Tarea 10: DESCOMPOSICIÓN LU

Pasquel Johann

Tabla de Contenidos

GITHUB **1**

CONJUNTO DE EJERCICIOS **1**

1. Realice las siguientes multiplicaciones matriz-matriz: 1
2. Determine cuáles de las siguientes matrices son no singulares y calcule la inversa de esas matrices: 4
3. Resuelva los sistemas lineales 4 x 4 que tienen la misma matriz de coeficientes: . . 7
4. Encuentre los valores de A que hacen que la siguiente matriz sea singular. 8
5. Resuelva los siguientes sistemas lineales: 9
6. Factorice las siguientes matrices en la descomposición LU mediante el algoritmo de factorización LU con $l_{ii} = 1$ para todas las i. 11
7. Modifique el algoritmo de eliminación gaussiana de tal forma que se pueda utilizar para resolver un sistema lineal usando la descomposición LU y, a continuación, resuelva los siguientes sistemas lineales. 15

GITHUB

https://github.com/Vladimirjon/MetodosNumericos_PasquelJohann/tree/main/Tarea10

CONJUNTO DE EJERCICIOS

1. Realice las siguientes multiplicaciones matriz-matriz:

```python
def multiply_matrices(matrix1, matrix2):
    # Get the number of rows and columns for both matrices
    rows_matrix1 = len(matrix1)
    cols_matrix1 = len(matrix1[0])
    rows_matrix2 = len(matrix2)
    cols_matrix2 = len(matrix2[0])

    # Check if the matrices can be multiplied
    if cols_matrix1 != rows_matrix2:
        raise ValueError("Ingrese matrices que se puedan multiplicar")

    # Initialize the result matrix with zeros
    result = [[0 for _ in range(cols_matrix2)] for _ in range(rows_matrix1)]

    # Perform matrix multiplication
    for i in range(rows_matrix1):
        for j in range(cols_matrix2):
            for k in range(cols_matrix1):
                result[i][j] += matrix1[i][k] * matrix2[k][j]

    return result
```

a.

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

```python
matriz1 = [
    [2, -3],
    [3, -1]
]
matriz2 = [
    [1, 5],
    [2, 0]
]
resultado = multiply_matrices(matriz1, matriz2)
for row in resultado:
    print(row)
```

```
[-4, 10]
[1, 15]
```

b.

$$\begin{bmatrix} 2 & -3 \\ 3 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 5 & -4 \\ -3 & 2 & 0 \end{bmatrix}$$

```
matriz1 = [
    [2, -3],
    [3, -1]
]
matriz2 = [
    [1, 5, -4],
    [-3, 2, 0]
]
resultado = multiply_matrices(matriz1, matriz2)
for row in resultado:
    print(row)
```

```
[11, 4, -8]
[6, 13, -12]
```

c.

$$\begin{bmatrix} 2 & -3 & 1 \\ 4 & 3 & 0 \\ 5 & 2 & -4 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & -2 \\ 1 & 0 & -1 \\ 2 & 3 & -2 \end{bmatrix}$$

```
matriz1 = [
    [2, -3, 1],
    [4, 3, 0],
    [5, 2, -4]
]
matriz2 = [
    [0, 1, -2],
    [1, 0, -1],
    [2, 3, -2]
]
resultado = multiply_matrices(matriz1, matriz2)
for row in resultado:
    print(row)
```

```
[-1, 5, -3]
[3, 4, -11]
[-6, -7, -4]
```

d.

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

```
matriz1 = [
    [2, 1, 2],
    [-2, 3, 0],
    [2, -1, 3]
]
matriz2 = [
    [1, -2],
    [-4, 1],
    [0, 2]
]
resultado = multiply_matrices(matriz1, matriz2)
for row in resultado:
    print(row)
```

```
[-2, 1]
[-14, 7]
[6, 1]
```

2. Determine cuáles de las siguientes matrices son no singulares y calcule la inversa de esas matrices:

```
import numpy

def calcular_determinante_inversa(matriz):
    determinante = np.linalg.det(matriz)

    if np.isclose(determinante, 0):
        return determinante, None
    else:
        inversa = np.linalg.inv(matriz)
        return determinante, inversa

def resultado(matriz):
    determinante, inversa = calcular_determinante_inversa(matriz)
    print(f"Determinante: {determinante}")
    if inversa is not None:
```

```python
        print("Inversa:")
        print(inversa)
    else:
        print("La matriz no tiene inversa porque su determinante es 0.")
```

a.

