ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA EN SISTEMAS

METODOS NUMERICOS ICCD412

JUAN FRANCISCO PINTO ANDRANGO

Tarea 10 Ejercicios Unidad 04-C Descomposición LU

GR1CC

FECHA DE ENTREGA 28 DE ENERO DEL 2026

Resuelva los ejercicios adjuntos.

Indicaciones

Puede realizar los cálculos a mano, o utilizar cualquier librería o implementar su propia función.

En caso de usar código, subir la resolución de los ejercicios en un repositorio público en Github e incluir enlace de su repositorio.

1 Realice las siguientes multiplicaciones Matriz-matriz

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

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

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

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

In [2]:
```python
import numpy as np
# creacmos np arrays para cada caso para multiplicar las matrices
# a)
A_a = np.array([[2, -3], [3, -1]])
B_a = np.array([[1, 5], [2, 0]])
# b)
A_b = np.array([[2, -3], [3, -1]])
B_b = np.array([[1, 5, -4], [-3, 2, 0]])
# c)
A_c = np.array([[2, -3, 1], [4, 3, 0], [5, 2, -4]])
B_c = np.array([[1, 0, 1, -2], [1, 0, -1, -1], [2, 3, -2, -2]])
# d)
A_d = np.array([[2, 1, 2], [-2, 3, 0], [2, -1, 3]])
```

```
B_d = np.array([[1, -2], [-4, 1], [0, 2]])

result_a = np.dot(A_a, B_a)
result_b = np.dot(A_b, B_b)
result_c = np.dot(A_c, B_c)
result_d = np.dot(A_d, B_d)

print(" a):\n", result_a)
print(" b):\n", result_b)
print(" c):\n", result_c)
print(" d):\n", result_d)
```

```
a):
[[-4 10]
 [ 1 15]]
b):
[[ 11   4  -8]
 [  6  13 -12]]
c):
[[  1   3   3  -3]
 [  7   0   1 -11]
 [ -1 -12  11  -4]]
d):
[[ -2   1]
 [-14   7]
 [  6   1]]
```

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

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

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

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

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

In [3]:
```
import numpy as np

A_a = np.array([[4, 2, 6], [3, 0, 7], [-2, -1, -3]])
A_b = np.array([[1, 2, 0], [2, 1, -1], [3, 1, 1]])
A_c = np.array([[1, 1, -1, 1], [1, 2, -4, -2], [2, 1, 1, 5], [-1, 0, -2, -4]])
A_d = np.array([[4, 0, 0, 0], [6, 7, 0, 0], [9, 11, 1, 0], [5, 4, 1, 1]])

matrices = [("a", A_a), ("b", A_b), ("c", A_c), ("d", A_d)]

for name, matrix in matrices:
    try:
        det = np.linalg.det(matrix)
        if det != 0:
```

```
            inverse = np.linalg.inv(matrix)
            print(f"La matriz {name} es no singular.")
            print(f"Inversa de {name}:\n", inverse)
        else:
            print(f"La matriz {name} es singular (determinante = 0).")
    except np.linalg.LinAlgError:
        print(f"La matriz {name} no tiene inversa.")
```

```
La matriz a es singular (determinante = 0).
La matriz b es no singular.
Inversa de b:
 [[-0.25   0.25   0.25 ]
 [ 0.625 -0.125 -0.125]
 [ 0.125 -0.625  0.375]]
La matriz c es singular (determinante = 0).
La matriz d es no singular.
Inversa de d:
 [[ 0.25        0.          0.          0.        ]
 [-0.21428571  0.14285714 -0.         -0.        ]
 [ 0.10714286 -1.57142857  1.         -0.        ]
 [-0.5         1.         -1.          1.        ]]
```

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

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

In [6]:
```
import numpy as np

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

b1 = np.array([6, 4, -2, 5])
b2 = np.array([1, 1, 2, -1])

x1 = np.linalg.solve(A, b1)
x2 = np.linalg.solve(A, b2)

print("la solucion del primer sistema:\n", x1)
print("la solucion del segundo sistema:\n", x2)
```

```
la solucion del primer sistema:
 [ 3. -6. -2. -1.]
la solucion del segundo sistema:
 [1. 1. 1. 1.]
```

4 Encuentre los valor de A que hace que la siguiente matriz sea singualar.

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

In [8]:
```python
import sympy as sp

alpha = sp.symbols('alpha')

A = sp.Matrix([
    [1, -1, alpha],
    [2, 2, 1],
    [0, alpha, -3/2]
])

det_A = A.det()
solutions = sp.solve(det_A, alpha)

print(f"El det de A: {det_A}")
print(f"valores para  alpha que hacen que A sea singular: {solutions}")
```

```
El det de A: 2*alpha**2 - alpha - 6.0
valores para  alpha que hacen que A sea singular: [-1.50000000000000, 2.0000
0000000000]
```

5 Resuelva los siguientes sistemas lineales:

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

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

In [9]:
```python
import sympy as sp

A_a = sp.Matrix([
    [1, 0, 0],
```

```
    [2, 1, 0],
    [-1, 0, 1]
])
B_a = sp.Matrix([
    [2, 3, -1],
    [0, -2, 1],
    [0, 0, 3]
])

C_a = sp.Matrix([2, -1, 1])
x1, x2, x3 = sp.symbols('x1 x2 x3')
X = sp.Matrix([x1, x2, x3])
solution_a = sp.solve(A_a * (B_a * X) - C_a, [x1, x2, x3])

A_b = sp.Matrix([
    [2, 0, 0],
    [-1, 1, 0],
    [3, 2, -1]
])

B_b = sp.Matrix([[1, 1, 1], [0, 1, 2], [0, 0, 1]])

C_b = sp.Matrix([-1, 3, 0])
solution_b = sp.solve(A_b * (B_b * X) - C_b, [x1, x2, x3])

print("Solución del sistema a:", solution_a)
print("Solución del sistema b:", solution_b)
```

```
Solución del sistema a: {x1: -3, x2: 3, x3: 1}
Solución del sistema b: {x1: 1/2, x2: -9/2, x3: 7/2}
```

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

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

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

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

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

In [11]:
```
def lu_factorization(matrix):
    L, U = sp.eye(matrix.shape[0]), matrix.copy()
    for i in range(matrix.shape[0]):
        for j in range(i + 1, matrix.shape[0]):
            factor = U[j, i] / U[i, i]
            L[j, i] = factor
            U[j, :] -= factor * U[i, :]
    return L, U

A_a = sp.Matrix([
```

```
    [2, -1, 1],
    [3, 3, 9],
    [3, 3, 5]
])
L_a, U_a = lu_factorization(A_a)

A_b = sp.Matrix([
    [1.012, -2.132, 3.104],
    [-2.132, 4.096, -7.013],
    [3.104, -7.013, 0.014]
])
L_b, U_b = lu_factorization(A_b)

A_c = sp.Matrix([
    [2, 0, 0, 0],
    [1, 1.5, 0, 0],
    [0, -3, 0.5, 0],
    [2, -2, 1, 1]
])
L_c, U_c = lu_factorization(A_c)

A_d = sp.Matrix([
    [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]
])
L_d, U_d = lu_factorization(A_d)

print("factorizacion LU de la matriz A:")
print("L:", L_a)
print("U:", U_a)
print("\nfactorizacion LU de la matriz B:")
print("L:", L_b)
print("U:", U_b)
print("\nfactorizacion LU de la matriz C:")
print("L:", L_c)
print("U:", U_c)
print("\nfactorizacion LU de la matriz D:")
print("L:", L_d)
print("U:", U_d)
```

```
factorizacion LU de la matriz A:
L: Matrix([[1, 0, 0], [3/2, 1, 0], [3/2, 1, 1]])
U: Matrix([[2, -1, 1], [0, 9/2, 15/2], [0, 0, -4]])

factorizacion LU de la matriz B:
L: Matrix([[1, 0, 0], [-2.10671936758893, 1, 0], [3.06719367588933, 1.197755
52624215, 1]])
U: Matrix([[1.01200000000000, -2.13200000000000, 3.10400000000000], [0, -0.3
95525691699605, -0.473743083003951], [0, 0, -8.939140774273 50]])

factorizacion LU de la matriz C:
L: Matrix([[1, 0, 0, 0], [1/2, 1, 0, 0], [0, -2.00000000000000, 1, 0], [1, -
1.33333333333333, 2.00000000000000, 1]])
U: Matrix([[2, 0, 0, 0], [0, 1.50000000000000, 0, 0], [0, 0, 0.5000000000000
00, 0], [0, 0, 0, 1]])

factorizacion LU de la matriz D:
L: Matrix([[1, 0, 0, 0], [-1.84919102776246, 1, 0, 0], [-0.459643316786174,
-0.250121944170316, 1, 0], [2.76866151866152, -0.307943612790330, -5.3522830
2043569, 1]])
U: Matrix([[2.17560000000000, 4.02310000000000, -2.17320000000000, 5.1967000
0000000], [0, 13.4394804237911, -4.01866194153337, 10.8069910139732], [0, 4.
44089209850063e-16, -0.892952393819297, 5.09169402738881], [0, 2.37689113743
919e-15, 0, 12.0361280302542]])
```

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..

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

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

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

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

In [13]:
```python
def lu_decomposition(a):
    n = len(a)
    L = np.zeros((n, n))
    U = np.zeros((n, n))

    for i in range(n):
        # Calcula la matriz U
        for k in range(i, n):
            suma = sum(L[i][j] * U[j][k] for j in range(i))
```

```
            U[i][k] = a[i][k] - suma

        for k in range(i, n):
            if i == k:
                L[i][i] = 1
            else:
                suma = sum(L[k][j] * U[j][i] for j in range(i))
                L[k][i] = (a[k][i] - suma) / U[i][i]

    return L, U

def forward_substitution(L, b):
    n = len(b)
    y = np.zeros(n)
    for i in range(n):
        y[i] = (b[i] - sum(L[i][j] * y[j] for j in range(i))) / L[i][i]
    return y

def backward_substitution(U, y):
    n = len(y)
    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = (y[i] - sum(U[i][j] * x[j] for j in range(i + 1, n))) / U[i][
    return x

def solve_lu(a, b):
    L, U = lu_decomposition(a)
    y = forward_substitution(L, b)
    x = backward_substitution(U, y)
    return x

sistemas = [
    {
        "A": np.array([
            [2, -1, 1],
            [3, 3, 9],
            [3, 3, 5]
        ]),
        "b": np.array([-1, 0, 4])
    },
    {
        "A": np.array([
            [1.012, -2.132, 3.104],
            [-2.132, 4.096, -7.013],
            [3.104, -7.013, 0.014]
        ]),
        "b": np.array([1.984, -5.049, -3.895])
    },
    {
        "A": np.array([
            [2, 0, 0, 0],
            [1, 1.5, 0, 0],
            [0, -3, 0.5, 0],
            [0, -2, 1, 1]
        ]),
        "b": np.array([3, 4.5, -6.6, 0.8])
```

```
    },
    {
        "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]
        ]),
        "b": np.array([17.102, -6.1593, 3.0004, 0])
    }
]

for i, sistema in enumerate(sistemas, start=1):
    A, b = sistema["A"], sistema["b"]
    x = solve_lu(A, b)
    print(f"la solucion del sistema es  {chr(96 + i)}: {x}\n")
```

```
la solucion del sistema es  a: [ 1.  2. -1.]

la solucion del sistema es  b: [1. 1. 1.]

la solucion del sistema es  c: [ 1.5  2.  -1.2  6. ]

la solucion del sistema es  d: [2.9398512  0.0706777  5.67773512 4.37981223]
```

Link del Repositorio de Git-Hub

https://github.com/JuanfranPinto/Metodos-Numericos-/blob/main/Tarea-10-Ejercicios-Unidad-04-C-Descomposici%C3%B3n-LU-PJF.ipynb