MetodosNumericosT10

February 5, 2025

1 ESCUELA POLITÉCNICA NACIONAL

1.1 MÉTODOS NUMÉRICOS

1.1.1 TAREA 10

David Alejandro Puga Novoa - GR1CC - 05/02/2025

1.1.2 Ejercicio 1

Realice las siguientes multiplicaciones matriz-matriz:

a)

[1]:
```
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]]
```

b)

[2]:
```
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]]
```

c)

[3]:
```
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]]
```

d)

[4]:
```
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]]
```

1.1.3 Ejercicio 2

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

a)

[5]:
```python
A = [
    [4, 2, 6],
    [3, 0, 7],
    [-2, -1, -3]
]

B = np.linalg.inv(A)
print(B)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[5], line 7
      1 A = [
      2     [4, 2, 6],
      3     [3, 0, 7],
      4     [-2, -1, -3]
      5 ]
----> 7 B = np.linalg.inv(A)
      8 print(B)

File ~/miniforge3/envs/iccd332/lib/python3.11/site-packages/numpy/linalg/_linalg.
↪py:615, in inv(a)
    612 signature = 'D->D' if isComplexType(t) else 'd->d'
    613 with errstate(call=_raise_linalgerror_singular, invalid='call',
    614               over='ignore', divide='ignore', under='ignore'):
--> 615     ainv = _umath_linalg.inv(a, signature=signature)
    616 return wrap(ainv.astype(result_t, copy=False))

File ~/miniforge3/envs/iccd332/lib/python3.11/site-packages/numpy/linalg/_linalg.
↪py:104, in _raise_linalgerror_singular(err, flag)
    103 def _raise_linalgerror_singular(err, flag):
--> 104     raise LinAlgError("Singular matrix")

LinAlgError: Singular matrix
```

b)

[6]:
```python
A = [
    [1, 2, 0],
    [2, 1, -1],
    [3, 1, 1]
]
```

```
B = np.linalg.inv(A)
print(B)
```

```
[[-0.25   0.25   0.25 ]
 [ 0.625 -0.125 -0.125]
 [ 0.125 -0.625  0.375]]
```

c)

[7]:
```
A = [
    [1, 1, -1, 1],
    [1, 2, -4, -2],
    [2, 1, 1, 5],
    [-1, 0, -2, -4]
]

B = np.linalg.inv(A)
print(B)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[7], line 8
      1 A = [
      2     [1, 1, -1, 1],
      3     [1, 2, -4, -2],
      4     [2, 1, 1, 5],
      5     [-1, 0, -2, -4]
      6 ]
----> 8 B = np.linalg.inv(A)
      9 print(B)

File ~/miniforge3/envs/iccd332/lib/python3.11/site-packages/numpy/linalg/_linalg.
↪py:615, in inv(a)
    612 signature = 'D->D' if isComplexType(t) else 'd->d'
    613 with errstate(call=_raise_linalgerror_singular, invalid='call',
    614               over='ignore', divide='ignore', under='ignore'):
--> 615     ainv = _umath_linalg.inv(a, signature=signature)
    616 return wrap(ainv.astype(result_t, copy=False))

File ~/miniforge3/envs/iccd332/lib/python3.11/site-packages/numpy/linalg/_linalg.
↪py:104, in _raise_linalgerror_singular(err, flag)
    103 def _raise_linalgerror_singular(err, flag):
--> 104     raise LinAlgError("Singular matrix")

LinAlgError: Singular matrix
```

d)

[8]:
```
A = [
    [4, 0, 0, 0],
    [6, 7, 0, 0],
    [9, 11, 1, 0],
    [5, 4, 1, 1]
]

B = np.linalg.inv(A)
print(B)
```

```
[[ 0.25        0.          0.          0.        ]
 [-0.21428571  0.14285714 -0.         -0.        ]
 [ 0.10714286 -1.57142857  1.         -0.        ]
 [-0.5         1.         -1.          1.        ]]
```

1.1.4 Ejercicio 3

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

[9]:
```
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.]
```

1.1.5 Ejercicio 4

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

Se puede obtener α si calculamos el determinante de esta matriz y la igualamos a cero, entonces:

$$det(A) = -\alpha(1-2\alpha) - \frac{3}{2}(2+2)$$

$$det(A) = 2\alpha^2 - \alpha - 6 = 0$$

$$\alpha = 2 - - - - - - - \alpha = -\frac{3}{2}$$

Si α tiene alguno de los valores del conjunto: $\{-\frac{3}{2}, 2\}$, entonces el sistema no tiene solucion.

Si α, para este momento es igual a cero, entonces existen soluciones infinitas.

Esta es la matriz reducida, y el valor $(-\frac{1}{2} - 2\alpha)$ es lo que nos interesa.

$$-\frac{1}{2} - 2\alpha$$

Si α tiene alguno de los valores del conjunto $\{-\frac{1}{4}\}$, entonces el sistema tiene soluciones infinitas.

La respuesta es que α solo puede usar valores reales menos los del conjunto $\{-\frac{3}{2}, -\frac{1}{4}, 2\}$.

1.1.6 Ejercicio 5

Resuelva los siguientes sistemas lineales:

a)

[10]:
```
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.]
```

b)

[12]:
```
A1 = [
    [2, 0, 0],
    [-1, 1, 0],
    [3, 2, -1]
]

A2 = [
```

```python
    [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]
```

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

Primero definimos la funcion para la descomposición por LU:

[13]:
```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):  # loop por columna

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

        # --- 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

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

    return L, A
```

A continuación los ejercicios:

a)

[15]:
```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. ]]
```

b)

[16]:
```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]]
```

c)

[17]:
```python
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. ]]
```

d)

[18]:
```
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]]
```

Ejercicio 7 Modifique el algoritmo de eliminación gaussiana de tal forma que se pueda utilizar para resolver un sistema lineal usando la descomposicion LU y, a continuación, resuelva los siguientes sistemas lineales.

Entonces realizamos esta modificación en una función de eliminación gaussiana:

[19]:
```
def eliminacion_gaussiana(A: np.ndarray) -> np.ndarray:
    if not isinstance(A, np.ndarray):
        A = np.array(A)
```

```python
    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):  # 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 solucion unica.")

        if p != i:
            # swap rows
            _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
```

Y procedemos a la resolución de cada literal:

a)

[20]:
```
A = [
    [2, -1, 1, -1],
    [3, 3, 9, 0],
    [3, 3, 5, 4]
]

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

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

b)

[22]:
```
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.]
```

c)

[21]:
```
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. ]
```

d)

[23]:
```
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]
```

[ ]:

```

```

Enlace al repositorio: https://github.com/Vidcito/Metodos_Numericos/blob/main/Tareas/MetodosNumericosT1