Álgebra Linear

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
v1 = [2,5]
v2 = [1,6,8]
v1 ,v2
```

```
([2, 5], [1, 6, 8])
```

```
type(v1)
```

```
list
```

```
v3 = np.array([8,3,9])
type(v3)
```

```
numpy.ndarray
```

```
v3.shape[0]
```

```
3
```

```
v3.shape
```

```
(3,)
```

```
v4=np.array([1.+2.j, 3.+4.j, 5, 6.j], dtype=complex)
v4
```

```
array([1.+2.j, 3.+4.j, 5.+0.j, 0.+6.j])
```

```
type(v4)
```

```
numpy.ndarray
```

Lendo elementos de um array

```
a = np.array([7,5,3,9,0,2])
a
```

```
array([7, 5, 3, 9, 0, 2])
```

```
a[0]
```

```
7
```

```
a[1:]
```

```
array([5, 3, 9, 0, 2])
```

```
a[1:4]
```

```
array([5, 3, 9])
```

```
a[-1]
```

```
2
```

```
a[-3]
```

```
9
```

```
a[-6]
```

```
7
```

```
a[-3:-1]
```

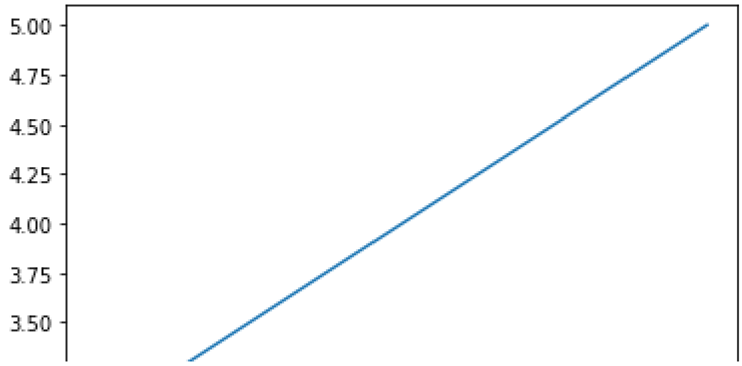
```
array([9, 0])
```

Plotando um vetor

```
v = [3,5]
u = [1,2,3]
```

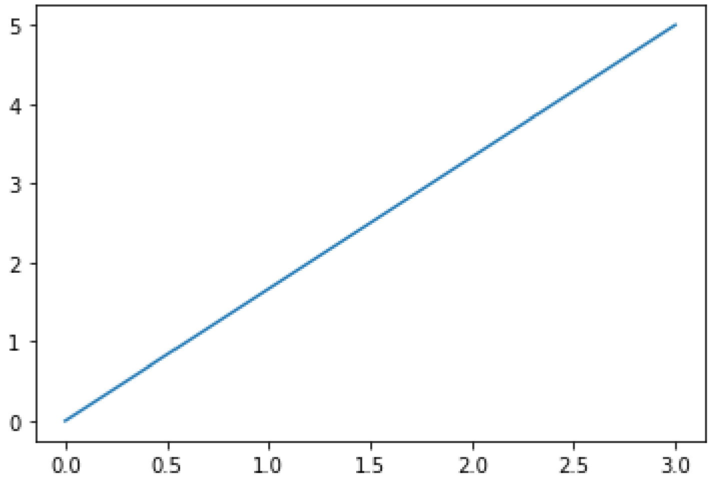
```
plt.plot(v)
```

```
[<matplotlib.lines.Line2D at 0x7f96fa0f12d0>]
```



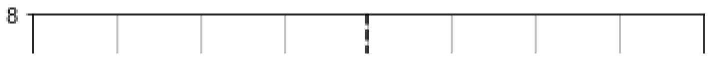

```
plt.plot([0,v[0]] , [0,v[1]])
```

```
[<matplotlib.lines.Line2D at 0x7f96fa0c5610>]
```

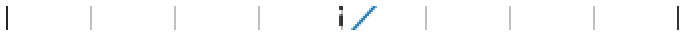


Plota um vetor 2D

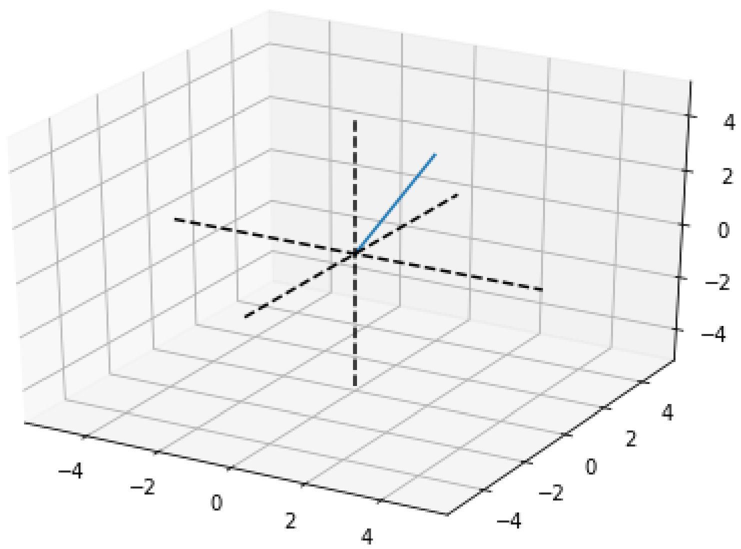
```
plt.plot([0,v[0]] , [0,v[1]])
plt.plot([8,-8] , [0,0] , 'k--')
plt.plot([0,0] , [8,-8] , 'k--')
plt.grid()
plt.axis((-8, 8, -8, 8))
plt.show()
```



Plota um vetor 3D



