Escuela Politécnica Nacional

Nombre: Francisco Ulloa
Fecha: Quito, 20 de enero de 2026
Tema: Factorización LU
Repositorio:
https://github.com/Fu5CHAR/Metodos_numericos_2025B_Ulloa-Francisco/tree/main

1. Realice las siguientes multiplicaciones matriz-matriz:

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

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

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

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

In [44]:
```
import numpy as np
A1= np.array([[2, -3],[3, -1]])
A2=np.array([[1, 5],[2,0]])
B1=np.array([[2, -3],[3, -1]])
B2=np.array([[1, 5, -4],[-3,2,0]])
C1=np.array([[2,-3,1],[4,3,1],[5,2,-4]])
C2=np.array([[0,1,-2],[1,0,-1],[2,3,-2]])
D1=np.array([[2,1,2],[-2,3,0],[2,-1,3]])
D2=np.array([[1,-2],[-4,1],[0,2]])

resultadoA1_A2 = A1 @ A2
print("Resultado de A1 * A2:\n", resultadoA1_A2)
resultadoB1_B2 = B1 @ B2
print("Resultado de B1 * B2:\n", resultadoB1_B2)
resultadoC1_C2 = C1 @ C2
print("Resultado de C1 * C2:\n", resultadoC1_C2)
resultadoD1_D2 = D1 @ D2
print("Resultado de D1 * D2:\n", resultadoD1_D2)
```

```
Resultado de A1 * A2:
 [[-4 10]
 [ 1 15]]
Resultado de B1 * B2:
 [[ 11   4  -8]
 [  6  13 -12]]
Resultado de C1 * C2:
 [[ -1   5  -3]
 [  5   7 -13]
 [ -6  -7  -4]]
Resultado de D1 * D2:
 [[ -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 [45]:
```python
import numpy as np
A=np.array([[4,2,6],[3,0,7],[-2,-1,-3]])
B=np.array([[1,2,0],[2,1,-1],[3,1,1]])
C=np.array([[1,1,-1,1],[1,2,-4,-2],[2,1,1,5],[-1,0,2,-4]])
D=np.array([[4,0,0,0],[6,7,0,0],[9,11,1,0],[5,4,1,1]])
def invert_matrix(matrix):
    if np.linalg.det(matrix) != 0:
        inv_matrix = np.linalg.inv(matrix)
        return print('La matriz :\n', matrix, '\nes no singular y la inversa de
    else:
        return print("La matriz:\n", matrix, "\nes singular y no tiene inversa."

invert_matrix(A)
invert_matrix(B)
invert_matrix(C)
invert_matrix(D)
```

```
La matriz:
 [[ 4  2  6]
 [ 3  0  7]
 [-2 -1 -3]]
es singular y no tiene inversa.
La matriz :
 [[ 1  2  0]
 [ 2  1 -1]
 [ 3  1  1]]
es no singular y la inversa de la matriz es:
 [[-0.25   0.25   0.25 ]
 [ 0.625 -0.125 -0.125]
 [ 0.125 -0.625  0.375]]
La matriz:
 [[ 1  1 -1  1]
 [ 1  2 -4 -2]
 [ 2  1  1  5]
 [-1  0  2 -4]]
es singular y no tiene inversa.
La matriz :
 [[ 4  0  0  0]
 [ 6  7  0  0]
 [ 9 11  1  0]
 [ 5  4  1  1]]
es no singular y la inversa de la matriz es:
 [[ 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 4x4 que tienen la misma matriz de coeficientes:

$$S1: \begin{array}{ccccc} x_1 & -x_2 & +x_3 & -x_4 & =6 \\ x_1 & & -x_3 & +x_4 & =4 \\ 2x_1 & +x_2 & +3x_3 & -4x_4 & =-2 \\ & -x_2 & +x_3 & -x4 & =5 \end{array}$$

$$S2: \begin{array}{ccccc} x_1 & -x_2 & +x_3 & -x_4 & =1 \\ x_1 & & -x_3 & +x_4 & =1 \\ 2x_1 & +x_2 & +3x_3 & -4x_4 & =2 \\ & -x_2 & +x_3 & -x4 & =-1 \end{array}$$

