# Numpy e Álgebra linear

May 14, 2020

### 0.0.1 Numpy

### 0.0.2 A partir de uma lista em Python

Podemos criar um array convertendo diretamente uma lista ou lista de listas:

[2]:
```
a = [1,2,3]
b = [1,2,3]
c = a + b
c
```

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

[3]:
```
import numpy as np
```

[5]:
```
aa = np.array(a)
bb = np.array(b)
```

[6]:
```
aa + bb
```

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

## 0.1 Métodos incorporados

Existem várias maneiras dentro de Numpy para gerar arrays.

- `zeros((dim1,dim2),type)`
- `ones((dim1,dim2),type)`
- `arange(começo,final,incremento)`

### 0.1.1 arange

Cria valores uniformemente espaçados dentro de um determinado intervalo.

[10]:
```
v = np.arange(0,10,0.1)
v
```

[10]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
       1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5,
       2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8,
```

```
3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5. , 5.1,
5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6. , 6.1, 6.2, 6.3, 6.4,
6.5, 6.6, 6.7, 6.8, 6.9, 7. , 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7,
7.8, 7.9, 8. , 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.9, 9. ,
9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, 9.8, 9.9])
```

### 0.1.2 zeros and ones

Gera arrays de zeros ou uns.

[12]:
```
a = np.zeros(5)
a
```

[12]:
```
array([0., 0., 0., 0., 0.])
```

[18]:
```
a = np.ones((3,3))
a
```

[18]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

### 0.1.3 linspace

Retorna números espaçados uniformemente em um intervalo especificado.

[24]:
```
a = np.linspace(0,20,5000)
print(a)
```