```
fig = plt.figure()
ax = Axes3D(fig)
ax.plot([0,u[0]],[0,u[1]],[0,u[2]])
#plt.axis('equal')
ax.plot([0, 0],[0, 0],[-5, 5],'k--')
ax.plot([0, 0],[-5, 5],[0, 0],'k--')
ax.plot([-5, 5],[0, 0],[0, 0],'k--')
plt.show()
```

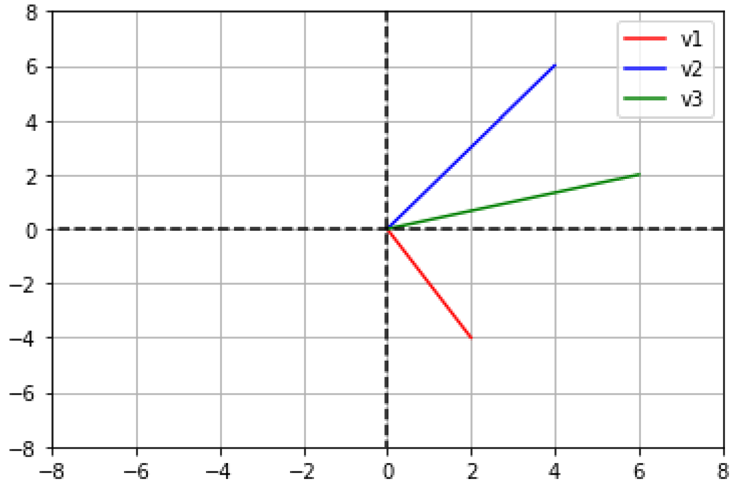


Soma de vetores

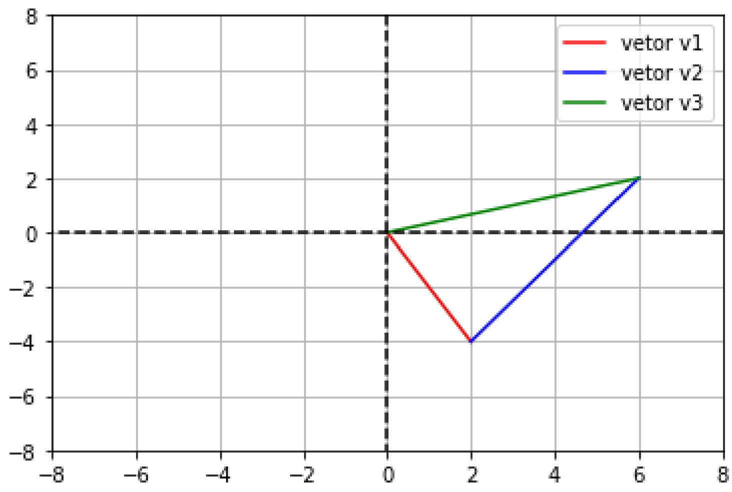
```
v1 = np.array([2,-4])
v2 = np.array([4,6])
v3 = v1+v2
v3 = np.add(v1,v2)
print('V3 =' ,v3)
plt.plot([0,v1[0]] , [0,v1[1]] , 'r' , label = 'v1')
plt.plot([0,v2[0]] , [0,v2[1]], 'b' , label = 'v2')
plt.plot([0,v3[0]] , [0,v3[1]] , 'g' , label = 'v3')
plt.plot([8,-8] , [0,0] , 'k--')
plt.plot([0,0] , [8,-8] , 'k--')
plt.grid()
```

