Algebra Lineal

June 17, 2023

Vectores y operaciones

[5]:
```
# Vectores
# Creando un vector con una lista simple de python

v1py = [1, 2, 3]
v2py = [4, 5, 6]

v1py + v2py
```

[5]:
```
[1, 2, 3, 4, 5, 6]
```

[6]:
```
7 * v1py
```

[6]:
```
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

[7]:
```
# Importar numpy que es la librería matemática que me permite utilizar los␣
↪objetos de algebra
import numpy as np
```

[8]:
```
# Crear un array de numpy
v1np = np.array(v1py) # np.array([1,2,3])
v2np = np.array(v2py) # np.array([4,5,6])
```

[9]:
```
# Sumando vectores
v1np + v2np
```

[9]:
```
array([5, 7, 9])
```

[10]:
```
# Restando vectores
v1np - v2np
```

[10]:
```
array([-3, -3, -3])
```

[11]:
```
# Multiplicación por escalar
7 * v1np
```

[11]:
```
array([ 7, 14, 21])
```

[14]:
```
# Funcion para crear vectores o arrays con 1's
vunos = np.ones(10)
vunos
```

[14]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

[20]:
```
#Crear arreglos de numpy por rango
vrange = np.arange(1,4)
vrange
```

[20]:
```
array([1, 2, 3])
```

[21]:
```
#Producto punto en numpy
np.dot(v1np,v2np)
```

[21]:
```
32
```

[22]:
```
#Producto punto con sum (función nativa de python)
sum(v1np * v2np)
```

[22]:
```
32
```

[23]:
```
# Ejemplo de vectores ortogonales o perpendiculares

v1 = np.array([3,4])
v2 = np.array([4,-3])

np.dot(v1,v2)
```

[23]:
```
0
```

[24]:
```
# Ejemplo calculo norma del vector

v3 = [3,7,5,1]
np.linalg.norm(v3)
```

[24]:
```
9.16515138991168
```

Matrices - suma, resta y multiplicación por escalar

[25]:
```
A = np.array([[1, 3, 2],  #Para crear matrices van dentro de una lista
              [1, 0, 0],  # y son listas separadas por comas
              [1, 2, 2]])
```

[26]:
```
B = np.array([[1, 0, 5],
              [7, 5, 0],
              [2, 1, 1]])
```

[27]:
```
# Suma de matrices
A + B
```

[27]:
```
array([[2, 3, 7],
       [8, 5, 0],
       [3, 3, 3]])
```

[28]:
```
# Resta de matrices
A - B
```

[28]:
```
array([[ 0,  3, -3],
       [-6, -5,  0],
       [-1,  1,  1]])
```

[29]:
```
# Multiplicacion de matriz por escalar
A * 5
```

[29]:
```
array([[ 5, 15, 10],
       [ 5,  0,  0],
       [ 5, 10, 10]])
```

[30]:
```
# Para ver la dimension (renglones x columnas) de mi matriz
A.shape
```

[30]:
```
(3, 3)
```

[31]:
```
# Para ver el número de elementos
A.size
```

[31]:
```
9
```

Multiplicación de matrices

[47]:
```
A = np.arange(1, 13).reshape(3, 4) #Aqui le digo que cree una matriz en el␣
↪rango del 1 hasta < 13
A                                  # Y con reshape le digo que me la acomode␣
↪como matriz de 3*4
```

[47]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[41]:
```
B = np.arange(8).reshape(4,2)
B
```

[41]:
```
array([[0, 1],
       [2, 3],
```

```
       [4, 5],
       [6, 7]])
```

[42]:
```
A @ B #El arroba es el simbolo en python numpy para realizar la multiplicación
 ↪de matrices
```

[42]:
```
array([[ 40,  50],
       [ 88, 114],
       [136, 178]])
```

[43]:
```
#Matriz identidad
I = np.eye(2) #Funcion que me permite crear matrices identidad
I             # El parametro que recibe es el numero de r y c
```

[43]:
```
array([[1., 0.],
       [0., 1.]])
```

[44]:
```
C = np.array([[4,7],
              [2,6]])
C
```

[44]:
```
array([[4, 7],
       [2, 6]])
```

[45]:
```
I @ C
```

[45]:
```
array([[4., 7.],
       [2., 6.]])
```

[48]:
```
A = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]])
```

[49]:
```
# Calculo del determinante de una matriz (solo funciona en matrices cuadradas)
np.linalg.det(A)
```

[49]:
```
0.0
```

[50]:
```
# Matriz transpuesta
A = np.arange(6).reshape(3, 2)
A
```

[50]:
```
array([[0, 1],
       [2, 3],
       [4, 5]])
```

[51]:
```
np.transpose(A)
```

[51]:
```
array([[0, 2, 4],
       [1, 3, 5]])
```

[52]:
```
# Matriz inversa
A = np.array([[4, 7],
              [2, 6]])
```

[53]:
```
inv = np.linalg.inv(A)
inv
```

[53]:
```
array([[ 0.6, -0.7],
       [-0.2,  0.4]])
```

[ ]: