Tarea 03

Pablo Antonio Stack Snchez
Métodos Numéricos

August 25, 2019

1 Solución de matriz diagonal

En álgebra lineal, una matriz diagonal es una matriz cuadrada en que las entradas de la matriz diagonal son todas nulas salvo en la diagonal principal. Obteniendose un sistema de ecuaciónes de la forma $AX = B$:

$$\begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Realizando la multiplicación de las matrices A y X e igualando a B obtenemos:

$$\begin{aligned} x_1 &= \frac{b_1}{a_{11}} \\ x_2 &= \frac{b_2}{a_{22}} \\ &\vdots \\ x_i &= \frac{b_i}{a_{ii}} \end{aligned} \tag{1}$$

En donde $i = 1, 2, 3..., n$
Se programó la formula (1) en C, se probó el algoritmo con la matriz M que se muestra a continuación:

$$M = \left[\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 3 \\ 0 & 0 & 0 & 4 & 4 \end{array}\right]$$

Se obtuvo el siguiente resultado:

$$\begin{aligned} x_1 &= 1 \\ x_2 &= 1 \\ x_3 &= 1 \\ x_4 &= 1 \end{aligned}$$

Se obtuvo un determinante de 24,es importante verificar que los elementos de la diagonal sean diferentes de 0, de otro modo el sistema de ecuaciones no tendrá solucion.

```
EL resultado es: 1.000000 1.000000 1.000000 1.000000

El determinante es: 24.000000==21969==
==21969== HEAP SUMMARY:
==21969==     in use at exit: 0 bytes in 0 blocks
==21969==   total heap usage: 10 allocs, 10 frees, 10,512 bytes allocated
==21969==
==21969== All heap blocks were freed -- no leaks are possible
==21969==
==21969== For counts of detected and suppressed errors, rerun with: -v
==21969== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 1: Ejemplo matriz diagonal

2 Solución de matriz triangular superior

Una matriz triangular superior es una matriz cuyos elementos por debajo de la diagonal son 0. Obteniendose un sistema de ecuaciónes de la forma $UX = B$:

$$\begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Realizando la multiplicación de las matrices U y X e igualando a B obtenemos:

$$x_n = \frac{b_n}{u_{nn}} \tag{2}$$

$$x_{[n-i]} = \frac{b_{[n-i]} - \sum_{j=1}^{i-1} u_{[n-i][n-j]} * x_{[n-j]}}{u_{[n-i][n-i]}} \tag{3}$$

Se programaron las ecuaciones (2) y (3) en C. Un ejemplo para comprobar que el programa funciona correctamente fue la matriz M_1, que se muestra a continuación:

$$M_1 = \left[\begin{array}{cccc|c} 1 & 2 & 3 & 4 & 1 \\ 0 & 5 & 6 & 7 & 2 \\ 0 & 0 & 8 & 9 & 3 \\ 0 & 0 & 0 & 10 & 4 \end{array}\right]$$

Se obtuvo el siguiente resultado:

$$\begin{aligned} x_1 &= -0.235 \\ x_2 &= -0.070 \\ x_3 &= -0.075 \\ x_4 &= 0.400 \end{aligned}$$

Se obtuvo un determinante de 400, de igual forma se debe verificar que los elementos de la diagonal principal sean distintos de 0.

```
EL resultado es: -0.235000 -0.070000 -0.075000 0.400000

El determinante es: 400.000000==22503==
==22503== HEAP SUMMARY:
==22503==     in use at exit: 0 bytes in 0 blocks
==22503==   total heap usage: 10 allocs, 10 frees, 10,512 bytes allocated
==22503==
==22503== All heap blocks were freed -- no leaks are possible
==22503==
==22503== For counts of detected and suppressed errors, rerun with: -v
==22503== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 2: Ejemplo matriz triangular superior

3 Solución de matriz triangular inferior

Una matriz triangular inferior es una matriz cuyos elementos por encima de la diagonal principal son 0. Obteniendose un sistema de ecuaciónes de la forma $LX = B$:

$$\begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Realizando la multiplicación de las matrices L y X e igualando a B obtenemos:

$$x_1 = \frac{b_1}{l_{11}} \tag{4}$$

$$\vdots$$

$$x_i = \frac{b_i - \sum_{j=0}^{i-1} l_{ij} * x_i}{l_{ii}} \tag{5}$$

Se programaron las ecuaciones (4) y (5) en C. Un ejemplo para comprobar que el programa funciona correctamente fue la matriz M_2, que se muestra a continuación:

$$M_2 = \left[\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 1 \\ 2 & 3 & 0 & 0 & 2 \\ 4 & 5 & 6 & 0 & 3 \\ 7 & 8 & 9 & 10 & 4 \end{array}\right]$$

Se obtuvo el siguiente resultado:

$$\begin{aligned} x_1 &= 1 \\ x_2 &= 0 \\ x_3 &= -0.1667 \\ x_4 &= -0.1500 \end{aligned}$$

Se obtuvo un determinante de 180, de igual forma se debe verificar que los elementos de la diagonal principal sean distintos de 0.

```
EL resultado es: 1.000000 0.000000 -0.166667 -0.150000

El determinante es: 180.000000==21738==
==21738== HEAP SUMMARY:
==21738==     in use at exit: 0 bytes in 0 blocks
==21738==   total heap usage: 10 allocs, 10 frees, 10,512 bytes allocated
==21738==
==21738== All heap blocks were freed -- no leaks are possible
==21738==
==21738== For counts of detected and suppressed errors, rerun with: -v
==21738== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 3: Ejemplo matriz triangular inferior

4 Solución de un sistema de ecuaciones por Gauss

El método de eliminación de Gauss consiste en operar sobre la matriz ampliada del sistema hasta hallar la forma escalonada (una matriz triangular superior). Así, se obtiene un sistema fácil de resolver por sustitución hacia atrás.

A continuación se presenta el pseudocódigo del algoritmo implementado en C:

M= Matriz aumentada
n= Número de filas
c= Número de columnas
r= Número de renglones

1) Iniciar con c=0 y r=0
2) Mientras r<n
    a) Si $M_{rc} == 0$
        i) Sea $max = argmax_{i=r,...,n-1}(M_{ic})$
        ii) Si $M_{rc}! = max$
            I) Intercambia el renglón r con el renglón en donde se encontró el máximo
    b) Multiplicar el renglón r por $\frac{1}{M_{rc}}$
    c) Para cada renglón $i = r + 1, ..., n.$ sumar al renglón i el resultado de multiplicar el renglón r por $-M_{ic}$
    d) r_{++}, c_{++}
3) Resolver la matriz resultante utilizando el método para solucionar una matriz triangular superior visto en la sección 2.

Como se observa en el pseudocódigo se implementó un pivoteo en los renglones para que el programa resuelva sistemas en los cuales la diagonal se hace 0. En la matriz M_3 se muestra un ejemplo con el que se probó este método:

$$M_3 = \left[\begin{array}{cccc|c} 2.402822 & 4.425232 & 1.929374 & 1.370355 & 0.060000 \\ 1.201411 & 2.212616 & 0.964687 & 0.685178 & 0.542716 \\ 1.119958 & 0.964687 & 2.053172 & 0.566574 & 0.857204 \\ 0.742142 & 0.685178 & 0.566574 & 1.696828 & 0.761270 \end{array}\right]$$

Se obtuvo el siguiente resultado:

$$\begin{aligned} x_1 &= -5425478.028316 \\ x_2 &= 1837966.091718 \\ x_3 &= 1812933.000000 \\ x_4 &= 1025432.000487 \end{aligned}$$

Se obtuvo un determinante igual a 0.000001.

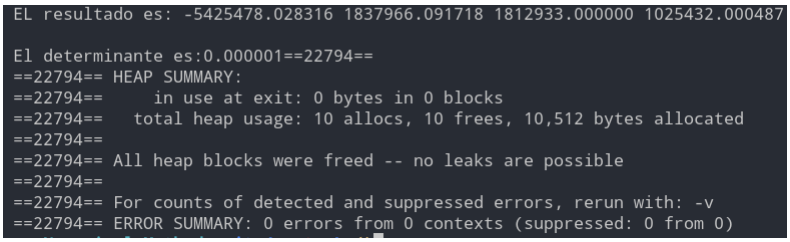

Figure 4: Ejemplo Gauss sin pivoteo

5 Solución de un sistema de ecuaciones por Gauss con pivoteo

Este método es una modifiación al expuesto en la sección anterior. El pivoteo no solo se hará en los renglones, se buscará el máximo elemento de la mátriz y se colocará en la posición del pivote correspondiente, sin importar si este se hace 0 o no. A continuación se muestra el pseudocodigo:

M= Matriz aumentada
n= Número de filas
c= Número de columnas
r= Número de renglones
max=Máximo elemento de la matriz

1) Iniciar con c=0 y r=0
2) Mientras r<n
    a) max=encontrar elemento máximo de la matriz y su posición
    b) Si $M_{rc}! = max$
        i) Si r =! renglón de la posicion de max
            I) Intercambia el renglón r con el renglón en donde se encontró el máximo
        ii) Si c =! columna de la posición de max
            I) Intercambia la columna c con la columna en donde se encontró el máximo
    c) Multiplicar el renglón r por $\frac{1}{M_{rc}}$
    d) Para cada renglón $i = r + 1, ..., n$. sumar al renglón i el resultado de multiplicar el renglón r por $-M_{ic}$
    e) r_{++}, c_{++}
3) Resolver la matriz resultante utilizando el método para solucionar una matriz triangular superior visto en la sección 2.

En la matriz M_4 se muestra un ejemplo con el que se probó este método:

$$M_4 = \left[\begin{array}{cccc|c} 2.402822 & 4.425232 & 1.929374 & 1.370355 & 0.060000 \\ 1.201411 & 2.212616 & 0.964687 & 0.685178 & 0.542716 \\ 1.119958 & 0.964687 & 2.053172 & 0.566574 & 0.857204 \\ 0.742142 & 0.685178 & 0.566574 & 1.696828 & 0.761270 \end{array}\right]$$

Se obtuvo el siguiente resultado:

$$\begin{aligned} x_1 &= -5425478.826295 \\ x_2 &= 1837966.781259 \\ x_3 &= 1812933.656220 \\ x_4 &= 1025432.002494 \end{aligned}$$

Se obtuvo un determinante igual a 0.000001.

