



Pró-reitora de Pós-Graduação, Pesquisa e Inovação
Especialização em Ciências de Dados e Analytics

Programação para Ciência de Dados

Parte 2

Agenda Parte 2

- Biblioteca Numpy
- Biblioteca Pandas
- Biblioteca Matplotlib

Numpy

- Biblioteca que oferece diversos recursos de **álgebra linear** para Python de maneira simples para o Programador, mas com grande eficiência, pois a maioria dos algoritmos foi compilado em C
- Numpy arrays são o core da biblioteca
 - Vetores
 - Matrizes

Instalação

- Com o Anaconda já instalado, o processo de instalação do Numpy é muito simples:
 - conda install numpy
 - OU
 - pip install numpy

Primeiros passos

- Criando arrays

```
import numpy as np
```

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

```
type(lista)
```

```
list
```

```
1 lista_np = np.array(lista)
```

```
type(lista_np)
```

```
numpy.ndarray
```



Carregando uma biblioteca em Python

Primeiros passos

- Criando matrizes

```
mat = [lista, lista, lista]
```

```
mat
```

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

```
mat_np = np.array(mat)
```

```
mat_np
```

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

```
np.size(mat_np)
```

9

Funções básicas

- `np.arange` : similar a função `range`

```
np.arange|
```

Docstring:

```
arange([start,] stop[, step,], dtype=None)
```

Return evenly spaced values within a given interval.

Dica: pressione SHIFT + TAB para ver a documentação de uma função.

Funções básicas

- np.arange
- np.zeros
- np.ones

```
np.arange(1, 10)
```

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

```
np.arange(1, 10, 2)
```

```
array([1, 3, 5, 7, 9])
```

```
np.zeros(5)
```

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

```
np.zeros([3,5])
```

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

```
np.ones([2,3], int)
```

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


Funções básicas

- `np.eye` : cria a matriz identidade da dimensão solicitada

```
mat_id3 = np.eye(3)
```

```
mat_id3
```

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

Funções básicas

- `np.linspace` : similar ao `np.arange`, mas com *step* indicando quantos elementos devem ser gerados na lista

```
np.linspace(1, 10, 3)
```

```
array([ 1. ,  5.5, 10. ])
```

```
np.linspace(1,10,5)
```

```
array([ 1. ,  3.25,  5.5 ,  7.75, 10. ])
```

```
np.linspace(1,10,10)
```

```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.] )
```

Módulo *np.random*

- Módulo usado para inicialização de arrays com valores aleatórios

```
np.random.rand|
```

Docstring:

```
rand(d0, d1, ..., dn)
```

Random values in a given shape.

```
np.random.rand(2, 3)
```

```
array([[0.55241602, 0.85438979, 0.70609091],  
       [0.86233566, 0.33420806, 0.99971444]])
```

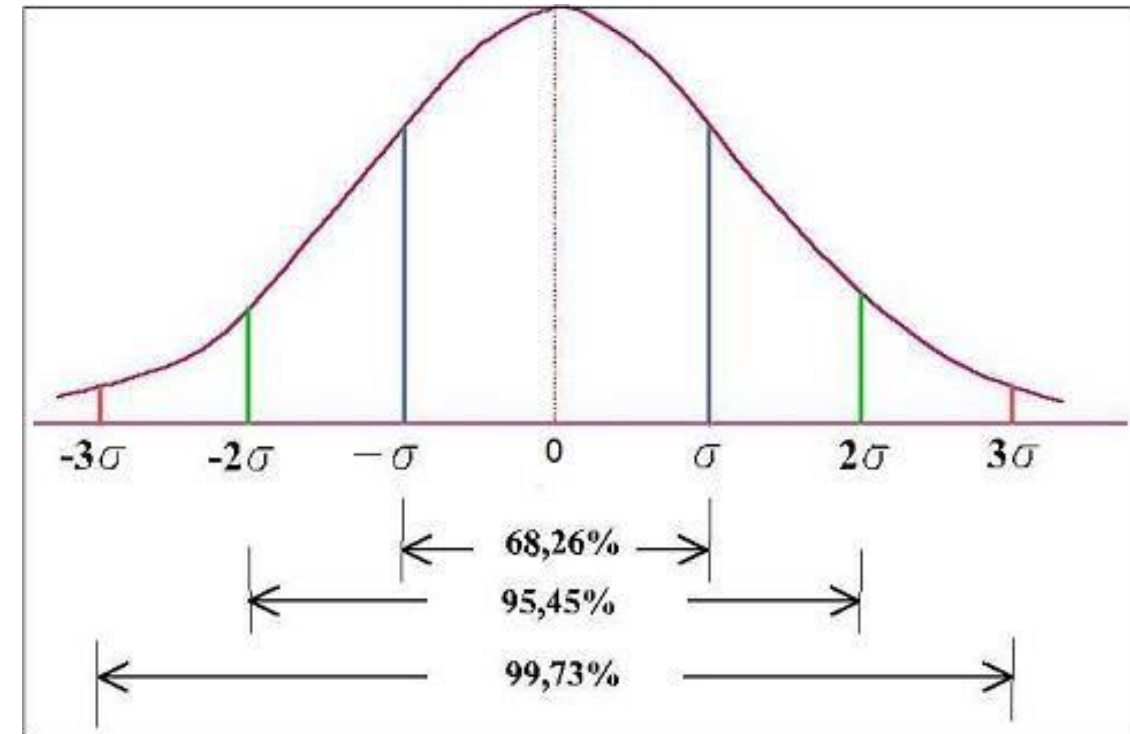
A função rand retorna valores
pertencentes ao intervalo de 0 a 1
com uma
Distribuição Uniforme.

