Tarea 3:
Sistemas de ecuaciones lineales

Resumen

En este trabajo se programaron distintos métodos númericos para resolver sistemas de ecuaciones lineales.

1. Introducción

Un conjunto de ecuaciones lineales busca resolver el sistema que se presenta en la ec. 1, en este caso el sistema consta de N incógnitas y M ecuaciones. Los términos a_{ij} son los elemtentos de la matriz A que dependiendo de su valor, se podrá resolver de distintas maneras [1].

$$\begin{pmatrix} a_{00}x_0 & a_{01}x_1 & \cdots & a_{0,N-1}x_{N-1} \\ a_{10}x_0 & a_{11}x_1 & \cdots & a_{1,N-1}x_{N-1} \\ \cdots & \cdots & \cdots & \cdots \\ a_{M-1,0}x_0 & a_{M-1,1}x_1 & \cdots & a_{M-1,N-1} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{M-1} \end{pmatrix}. \tag{1}$$

2. Metodología

2.1. Diferencias finitas y criterio de la primera y segunda derivada para encontrar máximos y mínimos.

La derivada de una función f en x_0 es:

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}. \tag{2}$$

Para desarrollar la derivada numérica, la función debe ser continua en un intervalo [a,b], y haciendo la aproximación numérica se tendrá:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h}. \tag{3}$$

Además, si nos interesa la segunda derivada, se aplica la ec. 2 en la ec. 3, y se obtiene:

$$f''(x_0) = \frac{f((x_0 + h) + h) - 2f(x_0 + h) + f(x_0)}{h^2}. \tag{4}$$

La primera y la segunda derivada de una función nos sirve para encontrar sus máximos y mínimos, cuando la primera derivada es igual a cero, se encuentran los puntos críticos de la función sin saber de qué tipo son, y al evaluar en la segunda derivada, se encuentran si son máximos o mínimos dependiendo de su signo.

Algorithm 1 Máximos y mínimos de f(x)

Entrada: f: $\in C^2[a,b]$
Salida: Máximo(s) y mínimo(s) de la función

Ingresar valores de: h, a, evaluaciones, TOL = 10^{-2}
vector puntos criticos, vector maximos, vector minimos
vector maxx, vector minn

function PRIMERA DERIVADA(x)
    derivada ← (f(x+h)-f(x))/h
    retorna: derivada

function SEGUNDA DERIVADA(x)
    derivada2 ← (f(x+2h)-2f(x+h)+f(x))/h^2
    retorna: derivada2

for $i = a;\ i <$ evaluaciones; $i++$; **do**
    if derivada[entero(i*h)] $== 0$ **then**
        guardar vector(puntos criticos)

for $i = 0;\ i <$ size(puntos criticos); $i++$; **do**
    if derivada2[puntos criticos[i]]=< TOL **then**
        if derivada2[puntos criticos[i]] < 0 **then**
            guardar vector máximos
        else
            guardar vector mínimos

Si por el error aparecen máximos o mínimos con diferencias mínimas, ejem: 7.9998, 7.999, 8.000
for $i = 0$; $i <$ size(vector maximos); $++i$ **do**
    if maximo[i]*maximo[i+1] < 0 **then**
        maxx ← maximo[i]

for $i = 0$; $i <$ size(vector minimos); $++i$ **do**
    if minimo[i]*minimo[i+1] < 0 **then**
        minn ← minn[i]
return maxx, minn

2.2. Sistema de ecuaciones

La ec. 1 también se puede representar de la forma

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}, \tag{5}$$

donde A es la matriz de coeficientes y b se escribe como vector columna del lado derecho de la ecuación.

$$\begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0m} \\ a_{10} & a_{11} & \cdots & a_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n0} & a_{n1} & \cdots & a_{nm} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}. \tag{6}$$

2.2.1. Caso 1: A es una matriz diagonal

$$A = \begin{pmatrix} a_{00} & 0 & \cdots & 0 \\ 0 & a_{11} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix}, \tag{7}$$

donde se encuentra que

$$x_i = \frac{b_i}{a_{ii}}. \tag{8}$$

Algorithm 2 Caso 1: Inversa, determinante matriz diagonal

Entrada: Matriz A, vector b
Salida: soluciones x, inversa A, determinante A
1: Vector x vacío, n = dimensión de A, matriz vacía inversa (inicializada en cero), determinante = 1
2: Leer A, b

3: **for** $i = 0;\ i < n;\ i++$; **do**
4: x[i] = b[i]/A[i][i]
5: **for** $i = 0;\ i < n;\ i++$; **do**
6: Inversa[i][i] = 1/A[i][i]
7: determinante = determinante*A[i][i]
8: **return** x, archivo inversa.txt, e imprime determinante

2.2.2. Caso 2. A es una matriz diagonal inferior

$$\begin{pmatrix} a_{00} & 0 & 0 & 0 \\ a_{10} & a_{11} & 0 & 0 \\ a_{20} & a_{21} & a_{22} & 0 \\ a_{30} & a_{31} & a_{31} & a_{32} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}, \tag{9}$$

y despejando las ecuaciones, se encontró la ec.10.

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=0}^{i-1} a_{ij} x_j \right). \tag{10}$$

Algorithm 3 Caso 2: A es triangular inferior

Entrada: Matriz A, vector b
Salida: vector x (solución)
1: Vector x vacío, n = filas de A, suma = 0
2: Leer A, b

3: **for** $i = 1;\ i < n;\ i++$; **do**
4: **for** $j = 0;\ j < n;\ j++$ **do**
5: suma = suma + $a_{ij}x_j$
6: x_i = (b_i-suma)/a_{ii}
7: **return** x

2.2.3. Caso 3. A es triangular superior

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ 0 & a_{11} & a_{12} & a_{13} \\ 0 & 0 & a_{22} & a_{23} \\ 0 & 0 & 0 & a_{33} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}, \tag{11}$$

donde, al despejar, se encuentra la fórmula:

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=i+1}^{N-1} a_{ij} x_j \right). \tag{12}$$

Algorithm 4 Caso 3: A es triangular superior

Entrada: Matriz A, vector b
Salida: vector x (solución)
1: Vector x vacío, n = filas de A, suma = 0
2: Leer A, b

3: **for** $i = 1;\ i < n;\ i++$; **do**
4: **for** $j = i + 1;\ j < n - 1;\ j++$ **do**
5: suma = suma + $a_{ij}x_j$
6: x_i = (b_i-suma)/a_{ii}
7: **return** x

2.3. Caso 4. A es una matriz completa

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} \tag{13}$$

Para la eliminación Gaussiana se debe de hacer una reducción de la matriz A, esta se reduce a una triangular superior y se hace un despeje inverso para encontrar los valores de x. Si la matriz A tiene alguno de sus elementos diagonales igual a cero, es necesario hacer un pivoteo de filas o de columnas para que la fómula funcione. Para hacerla triangular superior se utiliza la ec.14. Además, todos los cambios que se le hagan a la matriz A, se deben de hacer al vector b, por ello es que se utiliza la notación AB en el pseudocódigo.

$$a_{jk} = a_{jk} - \frac{a_{ji} \cdot a_{ik}}{a_{ii}}, \tag{14}$$

después de este paso, se hace una sustitución hacia atrás y con esto se encuentran los valores de las incóginitas x.