In [46]:
```python
matriz = 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])
inv_matrix = np.linalg.inv(matriz)
solucion1 = inv_matrix @ b1
print("Solución del sistema 1:\n", solucion1)
solucion2 = inv_matrix @ b2
print("Solución del sistema 2:\n", solucion2)
```

```
Solución del sistema 1:
 [ 3. -6. -2. -1.]
Solución del sistema 2:
 [1. 1. 1. 1.]
```

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

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

Para que una matriz sea singular, esta no debe ser invertible, es decir, su determinante debe ser igual a cero.
Entonces buscamos los valores de α tal que $det[A] = 0$.
Luego,

$$\begin{aligned} det[A] &= ((1 * 2 * \frac{-3}{2}) + (2 * \alpha * \alpha) + (0 * 1 * -1)) \\ &- ((\alpha * 2 * 0) + (1 * \alpha * 1) + (\frac{-3}{2} * 2 * -1)) \\ &= (-3 + 2(\alpha)^2 + 0) - (0 + \alpha + 3) \\ &= 2(\alpha)^2 - \alpha - 6 \end{aligned}$$

Si $det(A) = 0$, entonces

$$\begin{aligned} 2(\alpha)^2 - \alpha - 6 = 0 \\ (\alpha - 3)(\alpha + 2) = 0 \end{aligned}$$

Finalmente, la matriz A es singular para $\alpha = 3$ y $\alpha = -2$.

5. Resuelva los siguientes sistemas lineales

a.

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

Tenemos un sistema lineal de la forma (LUx = b). Definimos (Ux = y), entonces (Ly = b).

Primero resolvemos (Ly = b):

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$$

De donde:

$$\begin{aligned} &y_1 = 2, \\ &y_2 = -2y_1 - 1 = -2(2) - 1 = -5, \\ &y_3 = 1 + y_1 = 1 + 2 = 3. \end{aligned}$$

Ahora resolvemos (Ux = y):

$$\begin{bmatrix} 2 & 3 & -1 \\ 0 & -2 & -1 \\ 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -5 \\ 3 \end{bmatrix}$$

Entonces:

$$\begin{aligned} x_3 &= 1, \\ x_2 &= \frac{-5 + x_3}{-2} = \frac{-5+1}{-2} = 2, \\ x_1 &= \frac{2 - 3x_2 + x_3}{2} = \frac{2 - 3(2) + 1}{2} = -\frac{3}{2}. \end{aligned}$$

Finalmente, $x_1 = -\frac{3}{2}, \quad x_2 = 2, \quad x_3 = 1.$

b.

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

Definimos nuevamente (Ux = y) y resolvemos primero (Ly = b):

$$\begin{bmatrix} 2 & 0 & 0 \\ -1 & 1 & 0 \\ 3 & 2 & -1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 3 \\ 0 \end{bmatrix}$$

De donde:

$$\begin{aligned} y_1 &= -\frac{1}{2}, \\ y_2 &= 3 + y_1 = 3 - \frac{1}{2} = \frac{5}{2}, \\ y_3 &= 3y_1 + 2y_2 = 3\left(-\frac{1}{2}\right) + 2\left(\frac{5}{2}\right) = \frac{7}{2}. \end{aligned}$$

Ahora resolvemos (Ux = y):

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} \\ \frac{5}{2} \\ \frac{7}{2} \end{bmatrix}$$

Entonces:

$$\begin{aligned} x_3 &= \frac{7}{2}, \\ x_2 &= \frac{5}{2} - 2x_3 = \frac{5}{2} - 7 = -\frac{9}{2}, \\ x_1 &= -\frac{1}{2} - x_2 - x_3 = -\frac{1}{2} + \frac{9}{2} - \frac{7}{2} = \frac{1}{2}. \end{aligned}$$

Finalmente, $x_1 = \frac{1}{2}, \quad x_2 = -\frac{9}{2}, \quad x_3 = \frac{7}{2}.$

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

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

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

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

$$d. \begin{bmatrix} 2.1756 & 4.0231 & -2.1732 & 5.1967 \\ -4.0231 & 6.000 & 0 & -7.013 \\ -1.000 & -5.2107 & 1.1111 & 0 \\ 6.0235 & 7.000 & 0 & -4.1561 \end{bmatrix}$$