Módulo *np.random*

- randn: Função para geração de valores aleatórios seguindo uma **Distribuição Gaussiana** ($\sim N(\mu=0, \sigma=1)$)

```
np.random.randn(2,3)
```

```
array([[ -0.58486365, -0.08749364, -0.94673822],  
       [-1.01186062,  0.52588823,  0.81092734]])
```



Módulo *np.random*

- randint: Função para geração de números inteiros

```
np.random.randint(0, 100, 10)  
  
array([71, 31, 17, 59, 26, 68, 61, 93, 9, 91])
```

np.random.randint

Docstring:

randint(low, high=None, size=None, dtype='l')

Return random integers from `low` (inclusive) to `high` (exclusive).

Como obter um resultado similar com a função np.random.rand ???

Dimensões do Array

Observe que *shape* é um atributo do array.

reshape é um método que redefine as dimensões do array.

```
dados = np.random.randint(0, 100, 25)
```

```
dados
```

```
array([31, 51,  1, 51, 97, 49, 86, 37, 98, 83,  1, 45, 81, 70,  2, 76, 13,
       87, 18, 47, 24, 99, 59, 58, 25])
```

```
dados.shape
```

```
(25,)
```

```
dados = dados.reshape((5,5))
```

```
dados
```

```
array([[31, 51,  1, 51, 97],
       [49, 86, 37, 98, 83],
       [ 1, 45, 81, 70,  2],
       [76, 13, 87, 18, 47],
       [24, 99, 59, 58, 25]])
```

```
dados.shape
```

```
(5, 5)
```

Funções úteis

```
dados
```

```
array([31, 51,  1, 51, 97, 49, 86, 37, 98, 83,  1, 45, 81, 70,  2, 76, 13,  
      87, 18, 47, 24, 99, 59, 58, 25])
```

```
dados.max()
```

```
99
```

```
dados.argmax()
```

```
21
```

```
dados.min()
```

```
1
```

```
dados.argmin()
```

```
2
```

Indexação de array com numpy

```
dados
```

```
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, 25, 26, 27, 28, 29, 30, 31, 32, 33,  
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])
```

```
dados[10]
```

```
10
```

```
dados[5:10]
```

```
array([5, 6, 7, 8, 9])
```

```
dados[:5]
```

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

```
dados[:5] = 0
```

```
dados
```

```
array([ 0,  0,  0,  0,  0,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,  
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,  
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])
```


Indexação de array com numpy

- Conceito de Ponteiro

```
dados = np.arange(15).reshape(3,5)
```

```
dados
```

```
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14]])
```

```
ponteiro = dados[1][:]
```

```
ponteiro
```

```
array([5, 6, 7, 8, 9])
```

```
ponteiro[:] = 0
```

```
dados
```

```
array([[ 0,  1,  2,  3,  4],  
       [ 0,  0,  0,  0,  0],  
       [10, 11, 12, 13, 14]])
```

```
copia = dados[1][:].copy()
```

```
copia
```

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

```
copia[:] = 1
```

```
dados
```

```
array([[ 0,  1,  2,  3,  4],  
       [ 0,  0,  0,  0,  0],  
       [10, 11, 12, 13, 14]])
```

```
copia
```