Algorithm 5 Caso 4: A es una matriz completa

Entrada: Matriz A, vector b
Salida: vector x (solución)
1: Vector x vacío, n = filas de A, suma = 0
2: Leer A, b, matriz aumentada AB,

3: **for** $i = 0;\ i < N;_1++$ **do**
4: **if** AB[i][i] == 0 **then**
5: break;

6: **for** $j = 0;\ j < n;\ j++$; **do**
7: **if** (i != j) **then** // buscamos hacerla matriz triangular superior
8: suma ← AB[j][i]/AB[i][i];
9: **for** $k = 0\ k <= N; k++$ **do**
10: AB[j][k] ← AB[j][k]-suma*AB[i][k]
11: x sol ← AB[i][N]/AB[i][i];
12: **return** x sol;

3. Factorización de una matriz completa

3.1. Descomposición LU Crout

Este primer algoritmo, pone todos los elementos de la digonal de L igual a 1.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ L_{10} & 1 & 0 & 0 \\ L_{20} & L_{21} & 1 & 0 \\ L_{30} & L_{31} & L_{32} & 1 \end{pmatrix} \begin{pmatrix} U_{00} & U_{01} & U_{02} & U_{03} \\ 0 & U_{11} & U_{12} & U_{13} \\ 0 & 0 & U_{22} & U_{23} \\ 0 & 0 & 0 & U_{33} \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}, \tag{15}$$

así, con una factorización LU es posible encontrar el valor de x, para ciertas matrices.

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$$
$$\mathbf{LU} \cdot \mathbf{x} = \mathbf{b}$$
$$\mathbf{L} \cdot \mathbf{(Ux)} = \mathbf{b}$$

Si llamamos a Ux como y, se tendría:

$$\mathbf{L} \cdot \mathbf{y} = \mathbf{b}$$
$$\mathbf{U} \cdot \mathbf{x} = \mathbf{y}$$

Para la factorización se utilizaron las fórmulas:

$$U_{ij} = a_{ij} - \sum_{k=0}^{i-1} L_{ik} U_{kj}, \tag{16}$$

$$L_{ij} = \frac{1}{U_{jj}} \left(a_{ij} - \sum_{k=0}^{j-1} L_{ik} U_{kj} \right). \tag{17}$$

Para la resolución de las incógnitas:

$$y_0 = \frac{b_0}{L_{00}}, \tag{18}$$

$$y_i = \frac{1}{L_{ii}} \left[b_i - \sum_{j=0}^{i-1} L_{ij} y_j \right], \tag{19}$$

y para resolver $\mathbf{U} \cdot \mathbf{x} = \mathbf{y}$:

$$x_{N-1} = \frac{y_{N-1}}{U_{N-1,N-1}}, \tag{20}$$

$$x_i = \frac{1}{U_{ii}} \left[y_i - \sum_{j=i+1}^{N-1} U_{ij} x_j \right]. \tag{21}$$

Algorithm 6 Factorización LU

Entrada: Matriz A, vector b
Salida: vector x (solución)

Vector x vacío, n = filas de A, suma = 0
Matriz vacía L, U

Recorrer filas y columnas
Paso 1: Descomponer la matriz en L,U
for $i = 0;\ i < N; i++$ **do**
    $L_{ii} \leftarrow 1.0$;
    for $j = 0;\ j < N; j++$ **do**
        if $i > j$ **then**
            suma = 0
            for $k = 0\ k < i - 1;\ k++$ **do**
                suma = suma + $L_{ik}U_{kj}$
            $U_{ij} = a_{ij} - suma$;
        else
            suma = 0
            for $k = 0;\ k < j - 1;\ k++$ **do**
                suma = suma + $L_{ik}U_{kj}$
            $L_{ij} = \frac{1}{U_{jj}}(a_{ij} - suma)$

    Paso 2: Encontrar y
    y ← **función** triangular inferior (L, b)

    x ← **función** triangular superior (U, y)
    return L,U, x; =0

En el caso de la factorización de Doolittle, los valores que se ponen igual a 1 son los de la diagonal de la matriz U.

$$\begin{pmatrix} L_{00} & 0 & 0 & 0 \\ L_{10} & L_{11} & 0 & 0 \\ L_{20} & L_{21} & L_{22} & 0 \\ L_{30} & L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} 1 & U_{01} & U_{02} & U_{03} \\ 0 & 1 & U_{12} & U_{13} \\ 0 & 0 & 1 & U_{23} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}. \tag{22}$$

