DESCOMPOSICION PLU

Seudocódigo

```
function operacionFila(A, fm, fp, factor):
    A[fm, :] = A[fm, :] - factor * A[fp, :]
```

```
function intercambiaFil(A, fi, fj):
    intercambiar filas fi y fj en A
```

```
function PLU_decomposition(A):
    n = número de filas de A
    P = matriz identidad de tamaño n
    L = matriz de ceros de tamaño n x n
    U = copia de A

    for k desde 0 hasta n-1:
        pivot = índice del valor máximo absoluto en U[k:n, k] + k
        if pivot != k:
            intercambiar filas k y pivot en P
            intercambiar filas k y pivot en U
            if k >= 1:
                intercambiar filas k y pivot en L

        for j desde k+1 hasta n-1:
            L[j, k] = U[j, k] / U[k, k]
            operacionFila(U, j, k, L[j, k])

    establecer 1 en la diagonal de L
    return P, L, U
```

```
function sustRegresiva(A, b):
    N = número de filas de b
    x = vector de ceros de tamaño N x 1

    for i desde N-1 hasta 0:
        x[i, 0] = (b[i, 0] - producto_punto(A[i, i+1:N], x[i+1:N, 0])) / A[i, i]

    return x
```

```
function sustProgresiva(A, b):
    N = número de filas de b
    x = vector de ceros de tamaño N x 1

    for i desde 0 hasta N-1:
        x[i, 0] = (b[i, 0] - producto_punto(A[i, 0:i], x[0:i, 0])) / A[i, i]

    return x
```

```
function PLU_solucion(A, b):
    P, L, U = PLU_decomposition(A)
    Pb = producto_matriz(P, b)
    y = sustProgresiva(L, Pb)
    x = sustRegresiva(U, y)
    return P, L, U, x
```

```
function comprobar_PLU(A, P, L, U):
    PA = producto_matriz(P, A)
    LU = producto_matriz(L, U)
    print "Matriz PA:"
    print PA
    print "Matriz LU:"
    print LU
    return matrices_cercanas(PA, LU)
```

```
function comprobar_solucion(A, x, b):
    Ax = producto_matriz(A, x)
    print "Ax calculado:"
    print Ax
    print "b original:"
    print b
    return matrices_cercanas(Ax, b)
```

```
A = matriz([
    [0, -1, 4],
    [2, 1, 1],
    [1, 1, -2]
])

b = vector([
    [5],
    [-2],
    [9]
])

# Descomposición y solución
P, L, U = PLU_decomposition(A)
print "Matriz P:"
print P
print "Matriz L:"
print L
print "Matriz U:"
print U
```

```
P, L, U, x = PLU_solucion(A, b)
print "Solución del sistema AX = b:"
print x

# Verificación de la descomposición PLU
es_PLU_correcto = comprobar_PLU(A, P, L, U)
print "¿PA = LU? ", es_PLU_correcto

# Verificación de la solución del sistema
es_solucion_correcta = comprobar_solucion(A, x, b)
print "¿AX = b? ", es_solucion_correcta
```

IMPLEMENTADO EN PYTHON

```python
def operacionFila(A, fm, fp, factor):
    # filafm = filafm - factor * filafp
    # A[fm, :] hace referencia a la fila fm de la matriz A
    # factor * A[fp, :] es la fila fp multiplicada por el factor
    # Se resta factor * fila fp de la fila fm en la matriz A
    A[fm, :] = A[fm, :] - factor * A[fp, :]

def intercambiaFil(A, fi, fj):
    # Intercambia las filas fi y fj en la matriz A
    # A[[fi, fj], :] selecciona las filas fi y fj
    # A[[fj, fi], :] reasigna las filas intercambiadas
    A[[fi, fj], :] = A[[fj, fi], :]
```


```python
def operacionFila(A, fm, fp, factor):
    # filafm = filafm - factor * filafp
    # A[fm, :] hace referencia a la fila fm de la matriz A
    # factor * A[fp, :] es la fila fp multiplicada por el factor
    # Se resta factor * fila fp de la fila fm en la matriz A
    A[fm, :] = A[fm, :] - factor * A[fp, :]

def intercambiaFil(A, fi, fj):
    # Intercambia las filas fi y fj en la matriz A
    # A[[fi, fj], :] selecciona las filas fi y fj
    # A[[fj, fi], :] reasigna las filas intercambiadas
    A[[fi, fj], :] = A[[fj, fi], :]
```


```python

def PLU_decomposicion(A):
    n = A.shape[0]  # Número de filas de la matriz A
    P = np.eye(n)   # Matriz identidad de tamaño n (matriz de permutación)
    L = np.zeros((n, n))  # Matriz de ceros de tamaño n x n (matriz triangular inferior)
    U = A.copy()    # Hacemos una copia de A para no modificar la original (matriz triangular superior)

    for k in range(n):  # Iteramos sobre cada columna
        # Encontramos el pivote (el valor absoluto más grande en la subcolumna)
        pivot = np.argmax(np.abs(U[k:, k])) + k
        if pivot != k:
            # Si el pivote no está en la fila actual, intercambiamos las filas
            intercambiaFil(P, k, pivot)
            intercambiaFil(U, k, pivot)
            if k >= 1:
                intercambiaFil(L, k, pivot)

        for j in range(k + 1, n):
            # Calculamos el multiplicador para la fila j
            L[j, k] = U[j, k] / U[k, k]
            # Realizamos la eliminación gaussiana
            operacionFila(U, j, k, L[j, k])