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

Indexação de array com numpy

- Selecionando uma região do array

```
dados = np.arange(25)
```

```
dados = dados.reshape(5,5)
```

```
dados
```

```
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]])
```

```
regiao = dados[2:4,2:4]
```

```
regiao[:] = 0
```

```
dados
```

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

```
indices = dados == 0
```

```
indices
```

```
array([[ True, False, False, False, False],
       [False, False, False, False, False],
       [False, False,  True,  True, False],
       [False, False,  True,  True, False],
       [False, False, False, False, False]])
```

```
nulos = dados[indices]
```

```
nulos
```

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

Operações entre arrays com numpy

- Soma
- Subtração
- Multiplicação
- Divisão
- ...

```
vetor = np.arange(10)
```

```
vetor
```

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

```
vetor + vetor
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

```
vetor - vetor
```

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

```
vetor * vetor
```

```
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

```
vetor ** vetor
```

```
array([          1,           1,           4,          27,         256,        3125,
        46656,       823543,    16777216,   387420489], dtype=int32)
```

Operações entre arrays com numpy

```
vetor / vetor
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning: invalid value encountered in true_divide  
    """Entry point for launching an IPython kernel.
```

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

```
1 / vetor
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning: divide by zero encountered in true_divide  
    """Entry point for launching an IPython kernel.
```

```
array([      inf,  1.          ,  0.5          ,  0.33333333,  0.25          ,  
        0.2          ,  0.16666667,  0.14285714,  0.125          ,  0.11111111])
```

Funções Numpy

- ***numpy*** oferece ainda inúmeras funções úteis! Confira:
 - <https://docs.scipy.org/doc/numpy-1.15.1/reference/>

```
np.sqrt(vetor)
```

```
array([0.          , 1.          , 1.41421356, 1.73205081, 2.          ,  
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.          ])
```

```
np.exp(vetor)
```

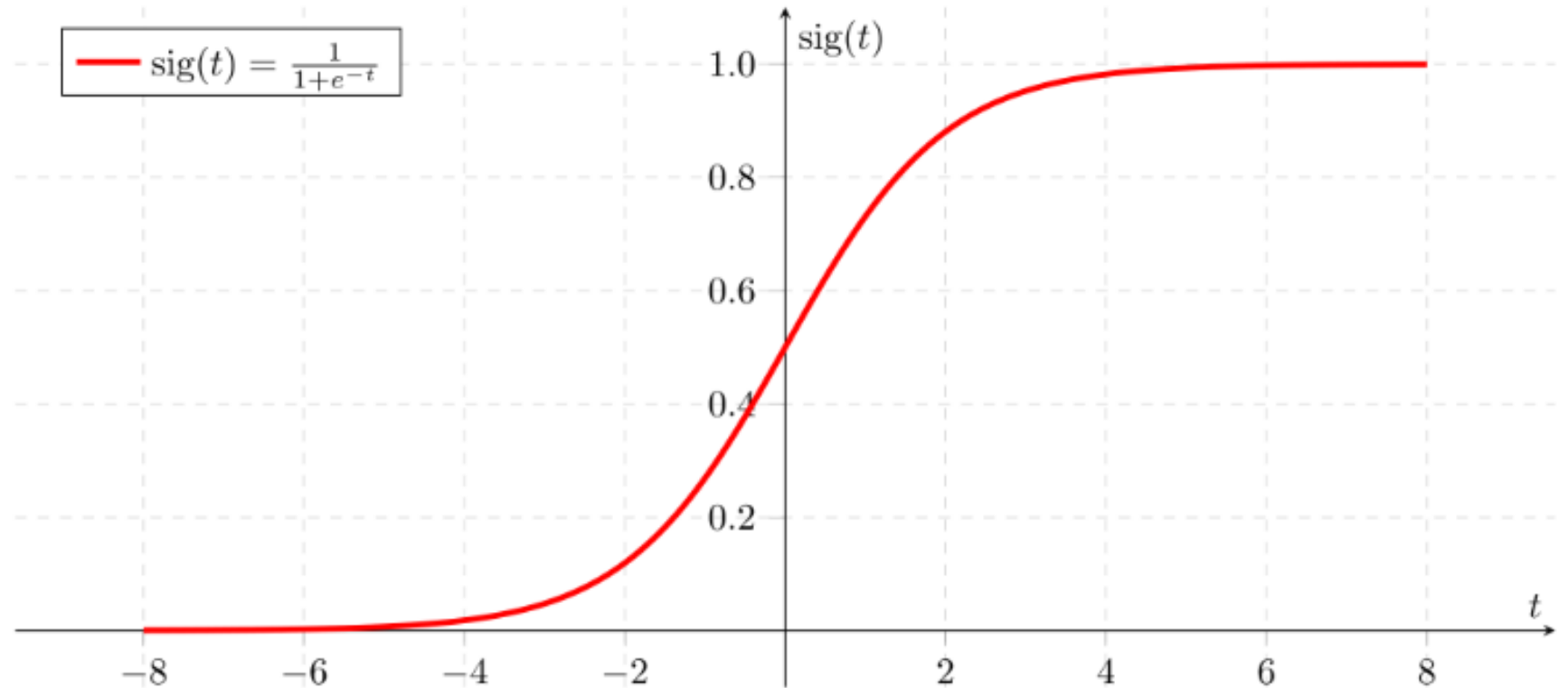
```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,  
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,  
       2.98095799e+03, 8.10308393e+03])
```

```
np.mean(vetor)
```

```
4.5
```

