

# Solución de Sistemas de ecuaciones lineales y factorización LU con Python

Johan Posada y Juan Morales

Mayo 14 2024

## 1 Introducción

La factorización LU descompone una matriz  $A$  en el producto de dos matrices: una matriz  $L$  (Lower) triangular inferior y una matriz  $U$  (Upper) triangular superior. Para la matriz  $L$ , todos los elementos de la diagonal son iguales a uno. Esta factorización sigue la forma  $A = L \cdot U$ , solo se cumple para matrices cuadradas y es bastante usada para solución de sistemas de ecuaciones de la forma  $Ax = b$  o para hallar el determinante:  $|A| = |L| \cdot |U|$ . El Método de reducción de Gauss-Jordan es bastante utilizado para resolver sistemas de ecuaciones de  $n \cdot n$ . El objetivo de este documento es explicar cómo se puede construir un programa en Python que permita resolver sistemas de ecuaciones lineales de única solución y otro que haga la factorización LU de una matriz cuadrada.

## 2 Construyendo el programa para Gauss-Jordan

En un principio el programa se había construido sin usar POO, sin embargo se decidió hacer uso de este paradigma de la programación para mejorar su estructura. El código se puede acceder en el perfil de GitHub del desarrollador del código: <https://github.com/johanP051/Equations-systems.git>

Para empezar, el sistema de ecuaciones lineales debe seguir esta forma:

$$\begin{aligned}a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n\end{aligned}$$

$$\left( \begin{array}{ccc|c} a_{11} & a_{12} & \cdots & b_1 \\ a_{21} & a_{22} & \cdots & b_2 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & b_n \end{array} \right)$$

Donde la última columna es el vector de igualdad. El programa se diseñó para que el usuario

pueda ingresar los valores de los coeficientes y luego el de los términos independientes, luego se encarga de realizar las operaciones necesarias para encontrar los valores de las variables o incógnitas.

Para empezar veamos el siguiente código:

```
1  import numpy as np
2
3  class SistemaEcuaciones:
4      def __init__(self, ecuaciones, variables):
5          self.ecuaciones = ecuaciones
6          self.variables = variables
7
8          # Lista para almacenar las filas de la matriz
9          self.filas = []
10         # Matriz para realizar las operaciones de Gauss-Jordan
11         self.matrizReducir = None
```

Listing 1: Atributos de la clase

Fíjese que se ha creado una clase llamada `SistemaEcuaciones`, después en el método `def __init__` se crean los atributos del objeto: `ecuaciones` y `variables` que son los valores que se solicitarán al usuario. Además se crea una lista llamada `filas` que almacenará los renglones de la matriz y un arreglo llamado `matrizReducir` que se usará para realizar las operaciones de Gauss-Jordan (Por el momento solo se define la variable como `self.matrizReducir`, por eso toma el valor `None`, eso después va a cambiar cuando el usuario digite los datos).

```
1  def recoger_datos(self):
2      for i in range(self.ecuaciones):
3          elementosFila = []
4          print(f"\nPara la ecuacion numero {i + 1}:")
5
6          for j in range(self.variables):
7              Xn = int(input(f"Inserte el valor del coeficiente X{
8                  j + 1}: "))
9              elementosFila.extend([Xn])
10
11         igualdad = int(input(f"A que valor esta igualada la
12             ecuacion?: "))
13         elementosFila.extend([igualdad])
14
15         # Agregar la fila a la lista de filas, cada fila
16         # representa un solo elemento de la lista
17         self.filas.append(elementosFila)
```

Listing 2: Método para recoger los datos

Aquí se crea un método llamado `recoger_datos` que solicita al usuario los valores de los coeficientes de las variables y los términos independientes. Nótese que hay un bucle `for`

anidado, el bucle externo se usa únicamente para decirle al usuario cuál ecuación o fila de la matriz está digitando (i), mientras que el bucle interno se usa para recorrer las columnas (j) solicitando los valores de los coeficientes, una vez termine este bucle se le pide el valor al que está igualada la ecuación y el bucle externo se vuelve a repetir hasta que finalice el rango del número de ecuaciones. Cada elemento de la fila se almacena en una lista llamada `elementosFila` y luego cada una se agrega a la lista `filas` que es una lista de listas.

Para entender bien el siguiente método y el siguiente ejemplo sobre cómo funciona el algoritmo para reducir un sistema de 3x3, una vez se entienda, esto se puede extrapolar a una nxn.