```
plt.axis((-8, 8, -8, 8))
plt.legend()
plt.show()
```

```
V3 = [6 2]
```



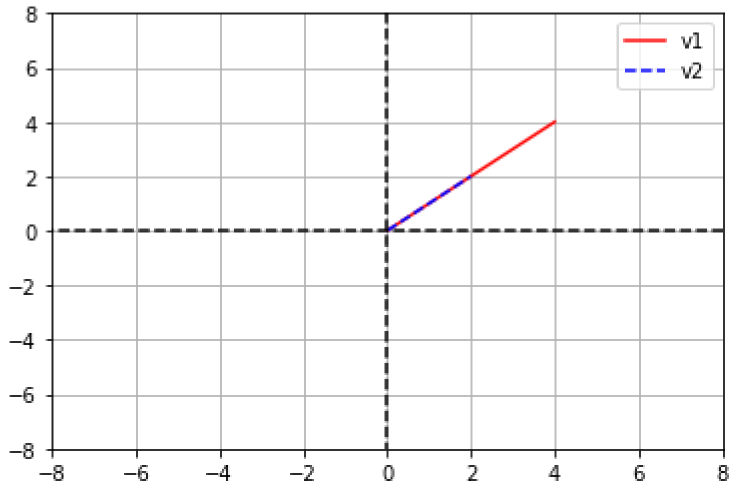

```
plt.plot([0,v1[0]] , [0,v1[1]] , 'r' , label = 'vetor v1')
plt.plot([0,v2[0]]+v1[0] , [0,v2[1]]+v1[1], 'b' , label = 'veto
plt.plot([0,v3[0]] , [0,v3[1]] , 'g' , label = 'vetor v3')
plt.plot([8,-8] , [0,0] , 'k--')
plt.plot([0,0] , [8,-8] , 'k--')
plt.grid()
plt.axis((-8, 8, -8, 8))
plt.legend()
plt.show()
```



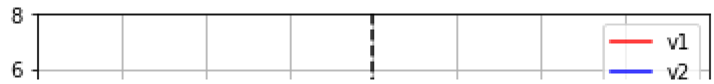
Multiplicação de vetor por um escalar

```
u1 = np.array([4,4])
a = .5
```

```
u2 = u1*a
plt.plot([0,u1[0]] , [0,u1[1]] , 'r' , label = 'v1')
plt.plot([0,u2[0]] , [0,u2[1]], 'b--' , label = 'v2')
plt.plot([8,-8] , [0,0] , 'k--')
plt.plot([0,0] , [8,-8] , 'k--')
plt.grid()
plt.axis((-8, 8, -8, 8))
plt.legend()
plt.show()
```




```
u1 = np.array([4,4])
a = -.3
u2 = u1*a
plt.plot([0,u1[0]] , [0,u1[1]] , 'r' , label = 'v1')
plt.plot([0,u2[0]] , [0,u2[1]], 'b' , label = 'v2')
plt.plot([8,-8] , [0,0] , 'k--')
plt.plot([0,0] , [8,-8] , 'k--')
plt.grid()
plt.axis((-8, 8, -8, 8))
plt.legend()
plt.show()
```



Multiplicação de vetores


```
a1 = [2,4,6]
a2 = [3,5,1]
print(np.multiply(a1,a2))
```

```
[ 6 20  6]
```

Produto interno

```
a1 = np.array([2,4,6])
a2 = np.array([3,5,1])

dotp = a1@a2
print(" Dot product - ",dotp)

dotp = np.dot(a1,a2)
print(" Dot product usign np.dot",dotp)

dotp = np.inner(a1,a2)
print(" Dot product usign np.inner", dotp)

dotp = sum(np.multiply(a1,a2))
print(" Dot product usign np.multiply & sum",dotp)

dotp = np.matmul(a1,a2)
print(" Dot product usign np.matmul",dotp)

dotp = 0
for i in range(len(a1)):
    dotp = dotp + a1[i]*a2[i]
print(" Dot product usign for loop" , dotp)
```

```
Dot product -  32
Dot product usign np.dot 32
Dot product usign np.inner 32
Dot product usign np.multiply & sum 32
Dot product usign np.matmul 32
Dot product usign for loop 32
```

Tamanho de um vetor

```
v3 = np.array([1,2,3,4,5,6,7])
length = np.sqrt(np.dot(v3,v3))
length
```

```
11.832159566199232
```

```
v3 = np.array([1,2,3,4,5,6,7])
length = np.sqrt(sum(np.multiply(v3,v3)))
length
```

```
11.832159566199232
```

```
v3 = np.array([1,2,3,4,5,6,7])
length = np.sqrt(np.matmul(v3,v3))
length
```

```
11.832159566199232
```

Vetor normalizado

```
v1 = [1,1]
length_v1 = np.sqrt(np.dot(v1,v1))
norm_v1 = v1/length_v1
length_v1 , norm_v1
```