```
[0.00000000e+00 4.00080016e-03 8.00160032e-03 … 1.99919984e+01
 1.99959992e+01 2.00000000e+01]
```

[26]:
```
np.arange(0,20.01,0.01)
```

[26]:
```
array([0.000e+00, 1.000e-02, 2.000e-02, …, 1.999e+01, 2.000e+01,
       2.001e+01])
```

## 0.2 eye

Cria um array (matriz) identidade

[30]:
```
I = np.eye(3,3)
I
```

[30]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

## 0.3 Random

O Numpy também tem várias maneiras de criar arrays de números aleatórios:

### 0.3.1 rand

Cria uma array e o preenche com números aleatórios de uma distribuição uniforme `[0, 1)`.

[34]:
```
a = np.random.rand(10)
a
```

[34]:
```
array([0.35525297, 0.7847035 , 0.51874351, 0.2536128 , 0.96667123,
       0.11041059, 0.90252268, 0.59565457, 0.7285736 , 0.95110809])
```

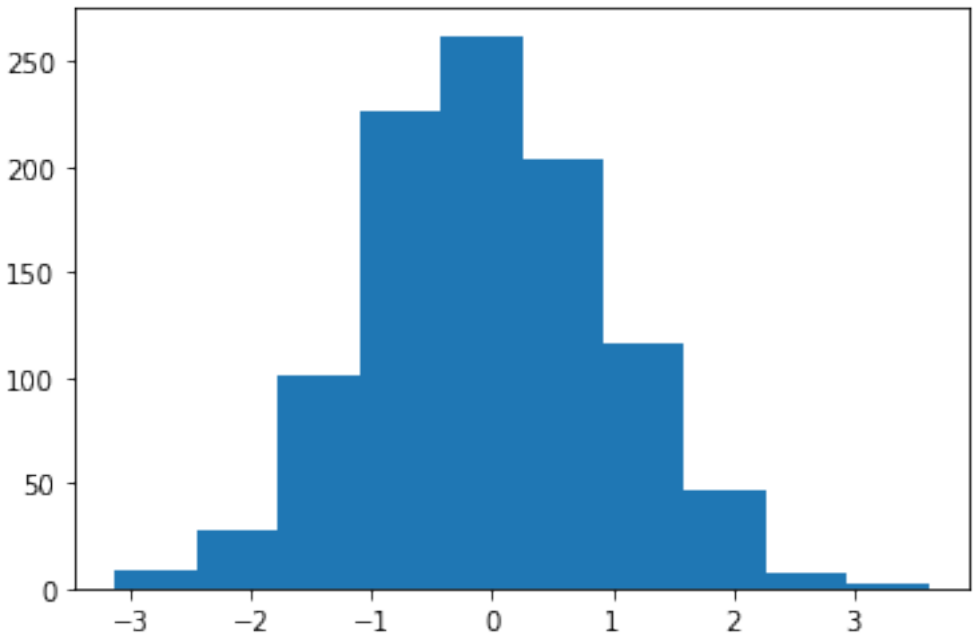
### 0.3.2 randn

Retorne um número (ou números) da distribuição normal. Ao contrário do rand, que é uniforme:

[36]:
```
v = np.random.randn(1000)
```

[37]:
```
import matplotlib.pyplot as plt
plt.hist(v)
```

[37]:
```
(array([  9.,  27., 101., 226., 262., 204., 116.,  46.,   7.,   2.]),
 array([-3.12281529, -2.44865676, -1.77449822, -1.10033968, -0.42618115,
         0.24797739,  0.92213592,  1.59629446,  2.270453  ,  2.94461153,
         3.61877007]),
 <a list of 10 Patch objects>)
```



### 0.3.3 randint

Retornar números inteiros aleatórios

[38]:
```
np.random.randint(0,50,10)
```

[38]:
```
array([38, 25, 25,  3, 20, 45, 39,  5, 45,  9])
```

**Reshape** Retorna uma matriz que contém os mesmos dados com uma nova forma.

[40]:
```
M = np.random.randint(0,100,(5,5))
M
```

[40]:
```
array([[ 3, 10, 26, 58, 70],
       [21, 90, 36, 80, 73],
       [52, 10,  5, 13, 87],
       [84,  6, 76, 92, 70],
       [30, 40, 19, 72, 10]])
```

[41]:
```
M.reshape(1,25)
```

[41]:
```
array([[ 3, 10, 26, 58, 70, 21, 90, 36, 80, 73, 52, 10,  5, 13, 87, 84,
         6, 76, 92, 70, 30, 40, 19, 72, 10]])
```

[42]:
```
v = np.arange(25)
v
```

[42]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

[44]:
```
v.reshape(5,5)
```

[44]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

**max, min, argmax, argmin** Estes são métodos úteis para encontrar valores máximos ou mínimos. Ou para encontrar seus locais de índice usando argmin ou argmax.

[48]:
```
T = np.random.randint(0,100,10)
T
```

[48]:
```
array([94, 92,  8, 96, 55, 76, 46, 81, 94, 42])
```

[49]:
```
T.max()
```

[49]:
```
96
```

[50]:
```
T.min()
```

[50]:
```
8
```

[51]:
```
T.argmin()
```

[51]:
```
2
```

[52]:
```
T.argmax()
```

[52]:
```
3
```

**Aritmética** É possível executar facilmente aritmética de arrays

[53]:
```
a = np.random.randint(0,20,5)
b = np.random.randint(70,150,5)
```

[54]:
```
a
```

[54]:
```
array([18, 18,  9, 10, 15])
```

[55]:
```
b
```

[55]:
```
array([ 72, 129,  92, 112, 135])
```

[56]:
```
a+b
```

[56]:
```
array([ 90, 147, 101, 122, 150])
```

[57]:
```
a-b
```

[57]:
```
array([ -54, -111,  -83, -102, -120])
```

[58]:
```
a*b
```

[58]:
```
array([1296, 2322,  828, 1120, 2025])
```

[59]:
```
a/b
```

[59]:
```
array([0.25      , 0.13953488, 0.09782609, 0.08928571, 0.11111111])
```

[61]:
```
c = np.arange(5)
c
```

[61]:
```
array([0, 1, 2, 3, 4])
```

[62]:
```
a/c
```

```
/home/victor/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1:
RuntimeWarning: divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

[62]:
```
array([       inf, 18.        ,  4.5       ,  3.33333333,  3.75      ])
```

[65]:
```
c
```

[65]:
```
array([0, 1, 2, 3, 4])
```

**Indexação e seleção** É possível selecionar elementos ou grupos de elementos de uma matriz.

[74]:
```
a = np.arange(20); a
```

[74]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

[78]:
```
a[5:12:2]
```

[78]:
```
array([ 5,  7,  9, 11])
```

**Indexing a 2D array (matrices)** O formato geral é:

```
array_2d[linha][linha]
array_2d[linha,linha]
```

[79]:
```
M = np.random.randint(0,50,(4,4))
M
```

[79]:
```
array([[35,  3, 23,  4],
       [26, 16, 20,  0],
       [ 0,  7,  1,  3],
       [15, 21, 39, 30]])
```