Tomemos la siguiente matriz:

$$\left( \begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array} \right)$$

El objetivo de la simplificación es que los pivotes sean igual a 1. Si simplificamos desde la fila con índice 0 hasta la fila con índice 3, vamos a dividir cada fila teniendo en cuenta el elemento de la columna correspondiente, obtendremos lo siguiente:

$$\left( \begin{array}{cccc} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array} \right) \begin{array}{l} F_1/a_{11} \rightarrow F_1 \\ F_2/a_{21} \rightarrow F_2 \\ F_3/a_{31} \rightarrow F_3 \end{array}$$

Una vez hecho esto, se tiene una matriz simplificada que se puede reducir más fácil::

$$\left( \begin{array}{cccc|l} 1 & a_{12}/a_{11} & a_{13}/a_{11} & b_1/a_{11} & \\ 1 & a_{22}/a_{21} & a_{23}/a_{21} & b_2/a_{21} & F_2 - F_1 \rightarrow F_2 \\ 1 & a_{32}/a_{31} & a_{33}/a_{31} & b_3/a_{31} & F_3 - F_1 \rightarrow F_3 \end{array} \right)$$

Reduciendo teniendo como pivote la fila 1 (índice 0):

$$\left( \begin{array}{cccc|c} 1 & a_{12}/a_{11} & a_{13}/a_{11} & b_1/a_{11} & \\ 0 & (a_{22}/a_{21}) - (a_{12}/a_{11}) & (a_{23}/a_{21}) - (a_{13}/a_{11}) & (b_2/a_{21}) - (b_1/a_{11}) & \\ 0 & (a_{32}/a_{31}) - (a_{12}/a_{11}) & (a_{33}/a_{31}) - (a_{13}/a_{11}) & (b_3/a_{31}) - (b_1/a_{11}) & \end{array} \right)$$

Ahora se debe volver a simplificar la matriz para que el pivote de la fila 2 (índice 1) sea igual a 1 y luego reducir las filas restantes.

$$\left( \begin{array}{cccc|c} 1 & a_{12}/a_{11} & a_{13}/a_{11} & b_1/a_{11} & \\ 0 & (a_{22}/a_{21}) - (a_{12}/a_{11}) & (a_{23}/a_{21}) - (a_{13}/a_{11}) & (b_2/a_{21}) - (b_1/a_{11}) & F_2/(a_{22}/a_{21}) - (a_{12}/a_{11}) \rightarrow F_2 \\ 0 & (a_{32}/a_{31}) - (a_{12}/a_{11}) & (a_{33}/a_{31}) - (a_{13}/a_{11}) & (b_3/a_{31}) - (b_1/a_{11}) & F_3/(a_{32}/a_{31}) - (a_{12}/a_{11}) \rightarrow F_3 \end{array} \right)$$

La matriz simplificada se puede reducir de nuevo teniendo como pivote la fila 2 (índice 1):

$$\left( \begin{array}{ccc|c} 1 & a_{12}/a_{11} & a_{13}/a_{11} & b_1/a_{11} \\ 0 & 1 & [(a_{23}/a_{21}) - (a_{13}/a_{11})]/[(a_{22}/a_{21}) - (a_{12}/a_{11})] & \cdots \\ 0 & 1 & [(a_{33}/a_{31}) - (a_{13}/a_{11})]/[(a_{32}/a_{31}) - (a_{12}/a_{11})] & \cdots \end{array} \right) F_3 - F_2 \rightarrow F_3$$

$$\left( \begin{array}{ccc|c} 1 & a_{12}/a_{11} & a_{13}/a_{11} & b_1/a_{11} \\ 0 & 1 & [(a_{23}/a_{21}) - (a_{13}/a_{11})]/[(a_{22}/a_{21}) - (a_{12}/a_{11})] & \cdots \\ 0 & 0 & \cdots & \cdots \end{array} \right) F_3 - F_2 \rightarrow F_3$$

Después se simplifica la fila 3 para lograr una matriz triangular superior con unos en la diagonal principal y ceros debajo de la diagonal principal. De igual manera el algoritmo revisa si hay más filas restantes para reducir, en este caso no hay más filas restantes, por lo que la primera parte del proceso termina.

Si notamos bien, el proceso se hace por cada fila y termina cuando las columnas de la matriz de coeficientes se acaba:

- En el paso 1 se simplificó desde la fila 1 hasta la fila 3 y se redujo desde la fila 2 hasta la fila 3.
- En el paso 2 se simplificó desde la fila 2 hasta la fila 3 y se redujo solamente la fila 3.
- En el paso 3 se simplificó solamente la fila 3 y la reducción no se hace, pues en este caso la fila a reducir sería la número 4 y como no existe, entonces el bucle for no se ejecuta.

```

1  def gauss_jordan(self):
2      inicioFilaReduccion = 0
3      finalFilaReduccion = self.ecuaciones
4      filaPivote = -1
5
6      columnas = self.variables
7
8      while columnas >= 1:
9          inicioFilaSimplificacion += 1
10         finalFS = finalFilaSimplificacion
11         elementoFilaSimplificacion += 1
12         #print(inicioFilaSimplificacion, finalFS,
13               elementoFilaSimplificacion)
14         # Simplificar la fila actual dividiendo por el elemento
15         de la columna correspondiente
16         for filaS in range(inicioFilaSimplificacion, finalFS):
17             filaSimplificada = self.matrizReducir[filaS]
```

```

16         if filaSimplificada[elementoFilaSimplificacion] !=
17             0:
18             filaSimplificada = filaSimplificada /
19                 filaSimplificada[elementoFilaSimplificacion]
20             self.matrizReducir[filaS] = filaSimplificada
21
22         print(f"\nResultado de la simplificacion: ")
23         print(self.matrizReducir)
24
25         inicioFilaReduccion += 1
26         finalFR = finalFilaReduccion
27         filaPivote += 1
28         #print(inicioFilaReduccion, finalFR, filaPivote)
29         # Reducir las filas restantes restando la fila pivote
30         # multiplicada por el elemento correspondiente
31         for filaR in range(inicioFilaReduccion, finalFR):
32             filaReducida = self.matrizReducir[filaR]
33             if filaReducida[filaPivote] != 0:
34                 filaReducida = filaReducida - self.matrizReducir
35                     [filaPivote]
36             self.matrizReducir[filaR] = filaReducida
37
38         print(f"\nResultado de la Reduccion: \n{self.
39             matrizReducir}")
40
41         columnas -= 1
42         print("\nSegunda parte del Gauss Jordan:")
43         self.gauss_jordan_segunda_parte()

```

Listing 3: Método para resolver el sistema de ecuaciones

Como se explicó antes, en el método `gauss_jordan` se inicializan las variables que se usarán para simplificar y reducir la matriz mientras que el número de variables sea mayor o igual a 1. Además, se le a agreado una condición que le indica al programa que si el elemento de la fila pivote es igual a 0, no se realice la operación de reducción. Ni tampoco de simplificación, ya que se encontraría con una división por cero.

```

1     def gauss_jordan_segunda_parte(self):
2         inicioFilaSimplificacion = 0
3         finalFilaSimplificacion = self.ecuaciones
4         elementoFilaSimplificacion = self.variables
5
6         inicioFilaReduccion = 0
7         finalFilaReduccion = self.ecuaciones
8         filaPivote = self.ecuaciones
9
10        columnas = self.variables
11

```

```

12     while columnas >= 1:
13         inicioFS = inicioFilaSimplificacion
14         finalFilaSimplificacion -= 1
15         elementoFilaSimplificacion -= 1
16
17         # Simplificar la fila actual dividiendo por el elemento
18         # de la columna correspondiente
19         for filaS in reversed(range(inicioFS,
20             finalFilaSimplificacion + 1)):
21             filaSimplificada = self.matrizReducir[filaS]
22             if filaSimplificada[elementoFilaSimplificacion] !=
23                 0:
24                 filaSimplificada = filaSimplificada /
25                     filaSimplificada[elementoFilaSimplificacion]
26                 self.matrizReducir[filaS] = filaSimplificada
27
28         print(f"\nResultado de la simplificacion: ")
29         print(self.matrizReducir)
30
31         inicioFR = inicioFilaReduccion
32         finalFilaReduccion -= 1
33         filaPivote -= 1
34
35         # Reducir las filas restantes restando la fila pivote
36         # multiplicada por el elemento correspondiente
37         for filaR in reversed(range(inicioFR, finalFilaReduccion
38             )):
39             filaReducida = self.matrizReducir[filaR]
40             filaReducida = filaReducida - self.matrizReducir[
41                 filaPivote]
42             self.matrizReducir[filaR] = filaReducida
43
44         print(f"\nResultado de la Reduccion: \n{self.
45             matrizReducir}")
46         columnas -= 1