```
EL resultado es: -5425479.826295 1837966.781259 1812933.656220 1025432.002494

El determinante es: 0.000001==23144==
==23144== HEAP SUMMARY:
==23144==     in use at exit: 0 bytes in 0 blocks
==23144==   total heap usage: 10 allocs, 10 frees, 10,512 bytes allocated
==23144==
==23144== All heap blocks were freed -- no leaks are possible
==23144==
==23144== For counts of detected and suppressed errors, rerun with: -v
==23144== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 5: Ejemplo Gauss con pivoteo

6 Factorización LU (Doolittle)

La factorización o descomposicin LU es una forma de factorización de una matriz como el producto de una matriz triangular inferior (L) y una superior (U) de la forma $LUX = B$ en donde $A = LU$. El método de Doolittle consiste en hacer 1's la diagonal de la matriz L, con lo cual se formaría el siguiente sistema de ecuaciones:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Realizando la multiplicación de las matrices L y U e igualando a A obtenemos:

$$u_{1i} = a_{1i} \tag{6}$$
$$l_{ij} = \frac{a_{ij}}{u_{jj}} \tag{7}$$
$$l_{ij} = \frac{a_{ij} - \sum_{k=0}^{j-1} l_{ik} * u_{kj}}{u_{jj}} \tag{8}$$
$$l_{ij} = a_{ij} - \sum_{k=0}^{i-1} l_{ik} * u_{kj} \tag{9}$$

Se programaron las ecuaciones (6),(7), (8) y (9). Fue necesario implementar un pivoteo ya que al hacerse cero la diagonal la ecuacion (8) se indetermina y no es posible calcular correctamente la factorización. Ya con L y U calculados se debe formar un nuevo sistema de ecuaciones:

$$LY = B \tag{10}$$

En donde $Y = UX$.
(10) debe ser resuelto mediante el programa que da solución a una matriz triangular inferior. Los valores obtenidos se utilizarán para resolver el sistema triangular superior $UX = Y$ y con esto obtener los valores x_i. En la matriz M_5 se muestra un ejemplo con el que se probó este método:

$$M_5 = \left[\begin{array}{cccc|c} 2.402822 & 4.425232 & 1.929374 & 1.370355 & 0.060000 \\ 1.201411 & 2.212616 & 0.964687 & 0.685178 & 0.542716 \\ 1.119958 & 0.964687 & 2.053172 & 0.566574 & 0.857204 \\ 0.742142 & 0.685178 & 0.566574 & 1.696828 & 0.761270 \end{array}\right]$$

