Escuela Politécnica Nacional

Nombre: Francisco Ulloa
Fecha: Quito, 13 de enero de 2026
Tema: Matriz inversa y descomposición LU
Repositorio:
https://github.com/Fu5CHAR/Metodos_numericos_2025B_Ulloa-Francisco/tree/main

Haga modificaciones a las funciones base del siguiente código

1. para encontrar la matriz inversa de las siguientes matrices:

In [47]:
```
A1 = [[1,3,4],[2,1,3],[4,2,1]]
A2= [[1,2,3],[0,1,4],[5,6,0]]
A3=[[4,2,1],[2,1,3],[1,3,4]]
A4 =[[2,4,6,1],[4,7,5,-6],[2,5,18,10],[6,12,38,16]]
b1=[1,2,4]
b2=[3,-5,2]
b3=[7,8,-1]
c1=[1,2,4,5]
c2=[3,-5,2,6]
c3=[7,8,-1,0]
```

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

def gauss_jordan(A: np.ndarray) -> np.ndarray:
    """Resuelve un sistema de ecuaciones lineales mediante el método de eliminación

    ## Parameters

    ``A``: matriz aumentada del sistema de ecuaciones lineales. Debe ser de tamaño

    ## Return

    ``solucion``: vector con la solución del sistema de ecuaciones lineales.

    """
    if not isinstance(A, np.ndarray):
        logging.debug("Convirtiendo A a numpy array.")
        A = np.array(A)
    n = A.shape[0]  # número de filas

    for i in range(0, n):  # loop por columna

        # --- encontrar pivote
        p = None  # default, first element
        for pi in range(i, n):
            if A[pi, i] == 0:
```

```
                # must be nonzero
                continue

            if p is None:
                # first nonzero element
                p = pi
                continue

            if abs(A[pi, i]) < abs(A[p, i]):
                p = pi

        if p is None:
            # no pivot found.
            raise ValueError("No existe solución única.")

        if p != i:
            # swap rows
            logging.debug(f"Intercambiando filas {i} y {p}")
            _aux = A[i, :].copy()
            A[i, :] = A[p, :].copy()
            A[p, :] = _aux

        # --- Eliminación: loop por fila
        for j in range(0, n):
            if j != i:
                m = A[j, i] / A[i, i]
                A[j, i:] = A[j, i:] - m * A[i, i:]

        for i in range(n):

            for j in range(n):
                if A[i,j] !=0:
                    A[i,:] = A[i,:]/A[i,j]
                    break

        logging.info(f"\n{A}")



    return A
```

In [27]:

```
import numpy as np
import logging
def separar_inversa(AI: np.ndarray) -> tuple[np.ndarray, np.ndarray]:
    """Separa la matriz aumentada en la matriz de coeficientes y el vector de térmi

    ## Parameters
    ``Ab``: matriz aumentada.

    ## Return
    ``A``: matriz de coeficientes.
    ``b``: vector de términos independientes.
    """
    n, m = AI.shape
    logging.debug(f"AI = \n{AI}")
    if not isinstance(AI, np.ndarray):
```

```
        logging.debug("Convirtiendo Ab a numpy array")
        AI = np.array(AI, dtype=float)
    return AI[:, :m-n], AI[:, m-n:]
```

$$A_1 = \begin{bmatrix} 1 & 3 & 4 \\ 2 & 1 & 3 \\ 4 & 2 & 1 \end{bmatrix}$$

In [ ]:
```
n = len(A1)
I = np.identity(n)
ampliada = np.hstack((A1,I))

inden,I = separar_inversa(gauss_jordan(ampliada))
print("Matriz A:\n", np.array(A1))
print('matriz inversa:\n',I)
```