```

```
233
234         # Establecemos 1 en la diagonal de L
235         np.fill_diagonal(L, 1)
236         return P, L, U  # Devolvemos las matrices P, L y U
237     def PLU_solucion(A, b):
238         # Resuelve el sistema A*x = b usando descomposición PLU
239         P, L, U = PLU_decomposicion(A)  # Obtenemos las matrices P, L y U
240         Pb = np.dot(P, b)  # Multiplicamos la matriz de permutación P por el vector b
241         y = sustProgresiva(L, Pb)  # Resolvemos L*y = P*b usando sustitución progresiva
242         x = sustRegresiva(U, y)  # Resolvemos U*x = y usando sustitución regresiva
243         return P, L, U, x  # Devolvemos P, L, U y el vector solución x
244
```

```
317 # Ejemplo de uso para DESCOMPOSICION PLU
318
319 A = np.array([[0, -1, 4], [2, 1, 1], [1, 1, -2]], dtype=float)
320 b = np.array([[5], [-2], [9]], dtype=float)
321 # Descomposición y solución
322
323 P, L, U = bib.PLU_decomposicion(A)
324 print("Matriz P:")
325 print(P)
326 print("Matriz L:")
327 print(L)
328 print("Matriz U:")
329 print(U)
330
331
332 P, L, U, x = bib.PLU_solucion(A, b)
333 print("Solución del sistema AX = b:", x)
334
```

```
332 P, L, U, x = bib.PLU_solucion(A, b)
333 print("Solución del sistema AX = b:", x)
334
335 def comprobar_PLU(A, P, L, U):
336     # Comprueba si PA = LU
337     PA = np.dot(P, A)  # Calculamos P*A
338     LU = np.dot(L, U)  # Calculamos L*U
339     print("Matriz PA:")
340     print(PA)  # Mostramos PA
341     print("Matriz LU:")
342     print(LU)  # Mostramos LU
343     return np.allclose(PA, LU)  # Verificamos si PA es aproximadamente igual a LU
344
345 def comprobar_solucion(A, x, b):
346     # Comprueba si Ax = b
347     Ax = np.dot(A, x)  # Calculamos A*x
348     print("Ax calculado:")
349     print(Ax)  # Mostramos A*x
350     print("b original:")
351     print(b)  # Mostramos b original
352     return np.allclose(Ax, b)  # Verificamos si A*x es aproximadamente igual a b
353
354 # Verificación de la descomposición PLU
355 es_PLU_correcto = comprobar_PLU(A, P, L, U)
356 print("¿PA = LU? ", es_PLU_correcto)
```

```
> & C:/Users/Usuario/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Use
rs/Usuario/Desktop/matematica_computacional/miprogramaprincipal.py
Matriz P:
[[0. 1. 0.]
 [1. 0. 0.]
 [0. 0. 1.]]
Matriz L:
[[ 1.   0.   0. ]
 [ 0.   1.   0. ]
 [ 0.5 -0.5  1. ]]
Matriz U:
[[ 2.   1.   1. ]
 [ 0.  -1.   4. ]
 [ 0.   0.  -0.5]]
Solución del sistema AX = b: [[  64.]
 [-105.]
 [ -25.]]
Matriz PA:
[[ 2.  1.  1.]
 [ 0. -1.  4.]
 [ 1.  1. -2.]]
Matriz LU:
[[ 2.  1.  1.]
 [ 0. -1.  4.]
 [ 1.  1. -2.]]
¿PA = LU?  True
Ax calculado:
[[ 5.]
 [-2.]
 [ 9.]]
b original:
[[ 5.]
 [-2.]
 [ 9.]]
¿AX = b?  True
PS C:\Users\Usuario\Desktop\matematica_computacional>
```