```
(1.4142135623730951, array([0.70710678, 0.70710678]))
```

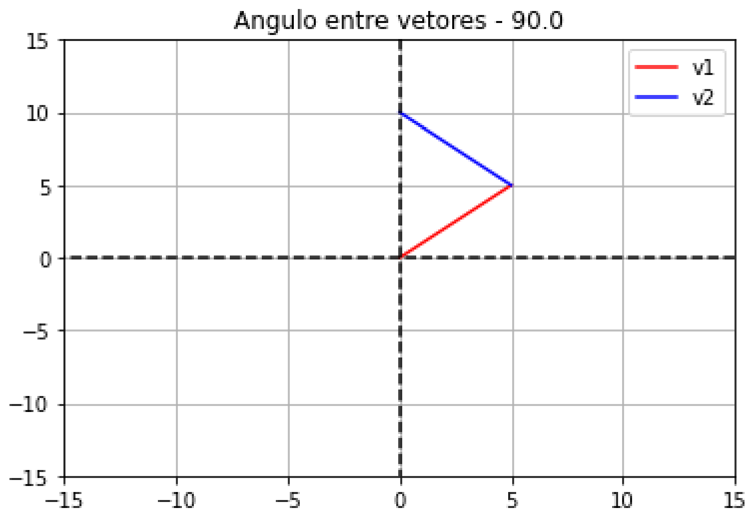
```
v1 = [1,1]
norm_v1 = v1/np.linalg.norm(v1)
norm_v1
```

```
array([0.70710678, 0.70710678])
```

Ângulo entre vetores

```
#First Method
v1 = np.array([5,5])
```

```
v2 = np.array([-5,5])
ang = np.rad2deg(np.arccos( np.dot(v1,v2) / (np.linalg.norm(v1)
plt.plot([0,v1[0]] , [0,v1[1]] , 'r' , label = 'v1')
plt.plot([0,v2[0]]+v1[0] , [0,v2[1]]+v1[1], 'b' , label = 'v2')
plt.plot([15,-15] , [0,0] , 'k--')
plt.plot([0,0] , [15,-15] , 'k--')
plt.grid()
plt.axis((-15, 15, -15, 15))
plt.legend()
plt.title('Angulo entre vetores - %s'  %ang)
plt.show()
```

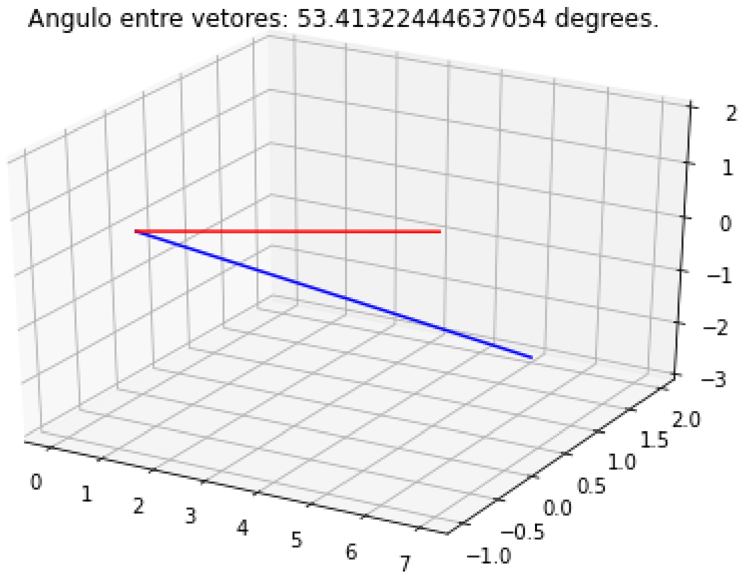



```
#Second Method
v1 = np.array([5,5])
v2 = np.array([-5,5])
lengthV1 = np.sqrt(np.dot(v1,v1))
lengthV2  = np.sqrt(np.dot(v2,v2))
ang = np.rad2deg(np.arccos( np.dot(v1,v2) / (lengthV1 * lengthV
print('Angulo entre vetores - %s' %ang)
```

```
Angulo entre vetores - 90.0
```

```
v1 = np.array([5,2,-3])
v2 = np.array([7,-1,2])
fig = plt.figure()
ax = Axes3D(fig)
ax.plot([0, v1[0]],[0, v1[1]],[0, v1[2]],'b')
ax.plot([0, v2[0]],[0, v2[1]],[0, v2[2]],'r')
ang = np.rad2deg(np.arccos( np.dot(v1,v2) / (np.linalg.norm(v1)
plt.title('Angulo entre vetores: %s degrees.' %ang)
```

```
Text(0.5, 0.92, 'Angulo entre vetores: 53.41322444637054 degrees.')
```



Produtos interno e externo

```
v1 = np.array([1,2,1])
v2 = np.array([2,1,2])
np.inner(v1,v2)

print("\n Inner Product ==>  \n", np.inner(v1,v2))
print("\n Outer Product ==>  \n", np.outer(v1,v2))
```