$$\begin{bmatrix} 4 & 2 & 6 \\ 3 & 0 & 7 \\ -2 & -1 & -3 \end{bmatrix}$$

```python
matriz = np.array([
    [4, 2, 6],
    [3, 0, 7],
    [-2, -1, -3]
])
resultado(matriz)
```

```
Determinante: 0.0
La matriz no tiene inversa porque su determinante es 0.
```

Matriz singular

b.

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

```python
matriz = np.array([
    [1, 2, 0],
    [2, 1, -1],
    [3, 1, 1]
])
resultado(matriz)
```

```
Determinante: -8.000000000000002
Inversa:
[[-0.25   0.25   0.25 ]
 [ 0.625 -0.125 -0.125]
 [ 0.125 -0.625  0.375]]
```

Matriz no singular

c.

$$\begin{bmatrix} 1 & 1 & -1 & 1 \\ 1 & 2 & -4 & -2 \\ 2 & 1 & 1 & 5 \\ -1 & 0 & -2 & -4 \end{bmatrix}$$

```
matriz = np.array([
    [1, 1, -1, 1],
    [1, 2, -4, -2],
    [2, 1, 1, 5],
    [-1, 0, -2, -4]
])
resultado(matriz)
```

```
Determinante: 0.0
La matriz no tiene inversa porque su determinante es 0.
```

Matriz singular

d.

$$\begin{bmatrix} 4 & 0 & 0 & 0 \\ 6 & 7 & 0 & 0 \\ 9 & 11 & 1 & 0 \\ 5 & 4 & 1 & 1 \end{bmatrix}$$

```
matriz = np.array([
    [4, 0, 0, 0],
    [6, 7, 0, 0],
    [9, 11, 1, 0],
    [5, 4, 1, 1]
])
resultado(matriz)
```

```
Determinante: 28.00000000000001
Inversa:
[[ 0.25        0.          0.          0.        ]
 [-0.21428571  0.14285714 -0.         -0.        ]
 [ 0.10714286 -1.57142857  1.         -0.        ]
 [-0.5         1.         -1.          1.        ]]
```

Matriz no singular

3. Resuelva los sistemas lineales 4 x 4 que tienen la misma matriz de coeficientes:

$$
\begin{aligned}
x_1 - x_2 + 2x_3 - x_4 &= 6, & x_1 - x_2 + 2x_3 - x_4 &= 1,\\
x_1 \quad - x_3 + x_4 &= 4, & x_1 \quad - x_3 + x_4 &= 1,\\
2x_1 + x_2 + 3x_3 - 4x_4 &= -2, & 2x_1 + x_2 + 3x_3 - 4x_4 &= 2,\\
-x_2 + x_3 - x_4 &= 5; & -x_2 + x_3 - x_4 &= -1;
\end{aligned}
$$

```
import numpy as np
import scipy.linalg

A = np.array([
    [1, -1, 2, -1],
    [1, 0, -1, 1],
    [2, 1, 3, -4],
    [0, -1, 1, -1]
], dtype=float)


b1 = np.array([6, 4, -2, 5], dtype=float)  # Primer sistema
b2 = np.array([1, 1, 2, -1], dtype=float)  # Segundo sistema

# LU
P, L, U = scipy.linalg.lu(A)

# Resolver Ly = Pb (sustitución progresiva)
y1 = scipy.linalg.solve(L, np.dot(P, b1))
y2 = scipy.linalg.solve(L, np.dot(P, b2))

# Resolver Ux = y (sustitución regresiva)
x1 = scipy.linalg.solve(U, y1)
x2 = scipy.linalg.solve(U, y2)
```