In [1]:
```
import numpy as np
import logging

def factorizacion_LU(A: np.ndarray | list[list[float | int]]):
    if not isinstance(A, np.ndarray):
        logging.debug("Convirtiendo A a numpy array.")
        A = np.array(A, dtype=float)

    assert A.shape[0] == A.shape[1], \
        "La matriz A debe ser cuadrada."

    n = A.shape[0]

    # Inicialización
    U = A.copy()
    L = np.eye(n)

    # ------------------------------------------------
    # Eliminación hacia adelante (tipo Gauss)
    # ------------------------------------------------
    for i in range(n - 1):

        if U[i, i] == 0:
            raise ValueError(
                "Pivote cero encontrado. "
                "Esta implementación no usa pivoteo."
            )

        for j in range(i + 1, n):
            m = U[j, i] / U[i, i]
            L[j, i] = m          # guardamos el multiplicador

            for k in range(i, n):
                U[j, k] -= m * U[i, k]
```

```
        logging.info(f"Paso {i}\nL:\n{L}\nU:\n{U}\n")

    return L, U
```

In [3]:
```
A_a = [
    [2, -1,  1],
    [3,  3,  9],
    [3,  3,  5]
]
L, U = factorizacion_LU(A_a)
print("L =\n", L)
print("U =\n", U)
```

```
L =
 [[1.  0.  0. ]
 [1.5 1.  0. ]
 [1.5 1.  1. ]]
U =
 [[ 2.  -1.   1. ]
 [ 0.   4.5  7.5]
 [ 0.   0.  -4. ]]
```

In [4]:
```
A_b = [
    [ 1.012, -2.132,  3.104],
    [-2.132,  4.096, -7.013],
    [ 3.104, -7.013,  0.014]
]
L, U = factorizacion_LU(A_b)
print("L =\n", L)
print("U =\n", U)
```

```
L =
 [[ 1.          0.          0.        ]
 [-2.10671937  1.          0.        ]
 [ 3.06719368  1.19775553  1.        ]]
U =
 [[ 1.012      -2.132       3.104     ]
 [ 0.         -0.39552569 -0.47374308]
 [ 0.          0.         -8.93914077]]
```

In [5]:
```
A_c = [
    [2,   0,   0,   0],
    [1,   1.5, 0,   0],
    [0,  -3,   0.5, 0],
    [2,  -2,   1,   1]
]
L, U = factorizacion_LU(A_c)
print("L =\n", L)
print("U =\n", U)
```

```
L =
 [[ 1.          0.          0.          0.        ]
 [ 0.5         1.          0.          0.        ]
 [ 0.         -2.          1.          0.        ]
 [ 1.         -1.33333333  2.          1.        ]]
U =
 [[2.  0.  0.  0. ]
 [0.  1.5 0.  0. ]
 [0.  0.  0.5 0. ]
 [0.  0.  0.  1. ]]
```

In [6]:
```
A_d = [
    [ 2.1756,  4.0231, -2.1732,  5.1967],
    [-4.0231,  6.0000,  0.0000, -7.0130],
    [-1.0000, -5.2107,  1.1111,  0.0000],
    [ 6.0235,  7.0000,  0.0000, -4.1561]
]
L, U = factorizacion_LU(A_d)
print("L =\n", L)
print("U =\n", U)
```

```
L =
 [[ 1.          0.          0.          0.        ]
 [-1.84919103  1.          0.          0.        ]
 [-0.45964332 -0.25012194  1.          0.        ]
 [ 2.76866152 -0.30794361 -5.35228302  1.        ]]
U =
 [[ 2.17560000e+00  4.02310000e+00 -2.17320000e+00  5.19670000e+00]
 [ 0.00000000e+00  1.34394804e+01 -4.01866194e+00  2.59669101e+00]
 [ 0.00000000e+00  4.44089210e-16 -8.92952394e-01  3.03811783e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00 -1.48350243e+00]]
```

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.