```
Matriz A:
 [[1 3 4]
 [2 1 3]
 [4 2 1]]
matriz inversa:
 [[-0.2  0.2  0.2]
 [ 0.4 -0.6  0.2]
 [ 0.   0.4 -0.2]]
```

$$A_2 = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 0 \end{bmatrix}$$

In [ ]:
```
n2 = len(A2)
I2 = np.identity(n2)
ampliada2 = np.hstack((A2,I2))
inden2,I2 = separar_inversa(gauss_jordan(ampliada2))
print("Matriz A2:\n", np.array(A2))
print('matriz inversa:\n',I2)
```

```
Matriz A2:
 [[1 2 3]
 [0 1 4]
 [5 6 0]]
matriz inversa:
 [[-24.  18.   5.]
 [ 20. -15.  -4.]
 [ -5.   4.   1.]]
```

$$A_3 = \begin{bmatrix} 4 & 2 & 1 \\ 2 & 1 & 3 \\ 1 & 3 & 4 \end{bmatrix}$$

In [ ]:
```
n3=len(A3)
I3=np.identity(n3)
```

```
ampliada3=np.hstack((A3,I3))
inden3,I3=separar_inversa(gauss_jordan(ampliada3))
print("Matriz A3:\n", np.array(A3))
print('matriz inversa:\n',I3)
```

```
Matriz A3:
 [[4 2 1]
 [2 1 3]
 [1 3 4]]
matriz inversa:
 [[ 0.2  0.2 -0.2]
 [ 0.2 -0.6  0.4]
 [-0.2  0.4  0. ]]
```

$$A_4 = \begin{bmatrix} 2 & 4 & 6 & 1 \\ 4 & 7 & 5 & -6 \\ 2 & 5 & 18 & 10 \\ 6 & 12 & 38 & 16 \end{bmatrix}$$

In [ ]:
```
n4 = len(A4)
I4 = np.identity(n4)
ampliada4 = np.hstack((A4,I4))
inden4,I4 = separar_inversa(gauss_jordan(ampliada4))
print("Matriz A4:\n", np.array(A4))
print('matriz inversa:\n',I4)
```

```
Matriz A4:
 [[ 2  4  6  1]
 [ 4  7  5 -6]
 [ 2  5 18 10]
 [ 6 12 38 16]]
matriz inversa:
 [[ 3.2         -1.71111111 -3.71111111  1.47777778]
 [-0.4          0.53333333  1.53333333 -0.73333333]
 [-0.8          0.28888889  0.28888889 -0.02222222]
 [ 1.          -0.44444444 -0.44444444  0.11111111]]
```

2. Calcule la descomposición LU para estas matrices y encuentre la solución para estos vectores de valores independientes b.

In [29]:
```
def matriz_aumentada(A: np.ndarray, b: np.ndarray) -> np.ndarray:
    """Construye la matriz aumentada de un sistema de ecuaciones lineales.

    ## Parameters

    ``A``: matriz de coeficientes.

    ``b``: vector de términos independientes.

    ## Return

    ``a``:

    """
```

```
    if not isinstance(A, np.ndarray):
        logging.debug("Convirtiendo A a numpy array.")
        A = np.array(A, dtype=float)
    if not isinstance(b, np.ndarray):
        b = np.array(b, dtype=float)
    assert A.shape[0] == b.shape[0], "Las dimensiones de A y b no coinciden."
    return np.hstack((A, b.reshape(-1, 1)))
```

In [30]:
```
def descomposicion_LU(A: np.ndarray) -> tuple[np.ndarray, np.ndarray]:
    """Realiza la descomposición LU de una matriz cuadrada A.
    [IMPORTANTE] No se realiza pivoteo.

    ## Parameters

    ``A``: matriz cuadrada de tamaño n-by-n.

    ## Return

    ``L``: matriz triangular inferior.