Soluciones:

```
print("Solución para el primer sistema:")
for i, x in enumerate(x1, start=1):
    print(f"x_{i} = {x:.2f}")

print("\nSolución para el segundo sistema:")
for i, x in enumerate(x2, start=1):
    print(f"x_{i} = {x:.2f}")
```

```
Solución para el primer sistema:
x_1 = 4.00
x_2 = -7.00
x_3 = -11.00
x_4 = -9.00

Solución para el segundo sistema:
x_1 = 1.00
x_2 = 1.00
x_3 = 2.00
x_4 = 2.00
```

4. Encuentre los valores de A que hacen que la siguiente matriz sea singular.

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

$$\det(A) = 1 \begin{vmatrix} 2 & 1 \\ \alpha & -\frac{3}{2} \end{vmatrix} - (-1) \begin{vmatrix} 2 & 1 \\ 0 & -\frac{3}{2} \end{vmatrix} + \alpha \begin{vmatrix} 2 & 2 \\ 0 & \alpha \end{vmatrix}$$

$$\begin{vmatrix} 2 & 1 \\ \alpha & -\frac{3}{2} \end{vmatrix} = (2)(-\frac{3}{2}) - (1)(\alpha) = -3 - \alpha$$

$$\begin{vmatrix} 2 & 1 \\ 0 & -\frac{3}{2} \end{vmatrix} = (2)(-\frac{3}{2}) - (1)(0) = -3$$

$$\begin{vmatrix} 2 & 2 \\ 0 & \alpha \end{vmatrix} = (2)(\alpha) - (2)(0) = 2\alpha$$

$$\det(A) = -3 - \alpha - 3 + 2\alpha^2$$

$$\det(A) = 2\alpha^2 - \alpha - 6$$

Matriz singular si determinante es igual a cero, por lo tanto:

$$\det(A) = 0$$

$2\alpha^2 - \alpha - 6 = 0$

$(2\alpha + 3)(\alpha - 2) = 0$

$2\alpha + 3 = 0$

$\alpha = -\frac{3}{2}$

- $\alpha - 2 = 0$

  $\alpha = 2$

5. Resuelva los siguientes sistemas lineales:

```
import numpy as np
import scipy.linalg

def resolver_sistema_LU(L, U, b):
    """
    Resuelve un sistema de ecuaciones LUx = b en dos pasos:
    1. Ly = b (sustitución progresiva)
    2. Ux = y (sustitución regresiva)

    Parámetros:
    L: Matriz triangular inferior
    U: Matriz triangular superior
    b: Vector de términos independientes

    Retorna:
    x: Vector solución del sistema
    """
    # Paso 1: Resolver Ly = b (sustitución progresiva)
    y = scipy.linalg.solve_triangular(L, b, lower=True)

    # Paso 2: Resolver Ux = y (sustitución regresiva)
    x = scipy.linalg.solve_triangular(U, y, lower=False)

    return x
```

$$\textbf{a.}\quad \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 & -1 \\ 0 & -2 & 1 \\ 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$$

```python
L1 = np.array([
    [1, 0, 0],
    [2, 1, 0],
    [-1, 0, 1]
], dtype=float)

U1 = np.array([
    [2, 3, -1],
    [0, -2, 1],
    [0, 0, 3]
], dtype=float)

b1 = np.array([2, -1, 1], dtype=float)

solucion1 = resolver_sistema_LU(L1, U1, b1)
print("Solución para el sistema es:")
print(solucion1)
```

```
Solución para el sistema es:
[-3.  3.  1.]
```

$$\textbf{b.} \quad \begin{bmatrix} 2 & 0 & 0 \\ -1 & 1 & 0 \\ 3 & 2 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 3 \\ 0 \end{bmatrix}$$

```python
L2 = np.array([
    [2, 0, 0],
    [-1, 1, 0],
    [3, 2, -1]
], dtype=float)

U2 = np.array([
    [1, 1, 1],
    [0, 1, 2],
    [0, 0, 1]
], dtype=float)

b2 = np.array([-1, 3, 0], dtype=float)

solucion2 = resolver_sistema_LU(L2, U2, b2)
```

```python
print("\nSolución para el sistema es:")
print(solucion2)
```

```
Solución para el sistema es:
[ 0.5 -4.5  3.5]
```

6. Factorice las siguientes matrices en la descomposición LU mediante el algoritmo de factorización LU con $l_{ii} = 1$ para todas las i.

```python
import numpy as np

def factorizacion_LU(A):
    """
    Implementa la descomposición LU con la condición de que L tenga unos en la diagonal
    según el algoritmo de factorización LU proporcionado.

    Parámetro:
    A : np.array (matriz cuadrada de tamaño n x n)

    Retorna:
    L : np.array (matriz triangular inferior con unos en la diagonal)
    U : np.array (matriz triangular superior)
    """
    n = A.shape[0]  # Dimensión de la matriz
    L = np.eye(n)   # Inicializamos L como la identidad (diagonal con 1)
    U = np.zeros((n, n))  # Inicializamos U como una matriz de ceros

    for i in range(n):
        # Paso 4: Calcular elementos de la diagonal
        suma_LU = sum(L[i, k] * U[k, i] for k in range(i))
        U[i, i] = A[i, i] - suma_LU

        if U[i, i] == 0:
            raise ValueError("Factorización imposible: la matriz es singular.")

        # Paso 5: Calcular los elementos de la fila i en U y la columna i en L
        for j in range(i + 1, n):  # Calcular elementos de U
            suma_U = sum(L[i, k] * U[k, j] for k in range(i))
            U[i, j] = (A[i, j] - suma_U) / L[i, i]
```

```
        for j in range(i + 1, n):  # Calcular elementos de L
            suma_L = sum(L[j, k] * U[k, i] for k in range(i))
            L[j, i] = (A[j, i] - suma_L) / U[i, i]

    return L, U
```

a.

$$\begin{bmatrix} 2 & -1 & 1 \\ 3 & 3 & 9 \\ 3 & 3 & 5 \end{bmatrix}$$

```
A = np.array([
    [2, -1, 1],
    [3, 3, 9],
    [3, 3, 5]
], dtype=float)

L, U = factorizacion_LU(A)

print("Matriz L:")
print(L)
print("\nMatriz U:")
print(U)
```

```
Matriz L:
[[1.  0.  0. ]
 [1.5 1.  0. ]
 [1.5 1.  1. ]]

Matriz U:
[[ 2.  -1.   1. ]
 [ 0.   4.5  7.5]
 [ 0.   0.  -4. ]]
```

b.

$$\begin{bmatrix} 1.012 & -2.132 & 3.104 \\ -2.132 & 4.096 & -7.013 \\ 3.104 & -7.013 & 0.014 \end{bmatrix}$$

```python
A = np.array([
    [1.012, -2.132, 3.104],
    [-2.132, 4.096, -7.013],
    [3.104, -7.013, 0.014]
], dtype=float)

L, U = factorizacion_LU(A)

print("Matriz L:")
print(L)
print("\nMatriz U:")
print(U)
```

```
Matriz L:
[[ 1.          0.          0.        ]
 [-2.10671937  1.          0.        ]
 [ 3.06719368  1.19775553  1.        ]]

Matriz U:
[[ 1.012      -2.132       3.104     ]
 [ 0.         -0.39552569 -0.47374308]
 [ 0.          0.         -8.93914077]]
```

c.

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 1 & 1.5 & 0 & 0 \\ 0 & -3 & 0.5 & 0 \\ 2 & -2 & 1 & 1 \end{bmatrix}$$

```python
A = np.array([
    [2, 0, 0, 0],
    [1, 1.5, 0, 0],
    [0, -3, 0.5, 0],
    [2, -2, 1, 1]
], dtype=float)

L, U = factorizacion_LU(A)

print("Matriz L:")
print(L)
print("\nMatriz U:")
print(U)
```

```
Matriz L:
[[ 1.          0.          0.          0.        ]
 [ 0.5         1.          0.          0.        ]
 [ 0.         -2.          1.          0.        ]
 [ 1.         -1.33333333  2.          1.        ]]

Matriz U:
[[2.  0.  0.  0. ]
 [0.  1.5 0.  0. ]
 [0.  0.  0.5 0. ]
 [0.  0.  0.  1. ]]
```

d.

