practica1

February 15, 2021

1 Práctica 1: Introducción a numpy

2 Importar módulos necesarios

[2]:
```
import numpy as np
```

3 1 Aplicar la ley D'Hondt

3.1 Implementación

[3]:
```
def ley_dhondt(votes: np.ndarray, n_seats: int):
    """
    Dado un array de votos(unidimensional, cada valor son los votos de un␣
↪partido),
    y el número de escaños, aplica La ley D'Hondt y devuelve otro vector con el
    número de escaños de cada partido. El proceso de adjudicación queda␣
↪documentado
    en las matrices seats_per_iter(escaños adjudicados) y␣
↪quotient_per_iter(cociente
    tras adjudicar i escaños).
    """
    seats_per_iter = np.zeros((n_seats, votes.shape[0]), dtype=int)
    quotient_per_iter = np.zeros((n_seats, votes.shape[0]))

    quotient_per_iter[0,] = votes

    for i in range(n_seats):
        quotient_per_iter[i] = votes/(seats_per_iter[i-1]+1)
        winner = np.argmax(quotient_per_iter[i])
        seats_per_iter[i] = seats_per_iter[i-1]
        seats_per_iter[i, winner] += 1
    return seats_per_iter[-1]
```

3.2 Desde archivo

[75]:
```
def ley_dhondt_from_file(filename: str):
    """
    Dado un nombre de archivo, lee el número de escaños y los votos y devuelve␣
↪los
    escaños de cada partido. La primera línea debe ser el número de escaños,␣
↪después
    irán los votos de cada partido en una línea
    """
    with open(filename, 'r') as file:
        seats = int(file.readline())
        votes = [int(line) for line in file]
    print(f"Distributing {seats} seats between {len(votes)} parties with votes:
↪")
    for i,v in enumerate(votes):
        print(f"\t Party {i+1}: {v} votes")

    result = ley_dhondt(np.asarray(votes), seats)
    print("\nResult of the election:")
    for party, seats in enumerate(result):
        print(f"\t Party {party+1}: {seats} seats")
```

[76]:
```
ley_dhondt_from_file('votes.txt')
```

```
Distributing 7 seats between 5 parties with votes:
         Party 1: 340000 votes
         Party 2: 280000 votes
         Party 3: 160000 votes
         Party 4: 60000 votes
         Party 5: 15000 votes

Result of the election:
         Party 1: 3 seats
         Party 2: 3 seats
         Party 3: 1 seats
         Party 4: 0 seats
         Party 5: 0 seats
```

3.3 Por consola

[81]:
```
def ley_dhondt_from_console():
    """
    Reads the number of seats and votes from user input and returns
    the result of the election
    """
```

```
    seats = int(input('Number of seats: '))
    parties = int(input('Number of parties: '))
    votes = []
    for party in range(parties):
        votes.append(input(f'Votes of party {party+1}: '))

    print(f"Distributing {seats} seats between {len(votes)} parties with votes:
↪")
    for i,v in enumerate(votes):
        print(f"\t Party {i+1}: {v} votes")

    result = ley_dhondt(np.asarray(votes, dtype=float), seats)
    print("\nResult of the election:")
    for party, seats in enumerate(result):
        print(f"\t Party {party+1}: {seats} seats")
```

[83]:
```
ley_dhondt_from_console()
```

```
Number of seats: 3
Number of parties: 2
Votes of party 1: 10000
Votes of party 2: 5001
Distributing 3 seats between 2 parties with votes:
         Party 1: 10000 votes
         Party 2: 5001 votes

Result of the election:
         Party 1: 2 seats
         Party 2: 1 seats
```

4 2 Operaciones con matriz de reales aleatoria

4.1 Generar matriz aleatoria

[136]:
```
def random_float_matrix():
    matrix = np.random.rand(
        int(input('Number of rows: ')),
        int(input('Number of columns: ')),
    )
    return(matrix)
```

[137]:
```
matrix = random_float_matrix()
print(matrix)
```

```
Number of rows: 3
Number of columns: 5
[[0.3651667  0.31996508 0.89078706 0.22339432 0.41578467]
 [0.64813605 0.16740822 0.5889485  0.4594975  0.90067532]
 [0.73768761 0.45511015 0.07814468 0.74133254 0.73704595]]
```

4.2 2.1 Máximo y mínimo

[138]:
```
max_row, max_col = np.unravel_index(np.argmax(matrix), matrix.shape)
max_value = matrix[max_row, max_col]

min_row, min_col = np.unravel_index(np.argmin(matrix), matrix.shape)
min_value = matrix[min_row, min_col]


print(f"Maximum value found at position ({max_row},{max_col}): {max_value}")
print(f"Minimum value found at position ({min_row},{min_col}): {min_value}")
```

```
Maximum value found at position (1,4): 0.9006753150373428
Minimum value found at position (2,2): 0.07814467764725264
```

4.3 2.2 Ángulo de dos vectores

[5]:
```
size = int(input("Size of the vectors: "))

v1, v2 = np.empty((size,), dtype=float), np.empty((size,), dtype=float)

for i in range(size):
    v1[i] = float(input(f'Element {i} of vector 1: '))

for i in range(size):
    v2[i] = float(input(f'Element {i} of vector 2: '))

scalar_prod = np.dot(v1, v2)

angle = np.arccos(scalar_prod/(np.linalg.norm(v1)*np.linalg.norm(v2)))

print(f'The angle between vectors {v1} and {v2} is of {angle*180/np.pi:.4f}º␣
 ↪({angle:.4f} radians)')
```

```
Size of the vectors: 3
Element 0 of vector 1: 1
Element 1 of vector 1: 3
Element 2 of vector 1: 1
Element 0 of vector 2: 4
Element 1 of vector 2: 1
```

```
Element 2 of vector 2: 3
The angle between vectors [1. 3. 1.] and [4. 1. 3.] is of 53.7498º (0.9381
radians)
```