```
Inner Product ==>
6

Outer Product ==>
[[2 1 2]
[4 2 4]
[2 1 2]]
```

Produto vetorial

```
v1 = np.array([7,0,0])
v2 = np.array([0,7,0])
print("\nVector Cross Product ==>  \n", np.cross(v1,v2))
```

```
Vector Cross Product ==>
 [ 0  0 49]
```

Operações com matrizes

Criação de matriz

```
A = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]])
A
```

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16]])
```

```
type(A)
```

```
numpy.ndarray
```

```
A.dtype
```

```
dtype('int64')
```

```
B = np.array([[2.7,6.03,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,1
B
```

```
array([[ 2.7 ,  6.03,  3.  ,  4.  ],
       [ 5.  ,  6.  ,  7.  ,  8.  ],
       [ 9.  , 10.  , 11.  , 12.  ],
       [13.  , 14.  , 15.  , 16.  ]])
```

```
type(B)
```

```
numpy.ndarray
```

```
B.dtype
```

```
dtype('float64')
```

```
A.shape
```

```
(4, 4)
```

```
A[0,]
```

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

```
A[:,0]
```

```
array([ 1,  5,  9, 13])
```

```
A[0,0]
```

```
1
```

```
A[0][0]
```

```
1
```

```
A[1:3 , 1:3]
```

```
array([[ 6,  7],
       [10, 11]])
```

Matriz de zeros

```
np.zeros(36).reshape(6,6)
```

```
array([[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., 0.]])
```

```
np.zeros((6,6))
```

```
array([[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., 0.]])
```

Matriz de 1's

```
np.ones(25).reshape(5,5)
```

```
array([[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.]])
```

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

```
array([[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.]])
```

Matriz de números aleatórios

```
X = np.random.random((3,4))
X
```

```
array([[0.53397137, 0.58065504, 0.32512569, 0.90971213],
       [0.45705303, 0.4927677 , 0.61340607, 0.30473222],
       [0.34779196, 0.6856701 , 0.70559851, 0.28757412]])
```

Matriz identidade

```
I = np.eye(7)
I
```

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

Matriz diagonal

```
D = np.diag([1,2,3,4,5,6,7])
D
```

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

Matrizes triangulares (superior e inferior)

```
M = np.random.randn(5,5)
U = np.triu(M)
L = np.tril(M)
print("matriz aleatória \n" , M)
print("\n")


print("matriz triangular inferior \n" , L)
print("\n")

print("matriz triangular superior \n" , U)
```

```
matriz aleatória
 [[-2.17576846  0.33262008  0.4093531  -1.10185554 -0.94736601]
 [ 0.25209     0.3011718  -0.72327479 -1.54251485 -1.19783959]
 [ 0.2553803   0.48403065 -0.88881891  0.52947003 -0.24585781]
 [-0.55550854  0.39665683 -0.05425757  0.67720513 -0.55145749]
 [-1.06509044  1.124052   -1.99757239  2.10018809  0.4990478 ]]


matriz triangular inferior
 [[-2.17576846  0.          0.          0.          0.        ]
 [ 0.25209     0.3011718   0.          0.          0.        ]
 [ 0.2553803   0.48403065 -0.88881891  0.          0.        ]
 [-0.55550854  0.39665683 -0.05425757  0.67720513  0.        ]
 [-1.06509044  1.124052   -1.99757239  2.10018809  0.4990478 ]]


matriz triangular superior
 [[-2.17576846  0.33262008  0.4093531  -1.10185554 -0.94736601]
 [ 0.          0.3011718  -0.72327479 -1.54251485 -1.19783959]
 [ 0.          0.         -0.88881891  0.52947003 -0.24585781]
 [ 0.          0.          0.          0.67720513 -0.55145749]
 [ 0.          0.          0.          0.          0.4990478 ]]
```

Concatenação de matrizes

```
A = np.array([[1,2] , [3,4] ,[5,6]])
B = np.array([[9,2] , [3,-3]])
C = np.concatenate((A,B))
C , C.shape , type(C) , C.dtype
```

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

```
        [ 5,  6],
        [ 9,  2],
        [ 3, -3]]), (5, 2), numpy.ndarray, dtype('int64'))
```

```
np.full((7,7) , 4)
```

```
array([[4, 4, 4, 4, 4, 4, 4],
       [4, 4, 4, 4, 4, 4, 4],
       [4, 4, 4, 4, 4, 4, 4],
       [4, 4, 4, 4, 4, 4, 4],
       [4, 4, 4, 4, 4, 4, 4],
       [4, 4, 4, 4, 4, 4, 4],
       [4, 4, 4, 4, 4, 4, 4]])
```

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

```
array([[ 1,  2,  3],
       [ 4, -3,  6],
       [ 8,  7,  0]])
```

```
M.flatten()
```

```
array([ 1,  2,  3,  4, -3,  6,  8,  7,  0])
```