    ``U``: matriz triangular superior. Se obtiene de la matriz ``A`` después de apl
    """

    A = np.array(
        A, dtype=float
    )  # convertir en float, porque si no, puede convertir como entero

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

    L = np.zeros((n, n), dtype=float)

    for i in range(0, n):  # loop por columna

        # --- deterimnar pivote
        if A[i, i] == 0:
            raise ValueError("No existe solución única.")

        # --- Eliminación: loop por fila
        L[i, i] = 1
        for j in range(i + 1, n):
            m = A[j, i] / A[i, i]
            A[j, i:] = A[j, i:] - m * A[i, i:]

            L[j, i] = m

        logging.info(f"\n{A}")

    if A[n - 1, n - 1] == 0:
        raise ValueError("No existe solución única.")

    return L, A
```

In [31]:
```
def resolver_LU(L: np.ndarray, U: np.ndarray, b: np.ndarray) -> np.ndarray:
    """Resuelve un sistema de ecuaciones lineales mediante la descomposición LU.
```

```
    ## Parameters

    ``L``: matriz triangular inferior.

    ``U``: matriz triangular superior.

    ``b``: vector de términos independientes.

    ## Return

    ``solucion``: vector con la solución del sistema de ecuaciones lineales.

    """

    n = L.shape[0]

    # --- Sustitución hacia adelante
    logging.info("Sustitución hacia adelante")

    y = np.zeros((n, 1), dtype=float)

    y[0] = b[0] / L[0, 0]

    for i in range(1, n):
        suma = 0
        for j in range(i):
            suma += L[i, j] * y[j]
        y[i] = (b[i] - suma) / L[i, i]

    logging.info(f"y = \n{y}")

    # --- Sustitución hacia atrás
    logging.info("Sustitución hacia atrás")
    sol = np.zeros((n, 1), dtype=float)

    sol[-1] = y[-1] / U[-1, -1]

    for i in range(n - 2, -1, -1):
        logging.info(f"i = {i}")
        suma = 0
        for j in range(i + 1, n):
            suma += U[i, j] * sol[j]
        logging.info(f"suma = {suma}")
        logging.info(f"U[i, i] = {U[i, i]}")
        logging.info(f"y[i] = {y[i]}")
        sol[i] = (y[i] - suma) / U[i, i]

    logging.debug(f"x = \n{sol}")
    return y, sol
```

$$b_1 = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}, b_2 = \begin{bmatrix} 3 \\ -5 \\ 2 \end{bmatrix}, b_3 = \begin{bmatrix} 7 \\ 8 \\ -1 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} 1 & 3 & 4 \\ 2 & 1 & 3 \\ 4 & 2 & 1 \end{bmatrix}$$

In [ ]:
```
L,U=descomposicion_LU(A1)
y,sol=resolver_LU(L,U,b1)
print('soluciones intermedias y:\n',y)
print('soluciones finales x:\n',sol)

comprobar= np.dot(np.array(A1),sol)
print('comprobacion Ax=b:\n',comprobar)
```

```
soluciones intermedias y:
 [[1.]
 [0.]
 [0.]]
soluciones finales x:
 [[ 1.]
 [-0.]
 [-0.]]
comprobacion Ax=b:
 [[1.]
 [2.]
 [4.]]
```

In [ ]:
```
L,U=descomposicion_LU(A1)
y,sol=resolver_LU(L,U,b2)
print('soluciones intermedias y:\n',y)
print('soluciones finales x:\n',sol)

comprobar= np.dot(np.array(A1),sol)
print('comprobacion Ax=b:\n',comprobar)
```

```
soluciones intermedias y:
 [[  3.]
 [-11.]
 [ 12.]]
soluciones finales x:
 [[-1.2]
 [ 4.6]
 [-2.4]]
comprobacion Ax=b:
 [[ 3.]
 [-5.]
 [ 2.]]
```

In [ ]:
```
L,U=descomposicion_LU(A1)
y,sol=resolver_LU(L,U,b3)
```

