Algebra_Lineal_180622

June 18, 2022

[13]:
```
#Vectores
#Creando una variable con una lista simple de python
v1py = [2,4,6]
v1py
```

[13]:
```
[2, 4, 6]
```

[14]:
```
type(v1py)
```

[14]:
```
list
```

[4]:
```
#Importando numpy
import numpy as np
```

[8]:
```
#Crear array de numpy con unos
v1numpy = np.ones(3)
v1numpy
```

[8]:
```
array([1., 1., 1.])
```

[15]:
```
type(v1numpy)
```

[15]:
```
numpy.ndarray
```

[24]:
```
#Crear un array de numpy con valores pre-definidos
v2numpy = np.array([1,3,5])
v2numpy
```

[24]:
```
array([1, 3, 5])
```

[21]:
```
#Crear un array con un rango definido
v3numpy = np.arange(1,4)
v3numpy
```

[21]:
```
array([1, 2, 3])
```

[26]:
```
#Obtner un elemento por indice es igual que en las listas
v3numpy[1]
```

[26]: 2

[27]:
```
#Creando una nueva lista para el ejemplo
v2py = [3,6,9]
```

[28]:
```
#Comportamiento suma de listas
v1py + v2py
```

[28]: [2, 4, 6, 3, 6, 9]

[29]:
```
#Comportamiento suma de numpy arrays
v1numpy + v2numpy
```

[29]: array([2., 4., 6.])

[30]:
```
#Multiplicar lista por escalar
v2py * 5
```

[30]: [3, 6, 9, 3, 6, 9, 3, 6, 9, 3, 6, 9, 3, 6, 9]

[31]:
```
#Multiplicar array de numpy por escalar
v2numpy * 5
```

[31]: array([5, 15, 25])

[32]:
```
#Resta de arrays
v2numpy - v1numpy
```

[32]: array([0., 2., 4.])

[33]:
```
#Resta imposible de listas
v1py - v2py
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/vq/wc9_6gmx0h7gnvz5psj3xdnh0000gn/T/ipykernel_2150/2238391382.py i
 ↪<module>
----> 1 v1py - v2py

TypeError: unsupported operand type(s) for -: 'list' and 'list'
```

[34]:
```
x = np.arange(1,5)
y = np.array([2,4,6,8])
x, y
```

[34]: (array([1, 2, 3, 4]), array([2, 4, 6, 8]))

[35]:
```
#Producto punto, escalar o interior
#Python puro
sum(x*y)
```

[35]:
```
60
```

[36]:
```
#numpy
np.dot(x,y)
```

[36]:
```
60
```

[37]:
```
#calculo de la norma
np.linalg.norm(x)
```

[37]:
```
5.477225575051661
```

[41]:
```
#MATRICES
#Creando matrices en python con numpy
A = np.array([[1, 3, 2],
              [1, 0, 0],
              [1, 2, 2]])
A
```

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

[42]:
```
B = np.array([[1, 0, 5],
              [7, 5, 0],
              [2, 1, 1]])
B
```

[42]:
```
array([[1, 0, 5],
       [7, 5, 0],
       [2, 1, 1]])
```

[43]:
```
#Multiplicar matriz por escalares
A * 3
```

[43]:
```
array([[3, 9, 6],
       [3, 0, 0],
       [3, 6, 6]])
```

[44]:
```
#Suma de matrices (requerido que sean del mismo tamaño)
A + B
```

[44]:
```
array([[2, 3, 7],
       [8, 5, 0],
       [3, 3, 3]])
```

[45]:
```
#Resta de matrices (requerido que sean del mismo tamaño)
A -B
```

[45]:
```
array([[ 0,  3, -3],
       [-6, -5,  0],
       [-1,  1,  1]])
```

[46]:
```
#Para ver la dimensión o tamaño de la matriz
A.shape
```

[46]:
```
(3, 3)
```

[47]:
```
#Ver la cantidad de elemntos que tiene mi matriz
A.size
```

[47]:
```
9
```

[53]:
```
#Ejemplo de multiplicación de matrices - Ejemplo de cración de matrices con␣
↪arange y reshape
A = np.arange(1, 13).reshape(3, 4)
A
```

[53]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[54]:
```
B = np.arange(8).reshape(4,2)
B
```

[54]:
```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7]])
```

[55]:
```
#Multiplicacion de matrices
A @ B
```

[55]:
```
array([[ 40,  50],
       [ 88, 114],
       [136, 178]])
```

[57]:
```
#No se cumple la propiedad comutativa
B @ A
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/var/folders/vq/wc9_6gmx0h7gnvz5psj3xdnh0000gn/T/ipykernel_2150/1888990598.py i
 ↪<module>
      1 #No se cumple la propiedad comutativa
----> 2 B @ A

ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, wit
 ↪gufunc signature (n?,k),(k,m?)->(n?,m?) (size 3 is different from 2)
```

[62]:
```
#Creando una matriz identidad
I = np.eye(2)
I
```

[62]:
```
array([[1., 0.],
       [0., 1.]])
```

[63]:
```
A = np.array([[4,7],
              [2,6]])
A
```

[63]:
```
array([[4, 7],
       [2, 6]])
```

[64]:
```
A @ I
```

[64]:
```
array([[4., 7.],
       [2., 6.]])
```

[65]:
```
I @ A
```

[65]:
```
array([[4., 7.],
       [2., 6.]])
```

[66]:
```
#Calculo del determinante
np.linalg.det(A)
```

[66]:
```
10.000000000000002
```

[67]:
```
C = np.arange(1, 10).reshape(3, 3)
C
```

[67]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

[68]:
```
np.linalg.det(C)
```

[68]:
```
0.0
```

[69]:
```
#Transponiendo una matriz
A = np.arange(6).reshape(3, 2)
A
```

[69]:
```
array([[0, 1],
       [2, 3],
       [4, 5]])
```

[70]:
```
np.transpose(A)
```

[70]:
```
array([[0, 2, 4],
       [1, 3, 5]])
```

[71]:
```
#Calculo de la matriz inversa
A = np.array([[4,7],
              [2,6]])
A
```

[71]:
```
array([[4, 7],
       [2, 6]])
```

[72]:
```
A_inv = np.linalg.inv(A)
A_inv
```

[72]:
```
array([[ 0.6, -0.7],
       [-0.2,  0.4]])
```

[73]:
```
B = np.array([[1, 1, 0],
              [2, -1, 1],
              [0, 3, 0]])
```

[74]:
```
B_inv = np.linalg.inv(B)
B_inv
```

[74]:
```
array([[ 1.        ,  0.        , -0.33333333],
       [ 0.        ,  0.        ,  0.33333333],
       [-2.        ,  1.        ,  1.        ]])
```

[ ]:
```

```