Haciendo algunas multiplicaciones para deducir el algoritmo:

$$\begin{aligned} L_{00} &= a_{00}, \\ U_{01} &= \frac{a_{01}}{L_{00}}, \\ U_{02} &= \frac{a_{02}}{L_{00}}, \\ U_{03} &= \frac{a_{03}}{L_{00}}, \\ L_{10} &= a_{10}, \\ L_{11} &= a_{11} - L_{10}U_{01} \\ U_{12} &= \frac{a_{12-L_{10}U_{02}}}{L_{11}}, \\ U_{13} &= \frac{a_{13} - L_{10}U_{03}}{a_{11}}, \end{aligned}$$

se encuentran las fórmulas

$$\text{para } i >= j \tag{23}$$

$$L_{ij} = a_{ij} - \sum_{k=0}^{j-1} L_{ik}U_{kj},$$

$$\text{para } i < j \tag{24}$$

$$U_{ij} = \frac{a_{ij} - \sum_{k=0}^{i-1} L_{ik}U_{kj}}{L_{ii}}.$$

Algorithm 7 Factorización LU: Doolittle

Entrada: Matriz A, vector b
Salida: vector x (solución)

Vector x vacío, n = filas de A, suma = 0
Matriz vacía L, U

Recorrer filas y columnas
Paso 1: Descomponer la matriz en L,U
for $i = 0;\ i < N; i++$ **do**
    $U_{ii} \leftarrow 1.0$;
    for $j = 0;\ j < N; j++$ **do**
        if $i >= j$ **then**
            suma = 0
            for $k = 0\ k < j - 1;\ k++$ **do**
                suma = suma + $L_{ik}U_{kj}$
            $L_{ij} = a_{ij} - suma$;
        else
            suma = 0
            for $k = 0;\ k < j - 1;\ k++$ **do**
                suma = suma + $L_{jk}U_{ki}$
            $U_{ji} = \frac{1}{L_{jj}}(a_{ji} - suma)$

    Paso 2: Encontrar y
    y ← **función** triangular inferior (L, b)

    x ← **función** triangular superior (U, y)
    return L,U, x;

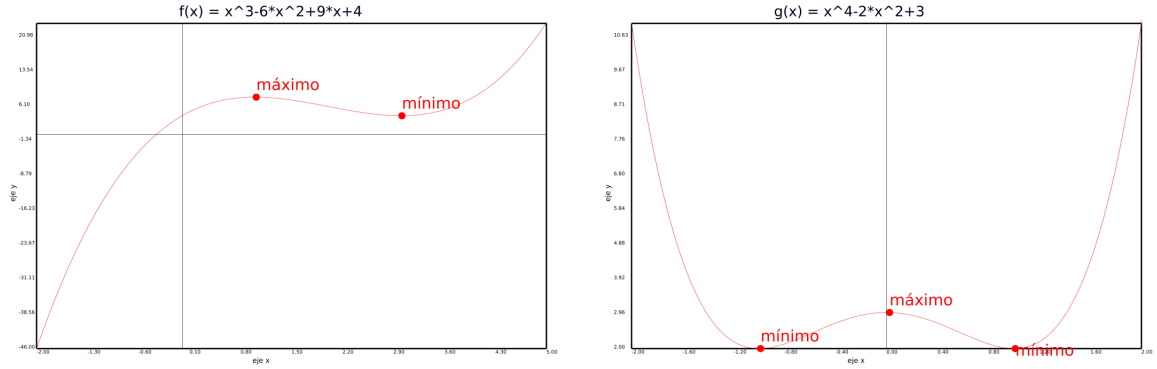
4. Resultados

4.1. Máximos y mínimos

Se hizo una prueba con dos funciones que se deben de escribir en las primeras líneas del programa, las funciones fueron:

$$f(x) = x^3 - 6x^2 + 9x + 4, \tag{25}$$
$$g(x) = x^4 - 2x^2 + 3. \tag{26}$$

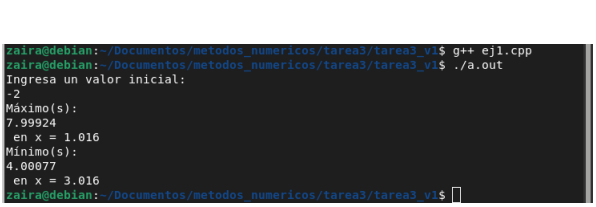
Así, se graficaron con sus máximos y mínimos encontrados por el programa.



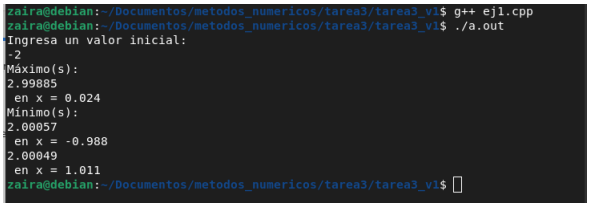
(a) f(x)

(b) g(x)

Figura 1: Gráfica de funciones con máximos y mínimos



(a) f(x)



(b) g(x)

Figura 2: Puntos máximos y mínimos de dos funciones distintas.

4.2. Ax = b

4.2.1. A diagonal

Se utilizó de ejemplo:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 5 \\ 4 \\ 1 \\ 8 \\ 25 \end{pmatrix}$$

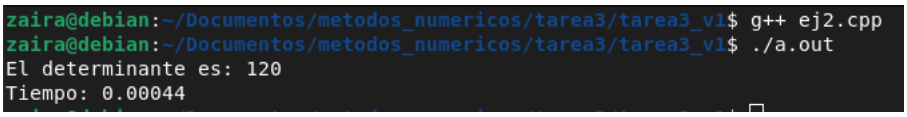


Figura 3: Resultados matriz diagonal

y regresó de respuesta:

$$\begin{pmatrix} 5 \\ 2 \\ 0{,}3333 \\ 2 \\ 5 \end{pmatrix}$$

4.3. A es una matriz diagonal inferior

Se utilizó de ejemplo:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 5 & 0 & 0 \\ 3 & 6 & 8 & 0 \\ 4 & 7 & 9 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \\ -16 \\ -18 \end{pmatrix} \tag{27}$$

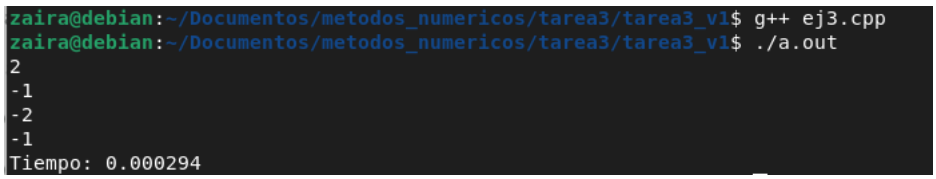


Figura 4: Resultados de matriz diagonal inferior.

4.4. A es una matriz diagonal superior

Se utilizó de ejemplo:

$$\begin{pmatrix} 1 & -2 & 2 & -3 \\ 0 & 10 & -7 & 10 \\ 0 & 0 & 13 & -40 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 15 \\ -51 \\ 79 \\ -1 \end{pmatrix} \tag{28}$$

```
zaira@debian:~/Documentos/metodos_numericos/tarea3/tarea3_v1$ g++ ej4.cpp
zaira@debian:~/Documentos/metodos_numericos/tarea3/tarea3_v1$ ./a.out
2
-2
3
-1
Tiempo: 0.00026
```

Figura 5: Ejecución en consola

4.5. A es una matriz completa

Para este caso, se ejecutó la matriz dada en la tarea de dimensión (5000, 5000), y el tiempo de ejecución fue de 33 minutos y 45 segundos. Además, se muestran algunas de las soluciones de x en la fig. 15.

```
4995
4996
4997
4998
4999
Tiempo: 2007.09
PS D:\OneDrive - Universidad de Guadalajara\Documents\maestria_computacion\primer_semestre\metodos_numericos\tarea3\tare
a3_v1>
```

Figura 6: Método Gauss Jordan aplicado a una matriz A de dimensiones largas.

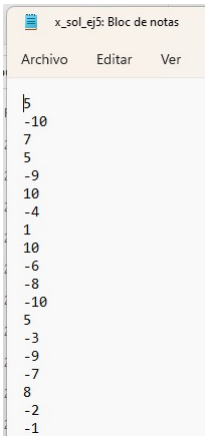
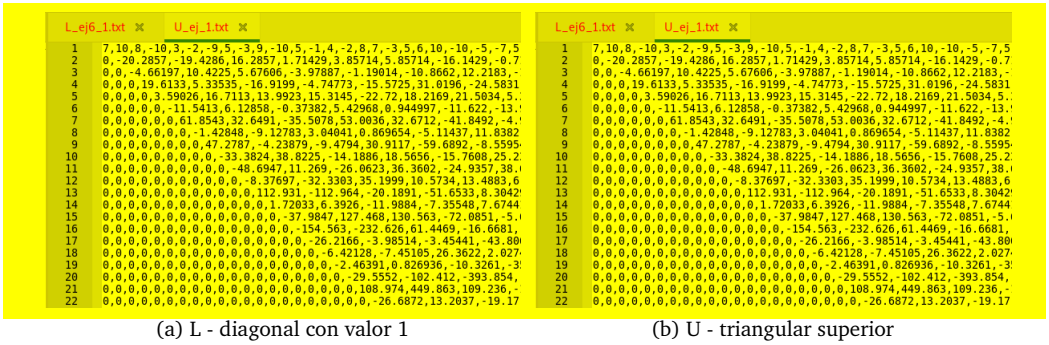


Figura 7: Primeros 20 resultados de las incógnitas.

4.6. Descomposición LU: Crout

Se desarrolló en tres programas separados, porque al ejecutarlo paralelo a la factorización, el tiempo aumentaba. Así, primero, se factorizó la matriz A.



(a) L - diagonal con valor 1 (b) U - triangular superior

Figura 8: Factorización de la matriz A en dos matrices L, U.

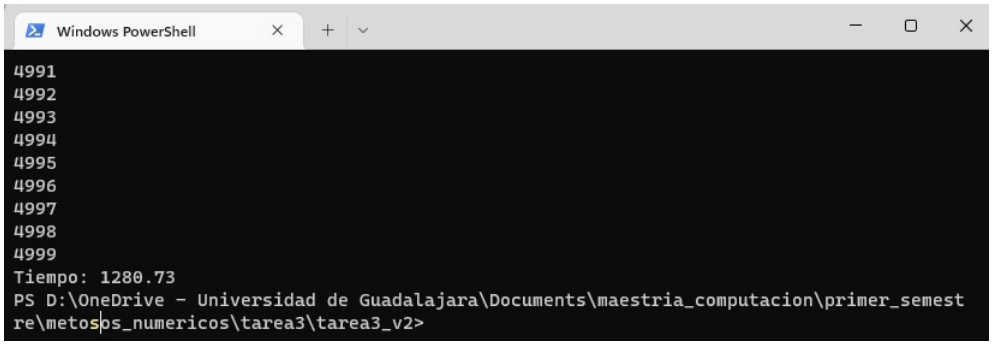


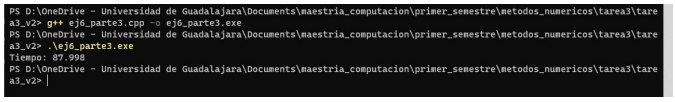
Figura 9: Tiempo de ejecución para factorizar una matriz de 5000 por 5000, en total 22 minutos.

Después, se aplicaron los métodos para factorización triangular inferior (parte2) y superior (parte 3).

Figura 10: Resultados de la variable x utilizando descomposición.

Con los tiempos: Sin embargo, las respuestas no son parecidas a las encontradas con el método Gauss Jordan.

(a) Menos de 2 minutos de ejecución para resolver una matriz triangular inferior

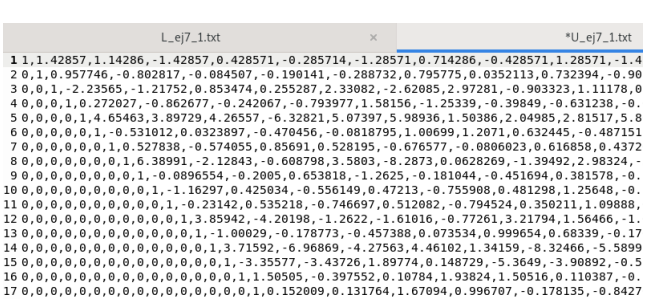


(b) Menos de 2 minutos para encontrar el valor de x dada una matriz triangular superior.

Figura 11: Tiempos de ejecución para pasos siguientes de descomposición LU.

4.7. Facorización LU: Doolittle

De la misma forma que el método Crout, primero se hizo la factorización LU:



(a) L

(b) U - unos en diagonal

Figura 12: Factorización de la matriz A utilizando el método de Doolittle.

Como se observa, los resultados son consistentes con la idea de unos en la diagonal de U, y de matrices triangulares inferiores o superiores. Sin embargo, el tiempo fue alto para su ejecución.

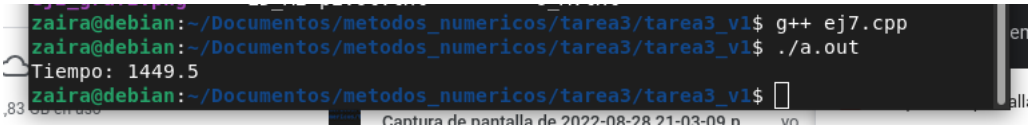


Figura 13: 24 minutos de ejecución para factorizar una matriz con el método de Doolittle (2 minutos más que la factorización con Crout).

5. Conclusiones

Para comprobar los resultados de la matriz LU en el método de Crout, se utilizó una función para calcular su multiplicación y ver si eran parecidos a los de la matriz que estabamos factorizando,

	0	1	2	3	4	5	6	7	8	9	...	4990	4991	4992
0	7.000000	10.000000	8.000000	-10.000000	3.000000	-2.000000	-9.000000	5.000000	-3.000000	9.000000	...	7.000000	-2.000000	9.000000
1	9.999990	-6.000000	-8.000040	2.000000	6.000000	1.000000	-6.999990	-9.000050	-4.999996	-1.999970	...	-1.000010	-1.999997	1.000030
2	5.000002	5.000000	-1.000003	4.999964	8.000005	-4.999997	-7.000001	-9.000009	9.999989	-9.000041	...	-8.999972	8.999971	-0.999999
3	4.000003	-7.999979	-0.999996	8.000008	-1.000020	-8.999956	-4.000001	-5.999971	8.999934	-6.999859	...	3.000045	1.999961	-2.999910
4	6.000001	-3.999983	3.000000	7.999965	3.999961	3.000067	6.000019	9.000032	-8.000198	10.000222	...	-8.999865	7.999897	-2.999878
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4995	-0.999999	9.999988	-4.000006	-9.000135	4.000013	-10.000011	-0.999969	9.000102	-0.999590	-4.000160	...	104.391875	-3.493348	-115.849132
4996	0.999999	-3.000001	4.999986	-2.999923	1.999998	3.000059	-3.999995	3.999886	-5.999983	-9.999899	...	-31.266830	63.278480	58.845844
4997	-7.000000	-1.000006	-8.999964	-2.000180	4.999969	-5.999894	5.000052	-1.999914	3.999819	3.000103	...	108.042831	28.653968	-12.352373
4998	0.999999	4.000006	-6.999964	6.999872	5.000001	-7.999914	-1.999946	3.000180	-4.000090	-8.999999	...	144.708142	-102.060320	-140.502833
4999	-7.000000	1.999986	4.000009	-9.999979	8.999976	3.999969	5.000095	-10.000275	4.999687	2.000160	...	40.868929	-26.276111	-121.030687

5000 rows × 5000 columns

Figura 14: Multiplicación de las matrices obtenidas L y U, utilizando el método de Crout.

	0	1	2	3	4	5	6	7	8	9	...	4990	4991	4992	4993	4994	4995	4996	4997	4998	4999
0	7	10	8	-10	3	-2	-9	5	-3	9	...	7	-2	9	7	-9	-10	-7	2	6	-2
1	10	-6	-8	2	6	1	-7	-9	-5	-2	...	-1	-2	1	-10	1	-5	5	-8	-10	2
2	5	5	-1	5	8	-5	-7	-9	10	-9	...	-9	9	-1	1	3	2	-1	-2	9	8
3	4	-8	-1	8	-1	-9	-4	-6	9	-7	...	3	2	-3	-6	-10	10	-4	-1	-2	-6
4	6	-4	3	8	4	3	6	9	-8	10	...	-9	8	-3	-9	4	-1	10	5	5	-5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4995	-1	10	-4	-9	4	-10	-1	9	-1	-4	...	-6	-7	10	4	-3	9	-9	-6	-2	-10
4996	1	-3	5	-3	2	3	-4	4	-6	-10	...	-9	4	6	9	3	1	4	-9	1	-1
4997	-7	-1	-9	-2	5	-6	5	-2	4	3	...	5	-9	9	1	-3	-2	-4	-7	-3	10
4998	1	4	-7	7	5	-8	-2	3	-4	-9	...	-7	-6	8	-2	-2	7	-9	-3	-10	8
4999	-7	2	4	-10	9	4	5	-10	5	2	...	3	-3	-5	7	-6	6	4	-4	-10	6

Figura 15: Matriz factorizada A.

en esta, se puede ver que no hay mucha diferencia, y que al parecer se mantiene en diferencias mínimas las primeras columnas y las primeras filas de la matriz. Sin embargo, al ejecutar para obtener x, hay un error absoluto muy grande como se mostró en resultados. Esto se debe a la cantidad de operaciones que se hace para obtener los elementos de cada matriz, y conforme avanzan estas operaciones, el error va creciendo. Por ello, es que se intentó hacer un medio pivoteo, se acomodó la matriz de manera que el máximo por columna estuviera en la diagonal principal, sin embargo, no mejoró. El número máximo de cada columna siempre fue 10, el mismo que toda la matriz. También, se intentó dividir los elementos de A y b, para que los resultados no fueran muy grands. Entre las futuras mejoras de los programas mostrados, se espera hacer una librería para

evitar tener todas las funciones en cada código. No obstante, se encontraron tipos de entradas de matrices en el lenguaje C++ que trabajan mejor que otras, por ejemplo, la función lectura de matriz para una matriz completa como fue la que se usó para la eliminación gaussiana, fue distinta a cuando era una matriz triangular. Asimismo, se esperan mejorar los tiempos de ejecución y unir los programas de factorización cuando estos funcionen correctamente.

Referencias

[1] R. Burden, J. Faires, and A. Burden, *Numerical Analysis*. Cengage Learning, 2015. [Online]. Available: https://books.google.com.mx/books?id=9DV-BAAAQBAJ