[81]:
```
M[0:2,2:4]
```

[81]:
```
array([[23,  4],
       [20,  0]])
```

**Seleção condicional** Vamos ver brevemente como usar colchetes para seleção com base em operadores de comparação.

[83]:
```
a = np.arange(10);a
```

[83]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[89]:
```
b = a[a < 4]
```

[90]:
```
b
```

[90]:
```
array([0, 1, 2, 3])
```

**Exemplo:** Por exemplo, as três equações simultâneas

$$\begin{aligned} 3x - 2y &= 8 \\ -2x + y - 3z &= -20 \\ 4x + 6y + z &= 7 \end{aligned}$$

pode ser representado como a equação $\mathbf{Mx} = \mathbf{b}$:

$$\begin{pmatrix} 3 & -2 & 0 \\ -2 & 1 & -3 \\ 4 & 6 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 8 \\ -20 \\ 7 \end{pmatrix}$$

e resolvido passando matrizes correspondentes à matriz $\mathbf{M}$ e o vetor $\mathbf{b}$ to np.linalg.

[94]:
```
M = np.array([[3,-2,0],[-2,1,-3],[4,6,1]]); M
```

[94]:
```
array([[ 3, -2,  0],
       [-2,  1, -3],
       [ 4,  6,  1]])
```

[95]:
```
b = np.array([8,-20,7]);b
```

[95]:
```
array([  8, -20,   7])
```

[100]:
```
x,y,z = np.linalg.solve(M,b)
```

[101]:
```
x
```

[101]:
```
2.0
```

[102]:
```
y
```

[102]:
```
-0.9999999999999999
```

[103]:
```
z
```

[103]:
```
4.999999999999999
```

**Exemplo:** Calcular os valores próprios da matriz de rotação bidimensional:

$$\mathbf{R} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

que gira pontos no plano xy no sentido anti-horário através $\theta = 30° = \frac{\pi}{6}$ about the origin.

[104]:
```
theta = np.pi/6
```

[105]:
```
c = np.cos(theta)
s = np.sin(theta)
```

[122]:
```
R = np.array([[c,-s],[s,c]])
R
```

[122]:
```
array([[ 0.8660254, -0.5      ],
       [ 0.5      ,  0.8660254]])
```

[108]:
```
aval,avec = np.linalg.eig(R)
```

[121]:
```
np.linalg.det(R)
```

[121]:
```
1.0
```

[125]:
```
np.matrix.transpose(R)
```

[125]:
```
array([[ 0.8660254,  0.5      ],
       [-0.5      ,  0.8660254]])
```

[124]:
```
np.matrix.trace(R)
```

[124]:
```
1.7320508075688774
```

[123]:
```
np.linalg.inv(R)
```

[123]:
```
array([[ 0.8660254,  0.5      ],
       [-0.5      ,  0.8660254]])
```

[114]:
```
lambda1 = aval[0]
```

[115]:
```
lambda2 = aval[1]
```

[120]:
```
np.imag(lambda1)
```

[120]:
```
0.4999999999999999
```

### 0.3.4 Funções:

A estrutura de uma função Python é

```
def nome_da_função(parâmetros):
    instruções
    return valores a serem retornados
```

### 0.3.5 Exemplo

Crie uma função chamada ***roots*** que recebe os valores $a, b$ e $c$ para resolver a equação quadrática $ax^2 + bx + c = 0$ onde $a, b$ e $c$ são números reais e $a \neq 0$.

Solução analítica:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

[141]:
```
def roots(a,b,c):
    import cmath as cm
    x1 = (-b + cm.sqrt((b**2) - (4*a*c))) / (2*a)
    x2 = (-b - cm.sqrt((b**2) - (4*a*c))) / (2*a)
    return x1,x2
```

[142]:
```
a = 1
b = 2
c = 3

x1, x2 = roots(a,b,c)
```

[143]:
```
x1
```

[143]:
```
(-1+1.4142135623730951j)
```

[144]:
```
x2
```

[144]:
```
(-1-1.4142135623730951j)
```