Se obtuvo el siguiente resultado:

$$x_1 = -5425478.813547$$
$$x_2 = 1837966.776940$$
$$x_3 = 1812933.651960$$
$$x_4 = 1025432.000084$$

Se obtuvo un determinante igual a 0.000001.

```
EL resultado es: -5425479.813547 1837966.776940 1812933.651960 1025432.000084
El determinante es: 0.000001==23611==
==23611== HEAP SUMMARY:
==23611==     in use at exit: 0 bytes in 0 blocks
==23611==   total heap usage: 10 allocs, 10 frees, 10,512 bytes allocated
==23611==
==23611== All heap blocks were freed -- no leaks are possible
==23611==
==23611== For counts of detected and suppressed errors, rerun with: -v
==23611== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 6: Ejemplo Factorización LU (Doolittle)

7 Inversa de una matriz utilizando factorización LU

Por definición, la matriz inversa es aquella que, multiplicada por la original, da lugar a la matriz identidad I:

$$I = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Para calcular la matriz inversa a través de las matrices L y U obtenidas de la factorización LU se debe construir un sistema de ecuaciones de la forma $LUX = I_{(:,i)}$ en donde $i = 1, ..., n$. Se resuelve para todas las columnas de i y cada X obtenida es una columna de la matriz inversa. A continuación se muestra el pseudocodigo que se programó en C:

L= Matriz triangular inferior obtenida de la factorización LU
L= Matriz triangular superior obtenida de la factorización LU
n= Número de filas
c= Número de columnas

1) Crear matriz identidad de tamao nxn
2) Desde i=0 hasta $i < n$
    a) Desde j=0 hasta $j < n$
        a.i) $L_{jm} = Identidad_{ji}$
    end
    c) $Inversa = solvtriangularinferior(L)$
end
2) Desde i=0 hasta $i < n$
    a) Desde j=0 hasta $j < n$
        a.i) $U_{jm} = Inversa_{ji}$
    end
    c) $Inversa = solvtriangulasuperior(U)$
end

En la matriz M_6 se presenta un ejemplo con el que se probó este método:

$$M_6 = \begin{bmatrix} 3192.302 & 2698.635 & 1978.379 & 2126.596 & 2790.948 & 3222.904 & 2115.401 & 2977.436 & 2323.447 & 2064.871 \\ 2698.635 & 3648.584 & 2353.185 & 2553.399 & 2775.733 & 2717.937 & 2426.996 & 3440.832 & 2690.923 & 1824.800 \\ 1978.379 & 2353.185 & 2832.480 & 1721.412 & 2479.264 & 2633.894 & 2114.959 & 2189.508 & 2124.635 & 1680.251 \\ 2126.596 & 2553.399 & 1721.412 & 2457.944 & 1722.838 & 2208.851 & 1909.113 & 2425.035 & 1814.554 & 1303.733 \\ 2790.948 & 2775.733 & 2479.264 & 1722.838 & 3632.273 & 3450.131 & 2061.848 & 2784.201 & 2362.622 & 2397.096 \\ 3222.904 & 2717.937 & 2633.894 & 2208.851 & 3450.131 & 4206.092 & 2283.862 & 2789.959 & 2211.154 & 2503.426 \\ 2115.401 & 2426.996 & 2114.959 & 1909.113 & 2061.848 & 2283.862 & 2666.696 & 2482.297 & 2241.853 & 1505.690 \\ 2977.436 & 3440.832 & 2189.508 & 2425.035 & 2784.201 & 2789.959 & 2482.297 & 3883.093 & 2756.404 & 1785.132 \\ 2323.447 & 2690.923 & 2124.635 & 1814.554 & 2362.622 & 2211.154 & 2241.853 & 2756.404 & 2866.823 & 1814.375 \\ 2064.871 & 1824.800 & 1680.251 & 1303.733 & 2397.096 & 2503.426 & 1505.690 & 1785.132 & 1814.375 & 2255.474 \end{bmatrix}$$

Se obtuvo el siguiente resultado:

```
La inversa es:
0.003806 0.000589 0.001092 -0.000560 0.000058 -0.002212 0.000150 -0.001391 -0.001111 -0.000162
0.000589 0.003850 -0.000187 -0.002248 -0.001669 0.000874 0.000157 -0.001582 -0.000535 0.000166
0.001092 -0.000187 0.001833 -0.000210 -0.000443 -0.001077 -0.000451 0.000240 -0.000851 0.000369
-0.000560 -0.002248 -0.000210 0.002955 0.001868 -0.001045 -0.000345 0.000174 0.000384 -0.000262
0.000058 -0.001669 -0.000443 0.001868 0.003314 -0.001746 0.000282 -0.000338 -0.000017 -0.000943
-0.002212 0.000874 -0.001077 -0.001045 -0.001746 0.003576 -0.000320 0.000337 0.001397 -0.000566
0.000150 0.000157 -0.000451 -0.000345 0.000282 -0.000320 0.001642 -0.000337 -0.000700 0.000060
-0.001391 -0.001582 0.000240 0.000174 -0.000338 0.000337 -0.000337 0.002646 -0.000376 0.000692
-0.001111 -0.000535 -0.000851 0.000384 -0.000017 0.001397 -0.000700 -0.000376 0.002632 -0.001023
-0.000162 0.000166 0.000369 -0.000262 -0.000943 -0.000566 0.000060 0.000692 -0.001023 0.002199

Inversa por matriz original:
1.000000 -0.000000 -0.000000 -0.000000 -0.000000 -0.000000 0.000000 -0.000000 -0.000000 -0.000000
-0.000000 1.000000 -0.000000 -0.000000 -0.000000 -0.000000 -0.000000 -0.000000 -0.000000 -0.000000
0.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 -0.000000 0.000000 1.000000 -0.000000 0.000000 0.000000 -0.000000 0.000000
-0.000000 -0.000000 -0.000000 -0.000000 -0.000000 1.000000 -0.000000 -0.000000 0.000000 -0.000000
-0.000000 -0.000000 -0.000000 -0.000000 -0.000000 -0.000000 1.000000 -0.000000 -0.000000 -0.000000
-0.000000 0.000000 0.000000 0.000000 0.000000 -0.000000 0.000000 1.000000 0.000000 0.000000
0.000000 -0.000000 0.000000 -0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 -0.000000
-0.000000 0.000000 0.000000 0.000000 0.000000 -0.000000 0.000000 -0.000000 0.000000 1.000000
==23767==
==23767== HEAP SUMMARY:
==23767==     in use at exit: 0 bytes in 0 blocks
==23767==   total heap usage: 49 allocs, 49 frees, 14,080 bytes allocated
==23767==
==23767== All heap blocks were freed -- no leaks are possible
==23767==
==23767== For counts of detected and suppressed errors, rerun with: -v
==23767== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 7: Ejemplo inversa con LU (Doolittle)

8 Factorización LDL^T (Cholesky modificado)

Este tipo de factorización solo es aplicable a matrices definidas positivas y simétricas. Tiene como ventaja sobre Cholesky tradicional que se evita calcular raices cuadradas. Sean L una matriz triangular inferior y D una matriz diagonal, entonces:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & 1 \end{bmatrix} \begin{bmatrix} d_{11} & 0 & \dots & 0 \\ 0 & d_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & d_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{12} & \dots & l_{1n} \\ 0 & l_{22} & \dots & l_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & l_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Multiplicando, igualando a A y despejando para X obtenemos la siguientes formulas:

$$d_{ii} = a_{ii} - \sum_{k=0}^{i-1} l_{ik}^2 * d_{kk} \tag{11}$$

$$l_{ij} = a_{ij} - \sum_{k=0}^{j-1} l_{jk} * l_{ik} * d_{kk} \tag{12}$$

Se programaron las formulas (11) y (12). En pimera instancia se probó la matriz M_7:

$$M_7 = \left[\begin{array}{cccc|c} 2.402822 & 4.425232 & 1.929374 & 1.370355 & 0.060000 \\ 1.201411 & 2.212616 & 0.964687 & 0.685178 & 0.542716 \\ 1.119958 & 0.964687 & 2.053172 & 0.566574 & 0.857204 \\ 0.742142 & 0.685178 & 0.566574 & 1.696828 & 0.761270 \end{array}\right]$$

Se obtuvo el siguiente resultado:

```
No es una matriz simetrica, no se puede aplicar este metodo==24690==
==24690== HEAP SUMMARY:
==24690==     in use at exit: 0 bytes in 0 blocks
==24690==   total heap usage: 10 allocs, 10 frees, 10,512 bytes allocated
==24690==
==24690== All heap blocks were freed -- no leaks are possible
==24690==
==24690== For counts of detected and suppressed errors, rerun with: -v
==24690== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 8: Ejemplo Cholesky

Como se observa en la figura 8. la matriz utilizada como entrada no es simétrica, por lo que no es posible utilizar este método.

9 Función para calcular el máximo

Pseudocodigo:

Matriz=Matriz a la que se le calcula el máximo.
r= Renglon del pivote en el que nos encontramos.
c= columna del pivote en el que nos encontramos.
m= Número de filas de la matriz.
n= Número de columnas de la matriz.

a) Double max= $abs(matriz_{rc})$
b) Desde i=r hasta $i < n$
    b.i) Desde j=0 hasta $j < n$
        b.ii) si $abs(matriz_{ij}) > abs(max)$
            b.iii) $max = abs(matriz_{ij})$

```
            end
        end
end
```

En la matriz M_8 se observa un ejemplo con el que se probó este método:

$$M_8 = \begin{bmatrix} 3192.302 & 2698.635 & 1978.379 & 2126.596 & 2790.948 & 3222.904 & 2115.401 & 2977.436 & 2323.447 & 2064.871 \\ 2698.635 & 3648.584 & 2353.185 & 2553.399 & 2775.733 & 2717.937 & 2426.996 & 3440.832 & 2690.923 & 1824.800 \\ 1978.379 & 2353.185 & 2832.480 & 1721.412 & 2479.264 & 2633.894 & 2114.959 & 2189.508 & 2124.635 & 1680.251 \\ 2126.596 & 2553.399 & 1721.412 & 2457.944 & 1722.838 & 2208.851 & 1909.113 & 2425.035 & 1814.554 & 1303.733 \\ 2790.948 & 2775.733 & 2479.264 & 1722.838 & 3632.273 & 3450.131 & 2061.848 & 2784.201 & 2362.622 & 2397.096 \\ 3222.904 & 2717.937 & 2633.894 & 2208.851 & 3450.131 & 4206.092 & 2283.862 & 2789.959 & 2211.154 & 2503.426 \\ 2115.401 & 2426.996 & 2114.959 & 1909.113 & 2061.848 & 2283.862 & 2666.696 & 2482.297 & 2241.853 & 1505.690 \\ 2977.436 & 3440.832 & 2189.508 & 2425.035 & 2784.201 & 2789.959 & 2482.297 & 3883.093 & 2756.404 & 1785.132 \\ 2323.447 & 2690.923 & 2124.635 & 1814.554 & 2362.622 & 2211.154 & 2241.853 & 2756.404 & 2866.823 & 1814.375 \\ 2064.871 & 1824.800 & 1680.251 & 1303.733 & 2397.096 & 2503.426 & 1505.690 & 1785.132 & 1814.375 & 2255.474 \end{bmatrix}$$

Se obtuvo el siguiente resultado:

```
El maximo es: 4206.092000
fila: 5
columna: 5==25697==
==25697== HEAP SUMMARY:
==25697==     in use at exit: 0 bytes in 0 blocks
==25697==   total heap usage: 14 allocs, 14 frees, 6,632 bytes allocated
==25697==
==25697== All heap blocks were freed -- no leaks are possible
==25697==
==25697== For counts of detected and suppressed errors, rerun with: -v
==25697== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 9: Ejemplo Máximo de una matriz

10 Mejoras que se pueden hacer en el codigo

1) Al pedir memoria dinámica con la función "Malloc" se podría crear un arreglo de longitud mxn y los apuntadores de las filas apuntarlos al inicio correspondiente de cada rengón.

2) Los cambios de renglones se podrían hacer cambiando las direcciones de memoría en lugar de hacerlos paso por paso.

3) Utilizar estructuras.