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

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

$$c.\ \begin{array}{ccccc} 2x_1 & 0 & 0 & 0 & =3 \\ 1x_1 & +1.5x_2 & 0 & 0 & =4.5 \\ 0 & -3x_2 & +0.5x_3 & 0 & =-6.6 \\ 2x_1 & -2x_2 & +1x_3 & +1x_4 & =0.8 \end{array}$$

$$d.\ \begin{array}{ccccc} 2.1756x_1 & +4.0231x_2 & -2.1732x_3 & 5.1967x_4 & =17.102 \\ -4.0231x_1 & +6.000x_2 & 0 & -7.013x_4 & -6.1593 \\ -1.000x_1 & -5.2107x_2 & +1.1111x_3 & 0 & =3.0004 \\ 6.0235x_1 & +7.000x_2 & 0 & -4.1561x_4 & =0.0000 \end{array}$$

In [7]:
```
import numpy as np

def sustitucion_adelante(L: np.ndarray, b: np.ndarray | list):
    if not isinstance(L, np.ndarray):
        L = np.array(L, dtype=float)

    if not isinstance(b, np.ndarray):
        b = np.array(b, dtype=float)

    n = L.shape[0]
    y = np.zeros(n)

    for i in range(n):
```

```
        suma = 0.0
        for j in range(i):
            suma += L[i, j] * y[j]

        # Como L[i,i] = 1
        y[i] = b[i] - suma

    return y


def sustitucion_atras(U: np.ndarray, y: np.ndarray | list):
    if not isinstance(U, np.ndarray):
        U = np.array(U, dtype=float)

    if not isinstance(y, np.ndarray):
        y = np.array(y, dtype=float)

    n = U.shape[0]
    x = np.zeros(n)

    for i in range(n - 1, -1, -1):
        suma = 0.0
        for j in range(i + 1, n):
            suma += U[i, j] * x[j]

        if U[i, i] == 0:
            raise ValueError("Pivote cero en U. No se puede resolver.")

        x[i] = (y[i] - suma) / U[i, i]

    return x
```

In [8]:

```
A_a = [
    [2, -1,  1],
    [3,  3,  9],
    [3,  3,  5]
]
b_1=[-1,0,4]
L, U = factorizacion_LU(A_a)
y = sustitucion_adelante(L, b_1)
x = sustitucion_atras(U, y)

print("y =", y)
print("x =", x)
```

```
y = [-1.   1.5  4. ]
x = [ 1.  2. -1.]
```

In [9]:

```
A_b = [
    [ 1.012, -2.132,  3.104],
    [-2.132,  4.096, -7.013],
    [ 3.104, -7.013,  0.014]
]
b_2=[1.984,-5.049,-3.895]
L, U = factorizacion_LU(A_b)
y = sustitucion_adelante(L, b_2)
x = sustitucion_atras(U, y)

print("y =", y)
print("x =", x)
```

```
y = [ 1.984      -0.86926877 -8.93914077]
x = [1. 1. 1.]
```

In [10]:
```
A_c = [
    [2,   0,   0,   0],
    [1,   1.5, 0,   0],
    [0,  -3,   0.5, 0],
    [2,  -2,   1,   1]
]
b_3=[3.,4.5,-6.6,0.8]
L, U = factorizacion_LU(A_c)
y = sustitucion_adelante(L, b_3)
x = sustitucion_atras(U, y)

print("y =", y)
print("x =", x)
```

```
y = [ 3.   3.  -0.6  3. ]
x = [ 1.5  2.  -1.2  3. ]
```

In [11]:
```
A_d = [
    [ 2.1756,  4.0231, -2.1732,  5.1967],
    [-4.0231,  6.0000,  0.0000, -7.0130],
    [-1.0000, -5.2107,  1.1111,  0.0000],
    [ 6.0235,  7.0000,  0.0000, -4.1561]
]
b_4=[17.102,-6.1593,3.0004,0.0000]
L, U = factorizacion_LU(A_d)
y = sustitucion_adelante(L, b_4)
x = sustitucion_atras(U, y)

print("y =", y)
print("x =", x)
```

```
y = [17.102      25.46556496 17.23071662 52.71598078]
x = [  14.01865203  -33.16108356 -140.19746743  -35.53481254]
```