Soma de matrizes

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

print("\n Primeira matriz (M)  ==>  \n", M)
print("\n Segunda matriz (N)  ==>  \n", N)

C = M+N
print("\n soma (M+N)  ==>  \n", C)

# ou

C = np.add(M,N,dtype = np.float64)
print("\n soma usando np.add  ==>  \n", C)
```

```
 Primeira matriz (M)  ==>
 [[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]
```

```
Segunda matriz (N)  ==>
[[5 5 5]
[6 6 6]
[7 7 7]]

soma (M+N)  ==>
[[ 6  7  8]
[10  3 12]
[14 15  7]]

soma usando np.add  ==>
[[ 6.  7.  8.]
[10.  3. 12.]
[14. 15.  7.]]
```

Subtração de matrizes

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

print("\n Primeira matriz (M)  ==>  \n", M)
print("\n Segunda matriz (N)  ==>  \n", N)

C = M-N
print("\n Subtração (M-N)  ==>  \n", C)

# ou

C = np.subtract(M,N,dtype = np.float64)
print("\n Subtração usando np.subtract  ==>  \n", C)
```

```
Primeira matriz (M)  ==>
[[ 1  2  3]
[ 4 -3  6]
[ 7  8  0]]

Segunda matriz (N)  ==>
[[5 5 5]
[6 6 6]
[7 7 7]]

Subtração (M-N)  ==>
[[-4 -3 -2]
[-2 -9  0]
[ 0  1 -7]]

Subtração usando np.subtract  ==>
[[-4. -3. -2.]
```

```
 [-2. -9.  0.]
 [ 0.  1. -7.]]
```

Multiplicação de matriz por escalar

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

C = 20

print("\n Matriz (M)  ==>  \n", M)

print("\n Multiplicação por escalar ==>  \n", C*M)

# ou

print("\n Multiplicação por escalar usando np.multiply ==>  \n"
```

```
 Matriz (M)  ==> 
 [[ 1  9  3]
 [ 2 -7  6]
 [ 8  7  0]]

 Multiplicação por escalar ==> 
 [[  20  180   60]
 [  40 -140  120]
 [ 160  140    0]]

 Multiplicação por escalar usando np.multiply ==> 
 [[  20  180   60]
 [  40 -140  120]
 [ 160  140    0]]
```

Transposta de uma matriz

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

print("\n Matriz (M)  ==>  \n", M)

print("\n Transposta de M ==>  \n", np.transpose(M))

# ou

print("\nTransposta de M ==>  \n", M.T)
```

```
Matriz (M)  ==>
[[ 1  9  3]
[ 2 -7  6]
[ 8  7  0]]

Transposta de M ==>
[[ 1  2  8]
[ 9 -7  7]
[ 3  6  0]]

Transposta de M ==>
[[ 1  2  8]
[ 9 -7  7]
[ 3  6  0]]
```

Determinante de uma matriz

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

print("\n Matriz (M)  ==>  \n", M)

print("\n Determinante de M ==>  ", np.linalg.det(M))
```

```
Matriz (M)  ==>
[[ 1  9  3]
[ 2 -7  6]
[ 8  7  0]]

Determinante de M ==>   600.0
```

Posto de uma matriz

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

print("\n Matriz (M)  ==>  \n", M)

print("\n Posto de M ==> ", np.linalg.matrix_rank(M))
```

```
Matriz (M)  ==>
[[ 1  9  3]
[ 2 -7  6]
[ 8  7  0]]

Posto de M ==>  3
```

Traço de uma matriz

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

print("\n Matriz (M)  ==>  \n", M)

print("\n Traço de M ==> ", np.trace(M))
```

```
Matriz (M)  ==>
[[ 1  9  3]
[ 2 -7  6]
[ 8  7  0]]

Traço de M ==>  -6
```

Inversa de uma matriz

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

print("\n Matriz (M)  ==>  \n", M)

print("\n Inversa de M ==> \n", np.linalg.inv(M))
```

```
Matriz (M)  ==>
[[ 1  9  3]
[ 2 -7  6]
[ 8  7  0]]

Inversa de M ==>
[[-0.07        0.035       0.125     ]
[ 0.08       -0.04       -0.        ]
[ 0.11666667  0.10833333 -0.04166667]]
```

Multiplicação de matrizes (pontual)

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

print("\n Primeira matriz (M)  ==>  \n", M)
print("\n Segunda matriz (N)  ==>  \n", N)
```

```
print("\n Multiplication pontual de M e N  ==> \n", M*N)

# ou

print("\n Multiplication pontual de M e N  ==> \n", np.multiply
```

```
Primeira matriz (M)  ==>
[[ 1  9  3]
[ 2 -7  6]
[ 8  7  0]]

Segunda matriz (N)  ==>
[[5 5 5]
[6 6 6]
[7 7 7]]