$$\begin{bmatrix} 2.1756 & 4.0231 & -2.1732 & 5.1967 \\ -4.0231 & 6.0000 & 0 & 1.1973 \\ -1.0000 & -5.2107 & 1.1111 & 0 \\ 6.0235 & 7.0000 & 0 & -4.1561 \end{bmatrix}$$

```python
A = np.array([
    [2.1756, 4.0231, -2.1732, 5.1967],
    [-4.0231, 6.0000, 0, 1.1973],
    [-1.0000, -5.2107, 1.1111, 0],
    [6.0235, 7.0000, 0, -4.1561]
], dtype=float)

L, U = factorizacion_LU(A)

print("Matriz L:")
print(L)
print("\nMatriz U:")
print(U)
```

```
Matriz L:
[[ 1.          0.          0.          0.        ]
 [-1.84919103  1.          0.          0.        ]
 [-0.45964332 -0.25012194  1.          0.        ]
 [ 2.76866152 -0.30794361 -5.35228302  1.        ]]

Matriz U:
[[ 2.1756      4.0231     -2.1732      5.1967    ]
 [ 0.         13.43948042 -4.01866194 10.80699101]
 [ 0.          0.         -0.89295239  5.09169403]
 [ 0.          0.          0.         12.03612803]]
```

7. Modifique el algoritmo de eliminación gaussiana de tal forma que se pueda utilizar para resolver un sistema lineal usando la descomposición LU y, a continuación, resuelva los siguientes sistemas lineales.

```
import numpy as np

def factorizacion_LU(A):
    """
    Implementa la eliminación gaussiana modificada para calcular la descomposición LU.

    Parámetro:
    A : np.array (matriz cuadrada de tamaño n x n)

    Retorna:
    L : np.array (matriz triangular inferior con unos en la diagonal)
    U : np.array (matriz triangular superior)
    """
    n = A.shape[0]
    L = np.eye(n)   # Inicializamos L con 1 en la diagonal
    U = A.astype(float).copy()  # Copiamos A para modificarla en U

    for i in range(n):
        if U[i, i] == 0:
            raise ValueError("Factorización imposible: la matriz es singular o requiere pivot

        for j in range(i+1, n):
            factor = U[j, i] / U[i, i]  # Calculamos el multiplicador
            L[j, i] = factor  # Guardamos el factor en L
            U[j, i:] -= factor * U[i, i:]  # Modificamos filas en U

    return L, U

def resolver_sistema_LU(L, U, b):
    """
    Resuelve el sistema LUx = b en dos pasos:
    1. Ly = b (sustitución progresiva)
    2. Ux = y (sustitución regresiva)

    Parámetros:
    L: Matriz triangular inferior
    U: Matriz triangular superior
    b: Vector de términos independientes
```

```
    Retorna:
    x: Vector solución del sistema
    """
    n = L.shape[0]

    # Sustitución progresiva para resolver Ly = b
    y = np.zeros(n)
    for i in range(n):
        y[i] = b[i] - np.dot(L[i, :i], y[:i])

    # Sustitución regresiva para resolver Ux = y
    x = np.zeros(n)
    for i in range(n-1, -1, -1):
        x[i] = (y[i] - np.dot(U[i, i+1:], x[i+1:])) / U[i, i]

    return x
```

a.

$$\begin{aligned} 2x_1 - x_2 + x_3 &= -1, \\ 3x_1 + 3x_2 + 9x_3 &= 0, \\ 3x_1 + 3x_2 + 5x_3 &= 4. \end{aligned}$$

```
A = np.array([
    [2, -1, 1],
    [3, 3, 9],
    [3, 3, 5]
], dtype=float)

b = np.array([-1, 0, 4], dtype=float)

L, U = factorizacion_LU(A)
x = resolver_sistema_LU(L, U, b)

print("Matriz L:")
print(L)
print("\nMatriz U:")
print(U)
print("\nSolución del sistema:")
print(x)
```

```
Matriz L:
```

```
[[1.  0.  0. ]
 [1.5 1.  0. ]
 [1.5 1.  1. ]]

Matriz U:
[[ 2.  -1.   1. ]
 [ 0.   4.5  7.5]
 [ 0.   0.  -4. ]]

Solución del sistema:
[ 1.  2. -1.]
```

b.