```

Listing 4: Método para resolver el sistema de ecuaciones

En el método `gauss_jordan_segunda_parte` se realiza el mismo proceso que en el método `gauss_jordan` pero en sentido contrario, es decir, se simplifica y reduce desde la última fila hasta la primera. Python en la función `range()` crea una lista iterable, desde la primera fila hasta la última, sin embargo, necesitamos iterar desde la última fila hasta la primera, por eso se usa la función `reversed()` que invierte el orden de la lista.

```

1     # Solicitar el numero de ecuaciones y variables al usuario
2     ecuaciones = int(input("Inserte el numero de ecuaciones: "))
3     variables = int(input("Inserte el numero de variables: "))
4

```

```

5      # Crear una instancia de la clase SistemaEcuaciones y resolver
      el sistema
6      sistema = SistemaEcuaciones(ecuaciones, variables)
7      sistema.recoger_datos()
8      sistema.resolver()

```

Listing 5: Método para resolver el sistema de ecuaciones

Por último se crea el objeto sistema, haciendo una instancia a la clase `SistemaEcuaciones` y se llama al método `recoger_datos` para que el usuario pueda ingresar los valores de las ecuaciones y variables.

## Probando el código

Para probar el código, se van a usar los ejercicios de la página 23 del libro algebra lineal de Stanley I. Grossman, 6ta edición:

$$x_1 - 2x_2 + 3x_3 = 11$$

$$4x_1 + x_2 - x_3 = 4$$

$$2x_1 - x_2 + 3x_3 = 10$$

$$-2x_1 + x_2 + 6x_3 = 18$$

$$5x_1 + 0x_2 + 8x_3 = -16$$

$$3x_1 + 2x_2 - 10x_3 = -3$$

```

Inserte el número de ecuaciones: 3
Inserte el número de variables: 3

Para la ecuación número 1:
Inserte el valor del coeficiente X1: 1
Inserte el valor del coeficiente X2: -2
Inserte el valor del coeficiente X3: 3
¿A qué valor está igualada la ecuación?: 11

Para la ecuación número 2:
Inserte el valor del coeficiente X1: 4
Inserte el valor del coeficiente X2: 1
Inserte el valor del coeficiente X3: -1
¿A qué valor está igualada la ecuación?: 4

Para la ecuación número 3:
Inserte el valor del coeficiente X1: 2
Inserte el valor del coeficiente X2: -1
Inserte el valor del coeficiente X3: 3
¿A qué valor está igualada la ecuación?: 10

La matriz aumentada es:

[[ 1. -2.  3. 11.]
 [ 4.  1. -1.  4.]
 [ 2. -1.  3. 10.]]

```

Figure 1: Solicitando datos y construyendo la matriz aumentada

Primera parte del Gauss-Jordan

Resultado de la simplificación:

```
[[ 1.  -2.   3.  11. ]
 [ 1.   0.25 -0.25  1. ]
 [ 1.  -0.5  1.5  5. ]]
```

Resultado de la Reducción:

```
[[ 1.  -2.   3.  11. ]
 [ 0.   2.25 -3.25 -10. ]
 [ 0.   1.5  -1.5  -6. ]]
```

Resultado de la simplificación:

```
[[ 1.  -2.   3.  11. ]
 [ 0.   1.  -1.44444444 -4.44444444]
 [ 0.   1.  -1.  -4. ]]
```

Resultado de la Reducción:

```
[[ 1.  -2.   3.  11. ]
 [ 0.   1.  -1.44444444 -4.44444444]
 [ 0.   0.   0.44444444  0.44444444]]
```

Resultado de la simplificación:

```
[[ 1.  -2.   3.  11. ]
 [ 0.   1.  -1.44444444 -4.44444444]
 [ 0.   0.   1.   1. ]]
```

Resultado de la Reducción:

```
[[ 1.  -2.   3.  11. ]
 [ 0.   1.  -1.44444444 -4.44444444]
 [ 0.   0.   1.   1. ]]
```

Figure 2: Gauss-Jordan primera parte ejemplo 1



Segunda parte del Gauss Jordan:

Resultado de la simplificación:

```
[[ 0.33333333 -0.66666667 1.          3.66666667]
 [-0.         -0.69230769 1.          3.07692308]
 [ 0.          0.          1.          1.          ]]
```

Resultado de la Reducción:

```
[[ 0.33333333 -0.66666667 0.          2.66666667]
 [-0.         -0.69230769 0.          2.07692308]
 [ 0.          0.          1.          1.          ]]
```

Resultado de la simplificación:

```
[[-0.5  1.  -0.  -4. ]
 [ 0.   1.  -0.  -3. ]
 [ 0.   0.   1.   1. ]]
```

Resultado de la Reducción:

```
[[-0.5  0.   0.  -1. ]
 [ 0.   1.  -0.  -3. ]
 [ 0.   0.   1.   1. ]]
```

Resultado de la simplificación:

```
[[ 1. -0. -0.  2.]
 [ 0.  1. -0. -3.]
 [ 0.  0.  1.  1.]]
```

Resultado de la Reducción:

```
[[ 1. -0. -0.  2.]
 [ 0.  1. -0. -3.]
 [ 0.  0.  1.  1.]]
```

Figure 3: Gauss-Jordan segunda parte ejemplo 1

```
Inserte el número de ecuaciones: 3
Inserte el número de variables: 3

Para la ecuación número 1:
Inserte el valor del coeficiente X1: -2
Inserte el valor del coeficiente X2: 1
Inserte el valor del coeficiente X3: 6
¿A qué valor está igualada la ecuación?: 18

Para la ecuación número 2:
Inserte el valor del coeficiente X1: 5
Inserte el valor del coeficiente X2: 0
Inserte el valor del coeficiente X3: 8
¿A qué valor está igualada la ecuación?: -16

Para la ecuación número 3:
Inserte el valor del coeficiente X1: 3
Inserte el valor del coeficiente X2: 2
Inserte el valor del coeficiente X3: -10
¿A qué valor está igualada la ecuación?: -3

La matriz aumentada es:

[[ -2.   1.   6.  18.]
 [  5.   0.   8. -16.]
 [  3.   2. -10.  -3.]]
```

Figure 4: Solicitando datos y construyendo la matriz aumentada ejemplo 2

```

Primera parte del Gauss-Jordan

Resultado de la simplificación:
[[ 1.      -0.5      -3.      -9.      ]
 [ 1.       0.       1.6     -3.2     ]
 [ 1.      0.66666667 -3.33333333 -1.      ]]

Resultado de la Reducción:
[[ 1.      -0.5      -3.      -9.      ]
 [ 0.       0.5      4.6      5.8      ]
 [ 0.      1.16666667 -0.33333333 8.      ]]

Resultado de la simplificación:
[[ 1.      -0.5      -3.      -9.      ]
 [ 0.       1.       9.2      11.6     ]
 [ 0.       1.      -0.28571429 6.85714286]]

Resultado de la Reducción:
[[ 1.      -0.5      -3.      -9.      ]
 [ 0.       1.       9.2      11.6     ]
 [ 0.       0.      -9.48571429 -4.74285714]]

Resultado de la simplificación:
[[ 1.  -0.5 -3.  -9. ]
 [ 0.   1.  9.2 11.6]
 [-0.  -0.   1.  0.5]]

Resultado de la Reducción:
[[ 1.  -0.5 -3.  -9. ]
 [ 0.   1.  9.2 11.6]
 [-0.  -0.   1.  0.5]]

```

Figure 5: Gauss-Jordan primera parte ejemplo 2

Segunda parte del Gauss Jordan:

Resultado de la simplificación:

```
[[ -0.33333333  0.16666667  1.          3.          ]
 [  0.          0.10869565  1.          1.26086957]
 [ -0.          -0.          1.          0.5          ]]
```

Resultado de la Reducción:

```
[[ -0.33333333  0.16666667  0.          2.5          ]
 [  0.          0.10869565  0.          0.76086957]
 [ -0.          -0.          1.          0.5          ]]
```

Resultado de la simplificación:

```
[[ -2.   1.   0.  15. ]
 [  0.   1.   0.   7. ]
 [ -0.  -0.   1.   0.5 ]]
```

Resultado de la Reducción:

```
[[ -2.   0.   0.   8. ]
 [  0.   1.   0.   7. ]
 [ -0.  -0.   1.   0.5 ]]
```

Resultado de la simplificación:

```
[[ 1.  -0.  -0.  -4. ]
 [ 0.   1.   0.   7. ]
 [-0.  -0.   1.   0.5 ]]
```

Resultado de la Reducción:

```
[[ 1.  -0.  -0.  -4. ]
 [ 0.   1.   0.   7. ]
 [-0.  -0.   1.   0.5 ]]
```

Figure 6: Gauss-Jordan segunda parte ejemplo 2

### 3 Pasos para factorizar por LU

1. Realizamos el proceso de eliminación de Gauss para convertir la matriz en triangular superior, mientras mantenemos un registro de las operaciones realizadas. Por ejemplo, si tenemos la matriz:

$$\begin{pmatrix} 1 & 7 & 6 \\ -4 & 3 & 2 \\ 3 & -2 & 1 \end{pmatrix} \quad \begin{array}{l} F_2 + 4F_1 \rightarrow F_2 \\ F_3 - 3F_1 \rightarrow F_3 \end{array}$$
$$\begin{pmatrix} 1 & 7 & 6 \\ 0 & 31 & 26 \\ 0 & -23 & -17 \end{pmatrix} \quad F_3 + \frac{23}{31}F_2 \rightarrow F_3$$
$$U = \begin{pmatrix} 1 & 7 & 6 \\ 0 & 31 & 26 \\ 0 & 0 & \frac{71}{31} \end{pmatrix}$$

2.  $U$  es la matriz triangular superior resultante de la eliminación de Gauss.
3. Si tomamos los escalares que usamos para realizar la eliminación de Gauss y les cambiamos el signo, obtendremos una matriz que al sumarla con la identidad da como resultado la matriz inferior  $L$ .

$$L = \begin{pmatrix} 0 & 0 & 0 \\ -4 & 0 & 0 \\ 3 & \frac{-23}{31} & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ 3 & \frac{-23}{31} & 1 \end{pmatrix}$$

4. Por último se verifica que  $L \cdot U = A$ .

$$A = \begin{pmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ 3 & \frac{-23}{31} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 7 & 6 \\ 0 & 31 & 26 \\ 0 & 0 & \frac{71}{31} \end{pmatrix} = \begin{pmatrix} 1 & 7 & 6 \\ -4 & 3 & 2 \\ 3 & -2 & 1 \end{pmatrix}$$

### Construyendo el programa para Factorización LU

```
1 import numpy as np
2
3 dim_A = int(input("Ingrese la dimensi n de la matriz cuadrada A
4 : "))
5 A = np.zeros((dim_A, dim_A))
6 for i in range(dim_A):
7     for j in range(dim_A):
8         A[i, j] = float(input(f"Ingrese el elemento A[{i + 1}, {
9 j + 1}]: "))
```

```
9 print(f"\nLa matriz A es:\n{A}")
```

Listing 6: Construyendo la Matriz A

Aquí se solicita al usuario que ingrese la dimensión de la matriz cuadrada y luego los elementos de la matriz A,

para esto se usa un bucle for anidado que recorre las filas (i) y columnas (j) de la matriz.

```
1 # Factorizacion LU
2 L = np.eye(dim_A)
3 U = np.zeros((dim_A, dim_A))
4 haz el tama o de cada imagen mas grande para que ocupen una
   pagina completa_A):
5     L[i, k] = (A[i, k] e- np.dot(L[i, :k], U[:k, k])) / U[k,
      k]
6     for j in range(k + 1, dim_A):
7         U[k, j] = A[k, j] - np.dot(L[k, :k], U[:k, j])
8
9 print(f"\nLa matriz L es:\n{L}")
10 print(f"\nLa matriz U es:\n{U}")
```

Listing 7: Factorizando

Se inicializan las matrices L y U con dimensiones `dim_a` mediante las funciones `np.eye()` y `np.zeros()`, respectivamente. La primera función genera una matriz con unos en la diagonal y ceros en el resto de las posiciones, mientras que la segunda crea una matriz de ceros con las dimensiones especificadas.

Luego, se ejecuta un bucle for que itera sobre los índices de la matriz A. En cada iteración, se calculan los elementos de la matriz U utilizando la fórmula de la factorización LU. El elemento  $U[k, k]$  se calcula restando el producto punto de la fila k de L y la columna k de U a  $A[k, k]$ . Después, se calculan los elementos de la columna k de L dividiendo la diferencia entre la columna k de A y el producto punto de la fila i de L y la columna k de U por  $U[k, k]$ . Finalmente, se calculan los elementos de la fila k de U restando el producto punto de la fila k de L y las columnas anteriores de U a la fila k de A.

Una vez completados los cálculos, se imprime la matriz L y la matriz U.

## Probando el código

Para probar el código se va a usar la página 135 del libro algebra lineal de Stanley I. Grossman, 6ta edición:

$$\begin{pmatrix} 2 & 3 & 2 & 4 \\ 4 & 10 & -4 & 0 \\ -3 & -2 & -5 & -2 \\ -2 & 4 & 4 & -7 \end{pmatrix} \quad \begin{pmatrix} -1 & 2 & 3 \\ 2 & 1 & 7 \\ 1 & 3 & 10 \end{pmatrix}$$

```
Ingrese la dimensión de la matriz cuadrada A: 4
Ingrese el elemento A[1, 1]: 2
Ingrese el elemento A[1, 2]: 3
Ingrese el elemento A[1, 3]: 2
Ingrese el elemento A[1, 4]: 4
Ingrese el elemento A[2, 1]: 4
Ingrese el elemento A[2, 2]: 10
Ingrese el elemento A[2, 3]: -4
Ingrese el elemento A[2, 4]: 0
Ingrese el elemento A[3, 1]: -3
Ingrese el elemento A[3, 2]: -2
Ingrese el elemento A[3, 3]: -5
Ingrese el elemento A[3, 4]: -2
Ingrese el elemento A[4, 1]: -2
Ingrese el elemento A[4, 2]: 4
Ingrese el elemento A[4, 3]: 4
Ingrese el elemento A[4, 4]: -7
```

La matriz A es:

```
[[ 2.  3.  2.  4.]
 [ 4. 10. -4.  0.]
 [-3. -2. -5. -2.]
 [-2.  4.  4. -7.]]
```

La matriz L es:

```
[[ 1.         0.         0.         0.         ]
 [ 2.         1.         0.         0.         ]
 [-1.5        0.625      1.         0.         ]
 [-1.         1.75      6.66666667  1.         ]]
```

La matriz U es:

```
[[ 2.  3.  2.  4.]
 [ 0.  4. -8. -8.]
 [ 0.  0.  3.  9.]
 [ 0.  0.  0. -49.]]
```

Figure 7: Factorización LU primer ejemplo

```
Ingrese la dimensi3n de la matriz cuadrada A: 3
Ingrese el elemento A[1, 1]: -1
Ingrese el elemento A[1, 2]: 2
Ingrese el elemento A[1, 3]: 3
Ingrese el elemento A[2, 1]: 2
Ingrese el elemento A[2, 2]: 1
Ingrese el elemento A[2, 3]: 7
Ingrese el elemento A[3, 1]: 1
Ingrese el elemento A[3, 2]: 3
Ingrese el elemento A[3, 3]: 10
```

La matriz A es:

```
[[ -1.  2.  3.]
 [  2.  1.  7.]
 [  1.  3. 10.]]
```

La matriz L es:

```
[[ 1.  0.  0.]
 [-2.  1.  0.]
 [-1.  1.  1.]]
```

La matriz U es:

```
[[ -1.  2.  3.]
 [  0.  5. 13.]
 [  0.  0.  0.]]
```

Figure 8: Factorizaci3n LU segundo ejemplo



## 4 Conclusión

- En este documento se explicó de manera detallada el algoritmo de Gauss-Jordan, para la factorización LU se intentó usar este mismo, sin embargo el código era bastante extenso, por lo que se construyó desde cero.
- Para mejorar el programa se podría reducir la cantidad de líneas de código y no realizar tantas simplificaciones, pues se puede perder exactitud al realizar divisiones con tipos de datos de punto flotante.

## References

- [1] Torres Solís, M., Villalobos Castillo, N. (s.f). Factorización LU. Universidad del Bío-Bío. Recuperado de [http://repobib.ubiobio.cl/jspui/bitstream/123456789/1811/1/Torres\\_Solis\\_Marcos.pdf](http://repobib.ubiobio.cl/jspui/bitstream/123456789/1811/1/Torres_Solis_Marcos.pdf)