Multiplication pontual de M e N  ==>
[[  5  45  15]
[ 12 -42  36]
[ 56  49   0]]

Multiplication pontual de M e N  ==>
[[  5  45  15]
[ 12 -42  36]
[ 56  49   0]]
```

Produto escalar matricial

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

print("\n Primeira matriz (M) \n", M)
print("\n Segunda matriz (N) \n", N)

print("\n Produto escalar \n", M@N)

# ou

print("\n Produto escala usando np.matmul \n", np.matmul(M,N))

# ou

print("\n Produto escala usando np.dot \n", np.dot(M,N))
```

```
Primeira matriz (M)
[[ 1  9  3]
[ 2 -7  6]
[ 8  7  0]]

Segunda matriz (N)
[[5 5 5]
[6 6 6]
[7 7 7]]

Produto escalar
[[80 80 80]
[10 10 10]
[82 82 82]]

Produto escala usando np.matmul
[[80 80 80]
[10 10 10]
[82 82 82]]

Produto escala usando np.dot
[[80 80 80]
[10 10 10]
[82 82 82]]
```

"Divisão" de Matrizes

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

print("\n Primeira matriz (M) \n", M)
print("\n Segunda matriz (N) \n", N)



print("\n Divisão (M/N) \n", M/N)

# ou

print("\n Divisão (M/N) \n", np.divide(M,N))
```

```
Primeira matriz (M)
[[ 1  9  3]
[ 2 -7  6]
[ 8  7  0]]

Segunda matriz (N)
[[5 5 5]
[6 6 6]
[7 7 7]]

Divisão (M/N)
```

```
[[ 0.2         1.8         0.6       ]
 [ 0.33333333 -1.16666667  1.        ]
 [ 1.14285714  1.          0.        ]]

Divisão (M/N)
[[ 0.2         1.8         0.6       ]
 [ 0.33333333 -1.16666667  1.        ]
 [ 1.14285714  1.          0.        ]]
```

Soma de todos elementos da matriz

```
N = np.array([[5,5,5],[6,6,6],[7,7,7]])

print("Matriz (N) \n", N)

print ("Soma de todos elementos da matriz")
print (np.sum(N))
```

```
Matriz (N)
 [[5 5 5]
 [6 6 6]
 [7 7 7]]
Soma de todos elementos da matriz
54
```

Adição com base na coluna

```
N = np.array([[5,5,5],[6,6,6],[7,7,7]])

print("Matriz (N)  ==>  \n", N)

print ("Adição com base na coluna")
print (np.sum(N,axis=0))
```

```
Matriz (N)  ==>
 [[5 5 5]
 [6 6 6]
 [7 7 7]]
Adição com base na coluna
[18 18 18]
```

Adição com base na linha

```
N = np.array([[5,5,5],[6,6,6],[7,7,7]])
```

```
print("Matriz (N)  ==>  \n", N)

print ("Adição com base na linha")
print (np.sum(N,axis=1))
```

```
Matriz (N)  ==>
 [[5 5 5]
 [6 6 6]
 [7 7 7]]
Adição com base na linha
[15 18 21]
```

Produto de Kronecker de matrizes

```
M1 = np.array([[1,2,3] , [4,5,6]])
M1
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
M2 = np.array([[10,10,10],[10,10,10]])
M2
```

```
array([[10, 10, 10],
       [10, 10, 10]])
```

```
np.kron(M1,M2)
```

```
array([[10, 10, 10, 20, 20, 20, 30, 30, 30],
       [10, 10, 10, 20, 20, 20, 30, 30, 30],
       [40, 40, 40, 50, 50, 50, 60, 60, 60],
       [40, 40, 40, 50, 50, 50, 60, 60, 60]])
```

Multiplicação matriz-vetor

```
A = np.array([[1,2,3] ,[4,5,6]])
v = np.array([10,20,30])
print ("Multiplicação matriz-vetor \n", A*v)
```

```
Multiplicação matriz-vetor
 [[ 10  40  90]
 [ 40 100 180]]
```

Produto escalar matriz-vetor

```
A = np.array([[1,2,3] ,[4,5,6]])
v = np.array([10,20,30])

print ("Produto escalar matriz-vetor \n" , A@v)
```

```
Produto escalar matriz-vetor
 [140 320]
```

Potências de matriz

```
M1 = np.array([[1,2],[5,6]])
M1
```

```
array([[1, 2],
       [5, 6]])
```

```
# Matriz na potencia 3

M1@M1@M1
```

```
array([[ 81, 106],
       [265, 346]])
```

```
# Matriz na potencia 3 usando np.linalg.matrix_power

np.linalg.matrix_power(M1,3)
```

```
array([[ 81, 106],
       [265, 346]])
```

Tensores

```
# Criando um Tensor

T1 = np.array([
  [[1,2,3],    [4,5,6],    [7,8,9]],
  [[10,11,12], [13,14,15], [16,17,18]],
  [[19,20,21], [22,23,24], [25,26,27]],
```

