MATRICES CON PYTHON

Ramon Ceballos

21/2/2021

1. Definir Matrices en Python

Utilizaremos la librería **numpy**.

```
import numpy as np
```

Para crear una matriz fila, se realiza lo siguiente:

```
row = [1,2,3]
row
```

```
## [1, 2, 3]
```

Para crear una matriz columna, se realiza lo siguiente:

```
col = [[1],[2],[3]]
col
```

```
## [[1], [2], [3]]
```

Para crear una matriz, se realiza lo siguiente:

```
M = [[1,2,3],[4,5,6],[7,8,9]]
M
```

```
## [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

1.1 Acceso a los elementos de una matriz

Para llamar a un elemento, utilizamos la sintaxis siguiente:

```
M[1][1]
```

```
## 5
```

La primera posición indica la fila y, la segunda, la columna.

Observación. En `Python`, al contrario que en `R`, las posiciones empiezan en 0.

```
M[0][0]
```

```
## 1
```

Para llamar a una fila, se realiza lo siguiente:

```
M[0]
```

```
## [1, 2, 3]
```

1.2 Empleo de NumPy para matrices

Cambiémosle la sintaxis a la matriz, para poder trabajar correctamente con ella:

```
import numpy as np
M = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(M)
```

```
## [[1 2 3]
##  [4 5 6]
##  [7 8 9]]
```

La función **`np.array()`** tiene un parámetro, **`dtype`**, en el cual podemos indicar el tipo de dato de la matriz: int, float, complex...

```
M = np.array([[1,2,3],[4,5,6],[7,8,9]], dtype = complex)
print(M)
```

```
## [[1.+0.j 2.+0.j 3.+0.j]
##  [4.+0.j 5.+0.j 6.+0.j]
##  [7.+0.j 8.+0.j 9.+0.j]]
```

```
M = np.array([[1,2,3],[4,5,6],[7,8,9]], dtype = float)
print(M)
```

```
## [[1. 2. 3.]
##  [4. 5. 6.]
##  [7. 8. 9.]]
```

Para llamar a un elemento, utilizamos la sintaxis mostrada anteriormente:

```
M = np.array([[1,2,3],[4,5,6],[7,8,9]])
M[0][2]
```

```
## 3
```

Para llamar a una fila, utilizamos la sintaxis mostrada anteriormente:

```
print(M[1])
```

```
## [4 5 6]
```

También nos sirve la sintaxis siguiente para llamar una fila:

```
M[1,:]
```

```
## array([4, 5, 6])
```

Para llamar a una columna, se realiza lo siguiente:

```
M[:,0]
```

```
## array([1, 4, 7])
```

1.3 Definir tipos de matrices con NumPy

Para crear una **matriz de ceros**, utilizamos la función `np.zeros((fil,col))`:

```
print(np.zeros((5,7)))
```

```
## [[0. 0. 0. 0. 0. 0. 0.]
##  [0. 0. 0. 0. 0. 0. 0.]
##  [0. 0. 0. 0. 0. 0. 0.]
##  [0. 0. 0. 0. 0. 0. 0.]
##  [0. 0. 0. 0. 0. 0. 0.]]
```

Para crear una **matriz de unos**, utilizamos la función `np.ones((fil,col))`:

```
print(np.ones((5,10)))
```

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

Para crear una **matriz diagonal** hacemos lo siguiente:

```
x = [1,2,3,4]
N = np.diag(x)
N
```

```
## array([[1, 0, 0, 0],
##        [0, 2, 0, 0],
##        [0, 0, 3, 0],
##        [0, 0, 0, 4]])
```

1.4 Obtener la diagonal y dimensión de una matriz en Python

Y para obtener **la diagonal principal** de una matriz, utilizamos de nuevo la función **`numpy.diag()`**.

```
np.diag(N)
```

```
## array([1, 2, 3, 4])
```

Para saber **la dimensión de una matriz**, utilizamos la función **`np.shape()`**:

```
np.shape(M)
```

```
## (3, 3)
```

2. Manipulación de matrices con Python

Si queremos la suma de todos los elementos de una matriz:

```
np.sum(M)
```

```
## 45
```

Si queremos la suma por columnas, haremos:

```
np.sum(M, axis = 0)
```

```
## array([12, 15, 18])
```

Si queremos la suma por filas, haremos:

```
np.sum(M, axis = 1)
```

```
## array([ 6, 15, 24])
```

Si queremos el producto de todos los elementos de una matriz:

```
np.prod(M)
```

```
## 362880
```

Si queremos la media aritmética de todos los elementos de una matriz:

```
np.mean(M)
```

```
## 5.0
```

Si queremos la media aritmética por filas o columnas:

```
np.mean(M, axis = 0) #Por columnas
```

```
## array([4., 5., 6.])
```

```
np.mean(M, axis = 1) #Por filas
```

```
## array([2., 5., 8.])
```

3. Operaciones con matrices en Python

3.1 Transpuesta de una matriz

Para calcular la **transpuesta de una matriz**, utilizamos la función **`.transpose()`**:

```
print(M)
```

```
## [[1 2 3]
##  [4 5 6]
##  [7 8 9]]
```

```
print(M.transpose())
```

```
## [[1 4 7]
##  [2 5 8]
##  [3 6 9]]
```

3.2 Traza de una matriz

Para calcular la **traza de una matriz** (suma de los elementos de la diagonal):

```
print(M.trace())
```

```
## 15
```

3.3 Suma de una matriz

Suma de matrices:

```
A = np.array([[1,2],[2,0]])
B = np.array([[3,0],[1,4]])
print(A+B)
```

```
## [[4 2]
##  [3 4]]
```

3.4 Producto de una matriz por una escalar

El producto de un escalar por una matriz:

```
print(5*A)
```

```
## [[ 5 10]
##  [10  0]]
```

3.5 Producto de matrices

Producto de matrices:

```
print(A.dot(B))
```

```
## [[5 8]
##  [6 0]]
```

Observad que si utilizáis la sintaxis `A*B`, se multiplican elemento a elemento, como ocurría en `R`.

```
print(A*B)
```

```
## [[3 0]
##  [2 0]]
```

3.6 Potencia de una matriz

Para **calcular la potencia** de una matriz se emplea **np.linalg.matrix_power()**.

```
print(np.linalg.matrix_power(A,5))
```

```
## [[65 58]
##  [58 36]]
```

4. Rango e inversa de una matriz

4.1 Rango de una matriz en `Python`

Para calcular el rango de una matriz, utilizamos **`np.linalg.matrix_rank()`**.

```
A
```

```
## array([[1, 2],
##        [2, 0]])
```

```
np.linalg.matrix_rank(A)
```

```
## 2
```

```
B
```

```
## array([[3, 0],
##        [1, 4]])
```

```
np.linalg.matrix_rank(B)
```

```
## 2
```

4.2 Inversa de una matriz en Python

Para calcular la inversa, se usa **np.linalg.inv()**.

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

```
## [[ 0.    0.5 ]
##  [ 0.5  -0.25]]
```

Comprobamos si da la matriz identidad.

```
print(np.linalg.inv(A).dot(A)) #Comprobamos
```

```
## [[1. 0.]
##  [0. 1.]]
```