ESCUELA POLITECNICA NACIONAL

Materia: Métodos Numéricos

Nombre: Stiv Quishpe

link repositorio

https://github.com/stiv001/Tarea-10.git

Descomposición LU

CONJUNTO DE EJERCICIOS

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

Figure 1: image.png

Literal a)

```python
import numpy as np

A = [
    [2, -3],
    [3, -1]
]

B = [
    [1, 5],
    [2, 0]
]

C = np.matmul(A, B)
print(C)
```

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

Literal b)

```python

A = [
    [2, -3],
    [3, -1]
]

B = [
    [1, 5, -4],
    [-3, 2, 0]
]

C = np.matmul(A, B)
print(C)
```

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

Literal c)

```
A = [
    [2, -3, 1],
    [4, 3, 0],
    [5, 2, -4]
]

B = [
    [0, 1, -2],
    [1, 0, -1],
    [2, 3, -2]
]

C = np.matmul(A, B)
print(C)
```

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

Literal d)

```
A = [
    [2, 1, 2],
    [-2, 3, 0],
    [2, -1, 3]
]

B = [
    [1, -2],
    [-4, 1],
    [0, 2]
]

C = np.matmul(A, B)
print(C)
```

```
[[ -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.** $\begin{bmatrix} 4 & 0 & 0 & 0 \\ 6 & 7 & 0 & 0 \\ 9 & 11 & 1 & 0 \\ 5 & 4 & 1 & 1 \end{bmatrix}$

Figure 2: image.png

```python
import numpy as np

A = [
    [4, 2, 6],
    [3, 0, 7],
    [-2, -1, -3]
]

try:
    B = np.linalg.inv(A)
    print("La inversa de la matriz A es:")
    print(B)
except np.linalg.LinAlgError as e:
    print("No se puede calcular la inversa de la matriz A.")
```

```
No se puede calcular la inversa de la matriz A.
```

Literal b)

```python
A = [
    [1, 2, 0],
```

```python
    [2, 1, -1],
    [3, 1, 1]
]

try:
    B = np.linalg.inv(A)
    print("La inversa de la matriz A es:")
    print(B)
except np.linalg.LinAlgError as e:
    print("No se puede calcular la inversa de la matriz A.")
```

```
La inversa de la matriz A es:
[[-0.25   0.25   0.25 ]
 [ 0.625 -0.125 -0.125]
 [ 0.125 -0.625  0.375]]
```

Literal c)

```python
A = [
    [1, 1, -1, 1],
    [1, 2, -4, -2],
    [2, 1, 1, 5],
    [-1, 0, -2, -4]
]

try:
    B = np.linalg.inv(A)
    print("La inversa de la matriz A es:")
    print(B)
except np.linalg.LinAlgError as e:
    print("No se puede calcular la inversa de la matriz A.")
```

```
No se puede calcular la inversa de la matriz A.
```

Literal d)

```
A = [
    [4, 0, 0, 0],
    [6, 7, 0, 0],
    [9, 11, 1, 0],
    [5, 4, 1, 1]
]

try:
    B = np.linalg.inv(A)
    print("La inversa de la matriz A es:")
    print(B)
except np.linalg.LinAlgError as e:
    print("No se puede calcular la inversa de la matriz A.")
```

```
La inversa de la matriz A es:
[[ 2.50000000e-01  6.16790569e-18  0.00000000e+00  0.00000000e+00]
 [-2.14285714e-01  1.42857143e-01 -0.00000000e+00 -0.00000000e+00]
 [ 1.07142857e-01 -1.57142857e+00  1.00000000e+00 -0.00000000e+00]
 [-5.00000000e-01  1.00000000e+00 -1.00000000e+00  1.00000000e+00]]
```

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

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

Figure 3: image.png

```
A = [
    [1, -1, 2, -1],
    [1, 0, -1, 1],
```

```
    [2, 1, 3, -4],
    [0, -1, 1, -1]
]

b1 = [6, 4, -2, 5]

b2 = [1, 1, 2, -1]

B1 = np.linalg.solve(A, b1)
B2 = np.linalg.solve(A, b2)
print(B1)
print(B2)
```

```
[ 3. -6. -2. -1.]
[1. 1. 1. 1.]
```

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

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

Figure 4: image.png

$$A = \begin{pmatrix} 1 & -1 & \alpha \\ 2 & 2 & 1 \\ 0 & \alpha & -3/2 \end{pmatrix}$$

Como $\det(A) = 0 \Rightarrow$ la matriz A es singular por lo tanto.

$$1\cdot\left(2\left(-3/2\right) - \alpha\right) - (-1)\left(2\left(-3/2\right) - 0\right) + \alpha\left(2\alpha - 0\right) = 0$$

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

$$-3 - \alpha - 3 + 2\alpha^2 = 0$$

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

$$\alpha = \frac{-b \pm \sqrt{b - 4ac}}{2a} \Rightarrow \begin{matrix} \alpha_1 = -1,5 \\ \alpha_2 = 2. \end{matrix}$$

Figure 5: image.png

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

Figure 6: image.png

Literal a)

```
A1 = [
```

```
    [1, 0, 0],
    [2, 1, 0],
    [-1, 0, 1]
]

A2 = [
    [2, 3, -1],
    [0, -2, 1],
    [0, 0, 3]
]

b = [2, -1, 1]

C = np.matmul(A1, A2)
C = np.linalg.solve(C, b)
print(C)
```

```
[-3.  3.  1.]
```

Literal b)

```
A1 = [
    [2, 0, 0],
    [-1, 1, 0],
    [3, 2, -1]
]

A2 = [
    [1, 1, 1],
    [0, 1, 2],
    [0, 0, 1]
]

b = [-1, 3, 0]

C = np.matmul(A1, A2)
C = np.linalg.solve(C, b)
print(C)
```

```
[ 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.

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

Figure 7: image.png

```python
def descomposicion_LU(A: np.ndarray) -> tuple[np.ndarray, np.ndarray]:
    A = np.array(
        A, dtype=float
    )

    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):
        if A[i, i] == 0:
            raise ValueError("No existe solucion unica.")

        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

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

    return L, A
```

Literal a)

```python
A = [
    [2, -1, 1],
    [3, 3, 9],
    [3, 3, 5]
]

L, U = descomposicion_LU(A)
print(L)
print()
print(U)
```

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

[[ 2.  -1.   1. ]
 [ 0.   4.5  7.5]
 [ 0.   0.  -4. ]]
```

Literal b)

```python
A = [
    [1.012, -2.132, 3.104],
    [-2.132, 4.096, -7.013],
    [3.104, -7.013, 0.014]
]

L, U = descomposicion_LU(A)
print(L)
print()
print(U)
```

```
[[ 1.          0.          0.        ]
 [-2.10671937  1.          0.        ]
 [ 3.06719368  1.19775553  1.        ]]
```

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

Literal c)

```
A = [
    [2, 0, 0, 0],
    [1, 1.5, 0, 0],
    [0, -3, 0.5, 0],
    [2, -2, 1, 1]
]

L, U = descomposicion_LU(A)
print(L)
print()
print(U)
```

```
[[ 1.          0.          0.          0.        ]
 [ 0.5         1.          0.          0.        ]
 [ 0.         -2.          1.          0.        ]
 [ 1.         -1.33333333  2.          1.        ]]

[[2.  0.  0.  0. ]
 [0.  1.5 0.  0. ]
 [0.  0.  0.5 0. ]
 [0.  0.  0.  1. ]]
```

Literal d)

```
A = [
    [2.1756, 4.0231, -2.1732, 5.1967],
    [-4.0231, 6, 0, 1.1973],
    [-1, -5.2107, 1.1111, 0],
    [6.0235, 7, 0, -4.1561]
]

L, U = descomposicion_LU(A)
```

```
print(L)
print()
print(U)
```

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

[[ 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]]
```

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.

Figure 8: image.png

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

Figure 9: image-2.png

```
def eliminacion_gaussiana(A: np.ndarray) -> np.ndarray:
    if not isinstance(A, np.ndarray):
```

```python
    A = np.array(A)
assert A.shape[0] == A.shape[1] - 1, "La matriz A debe ser de tamanio n-by-(n+1)."
n = A.shape[0]

for i in range(0, n - 1):

    p = None
    for pi in range(i, n):
        if A[pi, i] == 0:
            continue

        if p is None:
            p = pi
            continue

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

    if p is None:
        raise ValueError("No existe solucion unica.")

    if p != i:
        _aux = A[i, :].copy()
        A[i, :] = A[p, :].copy()
        A[p, :] = _aux

    for j in range(i + 1, n):
        m = A[j, i] / A[i, i]
        A[j, i:] = A[j, i:] - m * A[i, i:]


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

    print(f"\n{A}")
solucion = np.zeros(n)
solucion[n - 1] = A[n - 1, n] / A[n - 1, n - 1]

for i in range(n - 2, -1, -1):
    suma = 0
    for j in range(i + 1, n):
        suma += A[i, j] * solucion[j]
```

```
        solucion[i] = (A[i, n] - suma) / A[i, i]

    return solucion
```

Literal a)

```
A = [
    [2, -1, 1, -1],
    [3, 3, 9, 0],
    [3, 3, 5, 4]
]

x = eliminacion_gaussiana(A)
print(x)
```

```
[ 1.  2. -1.]
```

Literal b)

```
A = [
    [1.012, -2.132, 3.104, 1.984],
    [-2.132, 4.096, -7.013, -5.049],
    [3.104, -7.013, 0.014, -3.895]
]

x = eliminacion_gaussiana(A)
print(x)
```

```
[1. 1. 1.]
```

Literal c)

```python
A = [
    [2, 0, 0, 0, 3],
    [1, 1.5, 0, 0, 4.5],
    [0, -3, 0.5, 0, -6.6],
    [2, -2, 1, 1, 0.8]
]

x = eliminacion_gaussiana(A)
print(x)
```

```
[ 1.5  2.  -1.2  3. ]
```

Literal d)

```python
A = [
    [2.1756, 4.0231, -2.1732, 5.1967, 17.102],
    [-4.0231, 6, 0, 1.1973, -6.1593],
    [-1, -5.2107, 1.1111, 0, 3.0004],
    [6.0235, 7, 0, -4.1561, 0]
]

x = eliminacion_gaussiana(A)
print(x)
```

```
[2.9398512  0.0706777  5.67773512 4.37981223]
```