```
])

T1
```

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

       [[10, 11, 12],
        [13, 14, 15],
        [16, 17, 18]],

       [[19, 20, 21],
        [22, 23, 24],
        [25, 26, 27]]])
```

```
T2 = np.array([
  [[0,0,0] , [0,0,0] , [0,0,0]],
  [[2,2,2] , [2,2,2] , [2,2,2]],
  [[4,4,4] , [4,4,4] , [4,4,4]]

])

T2
```

```
array([[[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]],

       [[2, 2, 2],
        [2, 2, 2],
        [2, 2, 2]],

       [[4, 4, 4],
        [4, 4, 4],
        [4, 4, 4]]])
```

Soma de tensores

```
A = T1+T2
A
```

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

       [[12, 13, 14],
        [15, 16, 17],
        [18, 19, 20]],
```

```
        [[23, 24, 25],
         [26, 27, 28],
         [29, 30, 31]]])
```

Subtração de tensores

```
S = T1-T2
S
```

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

       [[ 8,  9, 10],
        [11, 12, 13],
        [14, 15, 16]],

       [[15, 16, 17],
        [18, 19, 20],
        [21, 22, 23]]])
```

```
np.subtract(T1,T2)
```

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

       [[ 8,  9, 10],
        [11, 12, 13],
        [14, 15, 16]],

       [[15, 16, 17],
        [18, 19, 20],
        [21, 22, 23]]])
```

Produto de tensores (baseado em elementos)

```
P = T1*T2
P
```

```
array([[[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0]],

       [[ 20,  22,  24],
        [ 26,  28,  30],
        [ 32,  34,  36]],

       [[ 76,  80,  84],
```

```
       [ 88,  92,  96],
       [100, 104, 108]]])
```

```
np.multiply(T1,T2)
```

```
array([[[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0]],

       [[ 20,  22,  24],
        [ 26,  28,  30],
        [ 32,  34,  36]],

       [[ 76,  80,  84],
        [ 88,  92,  96],
        [100, 104, 108]]])
```

"Divisão" de tensores (baseado em elementos)

```
D = T1/T2
D
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarning: divic
  """Entry point for launching an IPython kernel.
array([[[ inf,  inf,  inf],
        [ inf,  inf,  inf],
        [ inf,  inf,  inf]],

       [[5.  , 5.5 , 6.  ],
        [6.5 , 7.  , 7.5 ],
        [8.  , 8.5 , 9.  ]],

       [[4.75, 5.  , 5.25],
        [5.5 , 5.75, 6.  ],
        [6.25, 6.5 , 6.75]]])
```

```
np.divide(T1,T2)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarning: divic
  """Entry point for launching an IPython kernel.
array([[[ inf,  inf,  inf],
        [ inf,  inf,  inf],
        [ inf,  inf,  inf]],

       [[5.  , 5.5 , 6.  ],
        [6.5 , 7.  , 7.5 ],
        [8.  , 8.5 , 9.  ]],

       [[4.75, 5.  , 5.25],
        [5.5 , 5.75, 6.  ],
        [6.25, 6.5 , 6.75]]])
```

Produto escalar de tensores

```
T1
```

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

       [[10, 11, 12],
        [13, 14, 15],
        [16, 17, 18]],

       [[19, 20, 21],
        [22, 23, 24],
        [25, 26, 27]]])
```

```
T2
```

```
array([[[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]],

       [[2, 2, 2],
        [2, 2, 2],
        [2, 2, 2]],

       [[4, 4, 4],
        [4, 4, 4],
        [4, 4, 4]]])
```

```
np.tensordot(T1,T2)
```

```
array([[126, 126, 126],
       [288, 288, 288],
       [450, 450, 450]])
```

Solução de sistemas lineares (AX=B)

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

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

```
B = np.random.random((3,1))
B
```

```
array([[0.10491694],
       [0.21342286],
       [0.1630831 ]])
```

```python
# Primeiro metodo
X = np.dot(np.linalg.inv(A) , B)
X
```

```
array([[ 0.00086686],
       [-0.00965696],
       [ 0.02737853]])
```

```python
# Segundo Metodo
X = np.matmul(np.linalg.inv(A) , B)
X
```

```
array([[ 0.00086686],
       [-0.00965696],
       [ 0.02737853]])
```

```python
# Terceiro metodo
X = np.linalg.inv(A)@B
X
```

```
array([[ 0.00086686],
       [-0.00965696],
       [ 0.02737853]])
```

```python
# Quarto metodo
X = np.linalg.solve(A,B)
X
```

```
array([[ 0.00086686],
       [-0.00965696],
       [ 0.02737853]])
```

0s conclusão: 06:01