$$
\begin{aligned}
1.012x_1 - 2.132x_2 + 3.104x_3 &= 1.984, \\
-2.132x_1 + 4.096x_2 - 7.013x_3 &= -5.049, \\
3.104x_1 - 7.013x_2 + 0.014x_3 &= -3.895.
\end{aligned}
$$

```python
A = np.array([
    [1.012, -2.132, 3.104],
    [-2.132, 4.096, -7.013],
    [3.104, -7.013, 0.014]
], dtype=float)

b = np.array([1.984, -5.049, -3.895], dtype=float)

L, U = factorizacion_LU(A)
x = resolver_sistema_LU(L, U, b)

print("Matriz L:")
print(L)
print("\nMatriz U:")
print(U)
print("\nSolución del sistema:")
print(x)
```

```
Matriz L:
[[ 1.          0.          0.        ]
 [-2.10671937  1.          0.        ]
 [ 3.06719368  1.19775553  1.        ]]

Matriz U:
[[ 1.012      -2.132       3.104     ]
```

```
 [ 0.         -0.39552569 -0.47374308]
 [ 0.          0.         -8.93914077]]

Solución del sistema:
[1. 1. 1.]
```

c.

$$
\begin{aligned}
2x_1 &= 3, \\
x_1 + 1.5x_2 &= 4.5, \\
-3x_2 + 0.5x_3 &= -6.6, \\
2x_1 - 2x_2 + x_3 + x_4 &= 0.8.
\end{aligned}
$$

```python
A = np.array([
    [2, 0, 0, 0],
    [1, 1.5, 0, 0],
    [0, -3, 0.5, 0],
    [2, -2, 1, 1]
], dtype=float)

b = np.array([3, 4.5, -6.6, 0.8], dtype=float)

L, U = factorizacion_LU(A)
x = resolver_sistema_LU(L, U, b)

print("Matriz L:")
print(L)
print("\nMatriz U:")
print(U)
print("\nSolución del sistema:")
print(x)
```

```
Matriz L:
[[ 1.          0.          0.          0.        ]
 [ 0.5         1.          0.          0.        ]
 [ 0.         -2.          1.          0.        ]
 [ 1.         -1.33333333  2.          1.        ]]

Matriz U:
[[2.  0.  0.  0. ]
 [0.  1.5 0.  0. ]
```

```
 [0.  0.  0.5 0. ]
 [0.  0.  0.  1. ]]

Solución del sistema:
[ 1.5  2.  -1.2  3. ]
```

d.

$$
\begin{aligned}
2.1756x_1 + 4.0231x_2 - 2.1732x_3 + 5.1967x_4 &= 17.102,\\
-4.0231x_1 + 6.0000x_2 + 1.1973x_4 &= -6.1593,\\
-1.0000x_1 - 5.2107x_2 + 1.1111x_3 &= 3.0004,\\
6.0235x_1 + 7.0000x_2 - 4.1561x_4 &= 0.0000.
\end{aligned}
$$

```python
A = np.array([
    [2.1756, 4.0231, -2.1732, 5.1967],
    [-4.0231, 6.0000, 0, 1.1973],
    [-1.0000, -5.2107, 1.1111, 0],
    [6.0235, 7.0000, 0, -4.1561]
], dtype=float)

b = np.array([17.102, -6.1593, 3.0004, 0.0000], dtype=float)

L, U = factorizacion_LU(A)
x = resolver_sistema_LU(L, U, b)

print("Matriz L:")
print(L)
print("\nMatriz U:")
print(U)
print("\nSolución del sistema:")
print(x)
```

```
Matriz L:
[[ 1.          0.          0.          0.        ]
 [-1.84919103  1.          0.          0.        ]
 [-0.45964332 -0.25012194  1.          0.        ]
 [ 2.76866152 -0.30794361 -5.35228302  1.        ]]

Matriz U:
[[ 2.17560000e+00  4.02310000e+00 -2.17320000e+00  5.19670000e+00]
 [ 0.00000000e+00  1.34394804e+01 -4.01866194e+00  1.08069910e+01]
```

```
 [ 0.00000000e+00  4.44089210e-16 -8.92952394e-01  5.09169403e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.20361280e+01]]

Solución del sistema:
[2.9398512  0.0706777  5.67773512 4.37981223]
```