```
print('soluciones intermedias y:\n',y)
print('soluciones finales x:\n',sol)

comprobar= np.dot(np.array(A1),sol)
print('comprobacion Ax=b:\n',comprobar)
```

```
soluciones intermedias y:
 [[  7.]
 [ -6.]
 [-17.]]
soluciones finales x:
 [[ 8.8817842e-16]
 [-2.2000000e+00]
 [ 3.4000000e+00]]
comprobacion Ax=b:
 [[ 7.]
 [ 8.]
 [-1.]]
```

$$b_1 = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}, b_2 = \begin{bmatrix} 3 \\ -5 \\ 2 \end{bmatrix}, b_3 = \begin{bmatrix} 7 \\ 8 \\ -1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 0 \end{bmatrix}$$

In [40]:
```
L,U=descomposicion_LU(A2)
y,sol=resolver_LU(L,U,b1)
print('soluciones intermedias y:\n',y)
print('soluciones finales x:\n',sol)

comprobar= np.dot(np.array(A2),sol)
print('comprobacion Ax=b:\n',comprobar)
```

```
soluciones intermedias y:
 [[1.]
 [2.]
 [7.]]
soluciones finales x:
 [[ 32.]
 [-26.]
 [  7.]]
comprobacion Ax=b:
 [[1.]
 [2.]
 [4.]]
```

In [41]:
```
L,U=descomposicion_LU(A2)
y,sol=resolver_LU(L,U,b2)
print('soluciones intermedias y:\n',y)
print('soluciones finales x:\n',sol)
```

```
comprobar= np.dot(np.array(A2),sol)
print('comprobacion Ax=b:\n',comprobar)
```

```
soluciones intermedias y:
 [[  3.]
 [ -5.]
 [-33.]]
soluciones finales x:
 [[-152.]
 [ 127.]
 [ -33.]]
comprobacion Ax=b:
 [[ 3.]
 [-5.]
 [ 2.]]
```

In [42]:
```
L,U=descomposicion_LU(A2)
y,sol=resolver_LU(L,U,b3)
print('soluciones intermedias y:\n',y)
print('soluciones finales x:\n',sol)

comprobar= np.dot(np.array(A2),sol)
print('comprobacion Ax=b:\n',comprobar)
```

```
soluciones intermedias y:
 [[ 7.]
 [ 8.]
 [-4.]]
soluciones finales x:
 [[-29.]
 [ 24.]
 [ -4.]]
comprobacion Ax=b:
 [[ 7.]
 [ 8.]
 [-1.]]
```

$$b_1 = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}, b_2 = \begin{bmatrix} 3 \\ -5 \\ 2 \end{bmatrix}, b_3 = \begin{bmatrix} 7 \\ 8 \\ -1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 4 & 2 & 1 \\ 2 & 1 & 3 \\ 1 & 3 & 4 \end{bmatrix}$$

In [46]:
```
L,U=descomposicion_LU(A3)
y,sol=resolver_LU(L,U,b1)
print('soluciones intermedias y:\n',y)
print('soluciones finales x:\n',sol)
comprobar= np.dot(np.array(A3),sol)
print('comprobacion Ax=b:\n',comprobar)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[46], line 1
----> 1 L,U=descomposicion_LU(A3)
      2 y,sol=resolver_LU(L,U,b1)
      3 print('soluciones intermedias y:\n',y)

Cell In[30], line 29, in descomposicion_LU(A)
     25 for i in range(0, n):  # loop por columna
     26
     27     # --- deterimnar pivote
     28     if A[i, i] == 0:
---> 29         raise ValueError("No existe solución única.")
     31     # --- Eliminación: loop por fila
     32     L[i, i] = 1

ValueError: No existe solución única.
```

In [44]:
```
L,U=descomposicion_LU(A3)
y,sol=resolver_LU(L,U,b2)
print('soluciones intermedias y:\n',y)
print('soluciones finales x:\n',sol)
comprobar= np.dot(np.array(A3),sol)
print('comprobacion Ax=b:\n',comprobar)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[44], line 1
----> 1 L,U=descomposicion_LU(A3)
      2 y,sol=resolver_LU(L,U,b2)
      3 print('soluciones intermedias y:\n',y)

