1 Se recorre la primera fila de la matriz.
      for(int i=0;i<Tamaño;i++)
      {

                  //D.3.2 Se multiplica el elemento de la matriz por su respectivo cofactor.
         determinante += (Matriz[0][i]*Cofactor(Matriz, Tamaño, 0, i));
      }
   }
   return determinante;
}

int Cofactor (double **Matriz, int Tamaño, int Fila, int Columna)
{
   int k=0, l=0, n=Tamaño-1, control=0, cofactor=0;
   double **Menor=Crear_Matriz_Ale(Tamaño-1);

   //C.1 Se recorren cada una de las entradas de la matriz.
   for(int i=0; i<Tamaño; i++)
   {
      for(int j=0; j<Tamaño; j++){

                  //C.2. Si el elemento NO pertenece al menor, se pasa a la siguiente iteracion del 'for' j.
         if (j==Columna||i==Fila){
            continue;
         }

         else{

                     // C.3. Si el elemento SI pertenece al menor, se asigna este valor a la entrada [k][l] del menor.
            Menor[k][l]=Matriz[i][j];

                     //C.4 Se aumenta una unidad a la posición columna de la matriz.
                     l++;
            control+=1;
```

```
            }
        }

        //C.5. Condicional que verifica que el 'for' j se halla ejecutado al menos una vez
        //y una vez ejecutado, baje la fila y reestablezca la posicion de la columna en 0.
        if(control!=0){
            k++;
            l=0;
        }
    }

    //C.6. Se realiza el calculo del cofactor.
    cofactor=pow(-1, Fila+Columna);
    cofactor *= Determinante(Menor, n);

    return cofactor;
}
```

A continuación se realiza la explicación del código:

1. D. 1. Cuando la matriz tiene tamaño 1x1, la matriz es un numero escalar, por lo tanto su determinante es la misma matriz. Este valor se guarda en la variable *determinante.*

2. D. 2. En caso de que la matriz sea de tamaño 2x2, el determinante se calcula de la siguiente forma:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad detA = ad - bc$$

3. Se realiza el cálculo anterior y el resultado se almacena en la variable *determinante.*

4. D. 3. En caso de que la matriz sea cuadrada de orden mayor a 2, se calcula el método de los cofactores:

5. D. 3. 1. Se escoge calcular el determinante de la matriz usando siempre si primera fila.
6. D. 3. 2. Se multiplica la entrada respectiva de la matriz en la primera fila por su cofactor correspondiente. El valor resultante se almacena en la variable *determinante.*

La “semi-funcion” retorna el valor almacenado en *determinante.*

1. C. 1. Se recorren cada una de las entradas de la matriz para luego veriicar cuales están presentes en el menor necesario para calcular el cofactor necesitado.

2. C. 2. Si el elemento se encuentra en la fila (almacenada en la variable *Fila*) o la columna (almacenada en la variable *Columna*) de la entrada de la matriz a la cual se le quiere calcular el cofactor, se rompe la iteración pero el ciclo sigue.

3. C. 3. Si el elemento esta fuera de la fila almacenada en *Fila* y de la columna almacenada en *Columna*, significa que el elemento pertenece al menor que se está buscando, asi que este elemento se guarda en la posición *[k][l]* de la matriz *Menor.*

   La matriz *Menor* fue definida al principio de la “semi-funcion” como una mariz cuadrada de orden n-1 (siendo n el tamaño de la matriz original) y fue inicializada como una matriz aleatoria, por otro lado, las variables *k y l* son enteras inicializadas en cero,

4. C. 4. Una vez que se almacene un valor en la matriz *Menor,* se aumenta en una unidad la variable *l,* la cual determina las columnas de la matriz. Al hacer esto se

pretende correr una entrada a la derecha la “posición de escritura” de los términos en *Menor*.

5. C. 5. El condicional se asegura que la variable *control* no sea cero, esto es, que el for que se encarga de asignar los elementos de una fila del *Menor* se halla ejecutado al menos una vez. En caso de que el for se halla ejecutado al menos una vez, se baja la línea de la “posición de escritura” al sumarle a la variable que determina las filas de *Menor* (*k*) una unidad y reestableciendo la variable que determina las columnas (l), asignándole el valor de cero.

6. C. 6. Se realiza el calculo del signo del cofactor y luego se calcula el determinante del menor encontrado con el procedimiento anterior, se multiplica el signo con el determinante y el resultado se guarda en la variable *Cofactor.*

La “semi-función” devuelve el valor almacenado en la variable *Cofactor.*

BIBLIOGRAFÍA

Grossman, S. (2012). *Algebra lineal*. 5th ed. México, D.F.: McGraw-Hill.

juanse, M. (2017). Compactar una matriz tridiagonal - Monografias.com. [en línea] Monografias.com. Disponible en: http://www.monografias.com/trabajos92/factorizacion-matrices/factorizacion-matrices.shtml [Revisado el 8 de mayo 2017].

Es.wikipedia.org. (2017). Factorización de Cholesky. [en línea] Disponible en: https://es.wikipedia.org/wiki/Factorización_de_Cholesky [Revisado 8 mayo 2017].

University of the Basque Country. (2017). *Factorización LU de matrices cuadradas*. [en línea] Disponible en: http://www.ehu.eus/juancarlos.gorostizaga/mn11b/temas/factorizac_LU.pdf [Revisado 8 mayo 2017].

Mathonline.wikidot (2017). Doolittle's Method for LU Decompositions [en línea] Disponible en: http://mathonline.wikidot.com/doolittle-s-method-for-lu-decompositions [Revisado el 8 de mayo 2017]