5 3 Operaciones con matriz de reales por teclado

5.1 Generar la matriz

[142]:
```
def read_float_matrix():
    matrix = np.empty(
        (int(input('Number of rows: ')), int(input('Number of columns: '))),
        dtype=float
    )

    for row in range(matrix.shape[0]):
        for col in range(matrix.shape[1]):
            matrix[row, col] = float(input(f'Element for position ({row},{col}):
 ↪ '))

    return(matrix)
```

[143]:
```
user_matrix = read_float_matrix()
print()
print(user_matrix)
```

```
Number of rows: 3
Number of columns: 3
Element for position (0,0): 12
Element for position (0,1): -21
Element for position (0,2): 32.4
Element for position (1,0): 4.232
Element for position (1,1): -2.34
Element for position (1,2): 4.242
Element for position (2,0): 10
Element for position (2,1): -12
Element for position (2,2): 6

[[ 12.    -21.     32.4  ]
 [  4.232  -2.34    4.242]
 [ 10.    -12.     6.   ]]
```

5.2 3.1 Máximo por filas y por columnas

[159]:
```
row_max = np.argmax(user_matrix, axis=1)

col_max = np.argmax(user_matrix, axis=0)

for i, max_index in enumerate(row_max):
    print(f'Maximum of row {i} ({user_matrix[i,:]}):  {user_matrix[i,␣
 ↪max_index]}')

print()
for i, max_index in enumerate(col_max):
    print(f'Maximum of column {i} ({user_matrix[:,i]}): ␣
 ↪{user_matrix[max_index, i]}')
```

```
Maximum of row 0 ([ 12.  -21.   32.4]):  32.4
Maximum of row 1 ([ 4.232 -2.34   4.242]):  4.242
Maximum of row 2 ([ 10. -12.   6.]):  10.0

Maximum of column 0 ([12.     4.232 10.   ]):  12.0
Maximum of column 1 ([-21.    -2.34 -12.  ]):  -2.34
Maximum of column 2 ([32.4    4.242  6.   ]):  32.4
```

5.3 3.2 Determinante

[155]:
```
def det(matrix):
    if matrix.shape[0] == matrix.shape[1]:
        determinant = np.linalg.det(matrix)

        print(f"Determinant of the matrix: {determinant}")
    else:
        print("Matrix is not square")

det(user_matrix)
```

```
Determinant of the matrix: -802.4616
```

5.4 3.3 Rango

[160]:
```
rank = np.linalg.matrix_rank(user_matrix)

print(f"Rank of the matrix: {rank}")
```

```
Rank of the matrix: 3
```

6 4 Operaciones con matriz de enteros por teclado

6.1 Generar la matriz

[170]:
```
def read_int_matrix():
    matrix = np.empty(
        (int(input('Number of rows: ')), int(input('Number of columns: '))),
        dtype=int
    )

    for row in range(matrix.shape[0]):
        for col in range(matrix.shape[1]):
            matrix[row, col] = int(input(f'Element for position ({row},{col}):␣
↪'))

    return(matrix)
```

[177]:
```
int_matrix = read_int_matrix()
print()
print(int_matrix)
```

```
Number of rows: 3
Number of columns: 3
Element for position (0,0): 1
Element for position (0,1): 3
Element for position (0,2): 1
Element for position (1,0): 4
Element for position (1,1): 3
Element for position (1,2): 1
Element for position (2,0): 1
Element for position (2,1): 5
Element for position (2,2): 1

[[1 3 1]
 [4 3 1]
 [1 5 1]]
```

6.2 4.1 Moda de la matriz

[185]:
```
values, frequencies = np.unique(int_matrix, return_counts=True)

index = np.argmax(frequencies)

mode = values[index]
```

```
print(f'The mode of the matrix is {mode}, with {frequencies[index]}␣
 ↪appearences')
```

```
The mode of the matrix is 1, with 5 appearences
```

6.3 4.2 Media de la matriz

[188]:
```
print(f'The mean of the matrix is {int_matrix.mean()}')
```

```
The mean of the matrix is 2.2222222222222223
```

6.4 5 Operaciones con matriz de reales desde archivo

6.4.1 Leer la matriz

El fichero tendrá la estructura:

```
A1,1 A2,1 .. Am,1
A1,2 ..... Am,2
.     .        .
.       .      .
.         .    .
An,1 ..... Am,n
```

[40]:
```
def read_from_file(filename: str):
    with open(filename, 'r') as file:
        lines = [line.strip().split(None) for line in file]

    matrix = np.asarray(lines, dtype=float)
    return matrix
```

[48]:
```
file_matrix = read_from_file('matrix.txt')
print(file_matrix)
```

```
[[ 1.    3.5   0.2 ]
 [ 3.   -1.2  -1.  ]
 [ 4.    0.32 -0.2 ]]
```

[54]:
```
def inv(matrix: np.ndarray):
    try:
        return np.linalg.inexcept np.linalg.LinAlgError as e:
        print(e)
```

[64]:
```
inverse = inv(file_matrix)
product = np.dot(file_matrix, inverse)
print(product)
```

```
print(np.allclose(product, np.eye(3)))
```

```
[[ 1.00000000e+00 -4.82142791e-17 -2.05567797e-17]
 [ 0.00000000e+00  1.00000000e+00  2.22044605e-16]
 [-2.04979339e-18 -7.29687218e-18  1.00000000e+00]]
True
```