Cell In[30], line 29, in descomposicion_LU(A)
     25 for i in range(0, n):  # loop por columna
     26
     27     # --- deterimnar pivote
     28     if A[i, i] == 0:
---> 29         raise ValueError("No existe solución única.")
     31     # --- Eliminación: loop por fila
     32     L[i, i] = 1

ValueError: No existe solución única.
```

In [45]:
```
L,U=descomposicion_LU(A3)
y,sol=resolver_LU(L,U,b3)
print('soluciones intermedias y:\n',y)
print('soluciones finales x:\n',sol)
comprobar= np.dot(np.array(A3),sol)
print('comprobacion Ax=b:\n',comprobar)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[45], line 1
----> 1 L,U=descomposicion_LU(A3)
      2 y,sol=resolver_LU(L,U,b3)
      3 print('soluciones intermedias y:\n',y)

Cell In[30], line 29, in descomposicion_LU(A)
     25 for i in range(0, n):  # loop por columna
     26
     27     # --- deterimnar pivote
     28     if A[i, i] == 0:
---> 29         raise ValueError("No existe solución única.")
     31     # --- Eliminación: loop por fila
     32     L[i, i] = 1

ValueError: No existe solución única.
```

Con A_3 no es posible realizar la factorización LU, dado que en algún momento de la ejecución una de las filas posee un cero en el elemento de la diagonal principal.

$$c_1 = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 5 \end{bmatrix}, c_2 = \begin{bmatrix} 3 \\ -5 \\ 2 \\ 6 \end{bmatrix}, c_3 = \begin{bmatrix} 7 \\ 8 \\ -1 \\ 0 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} 2 & 4 & 6 & 1 \\ 4 & 7 & 5 & -6 \\ 2 & 5 & 18 & 10 \\ 6 & 12 & 38 & 16 \end{bmatrix}$$

In [50]:
```
L,U=descomposicion_LU(A4)
y,sol=resolver_LU(L,U,c1)
print('soluciones intermedias y:\n',y)
print('soluciones finales x:\n',sol)
comprobar= np.dot(np.array(A4),sol)
print('comprobacion Ax=b:\n',comprobar)
```

```
soluciones intermedias y:
 [[  1.]
 [  0.]
 [  3.]
 [-10.]]
soluciones finales x:
 [[-7.67777778]
 [ 3.13333333]
 [ 0.82222222]
 [-1.11111111]]
comprobacion Ax=b:
 [[1.]
 [2.]
 [4.]
 [5.]]
```

In [49]:
```
L,U=descomposicion_LU(A4)
y,sol=resolver_LU(L,U,c2)
print('soluciones intermedias y:\n',y)
print('soluciones finales x:\n',sol)
comprobar= np.dot(np.array(A4),sol)
print('comprobacion Ax=b:\n',comprobar)
```

```
soluciones intermedias y:
 [[  3.]
 [-11.]
 [-12.]
 [ 45.]]
soluciones finales x:
 [[19.6]
 [-5.2]
 [-3.4]
 [ 5. ]]
comprobacion Ax=b:
 [[ 3.]
 [-5.]
 [ 2.]
 [ 6.]]
```

In [48]:
```
L,U=descomposicion_LU(A4)
y,sol=resolver_LU(L,U,c3)
print('soluciones intermedias y:\n',y)
print('soluciones finales x:\n',sol)
comprobar= np.dot(np.array(A4),sol)
print('comprobacion Ax=b:\n',comprobar)
```

```
soluciones intermedias y:
 [[  7.]
 [ -6.]
 [-14.]
 [ 35.]]
soluciones finales x:
 [[12.42222222]
 [-0.06666667]
 [-3.57777778]
 [ 3.88888889]]
comprobacion Ax=b:
 [[ 7.00000000e+00]
 [ 8.00000000e+00]
 [-1.00000000e+00]
 [ 1.42108547e-14]]
```