Exercício

- Implemente a função Sigmoid que recebe um numpy array e devolve outro numpy array resultante da aplicação da função sigmoid em cada elemento



Fonte: DeepLearning.AI Course

Broadcasting


- Um recurso importante de Python é o Broadcasting, que permite realizar operações matemáticas sobre numpy arrays de dimensões diferentes.

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

```
s = np.array([[16],
               [12]])
```

```
y = x / s
```

```
([[0.5625, 0.125, 0.3125, 0., 0.],
  [0.58333333, 0.41666667, 0., 0., 0.]])
```



Documentação útil: <https://numpy.org/doc/stable/user/basics.broadcasting.html>

Exercício

- Uma das funções usadas no projeto de redes neurais é a função softmax, abaixo ilustrada. Implemente uma função que recebe como entrada uma matriz numpy e retorna uma nova matriz resultante da aplicação da função softmax.

$$\text{softmax} \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & x_{m3} & \dots & x_{mn} \end{bmatrix} = \begin{bmatrix} \frac{e^{x_{11}}}{\sum_j e^{x_{1j}}} & \frac{e^{x_{12}}}{\sum_j e^{x_{1j}}} & \frac{e^{x_{13}}}{\sum_j e^{x_{1j}}} & \dots & \frac{e^{x_{1n}}}{\sum_j e^{x_{1j}}} \\ \frac{e^{x_{21}}}{\sum_j e^{x_{2j}}} & \frac{e^{x_{22}}}{\sum_j e^{x_{2j}}} & \frac{e^{x_{23}}}{\sum_j e^{x_{2j}}} & \dots & \frac{e^{x_{2n}}}{\sum_j e^{x_{2j}}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{e^{x_{m1}}}{\sum_j e^{x_{mj}}} & \frac{e^{x_{m2}}}{\sum_j e^{x_{mj}}} & \frac{e^{x_{m3}}}{\sum_j e^{x_{mj}}} & \dots & \frac{e^{x_{mn}}}{\sum_j e^{x_{mj}}} \end{bmatrix}$$

Vectorization

- A biblioteca numpy conta com diferentes otimizações no cálculo de operações matriciais. Em se tratando de dados que servirão como entrada para treinamento de um sistema, por exemplo, isso fará grande diferença.
- Em geral os frameworks de aprendizagem que usaremos no curso incorporam essas otimizações, mas é importante entendermos os principais conceitos envolvidos.

Referência útil: <https://www.geeksforgeeks.org/vectorization-in-python/>

Vectorization

$$a \cdot b = \underset{(1 \times n)}{\begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 \end{bmatrix}} \underset{(n \times 1)}{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix}} = \left\{ a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4 + a_5b_5 \right\}$$

Dot Product

Vectorization

$$a \otimes b = ab^T = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}_{(n \times 1)} \begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 \end{bmatrix}_{(1 \times n)} = \left\{ \begin{array}{ccccc} a_1b_1 & a_1b_2 & a_1b_3 & a_1b_4 & a_1b_5 \\ a_2b_1 & a_2b_2 & a_2b_3 & a_2b_4 & a_2b_5 \\ a_3b_1 & a_3b_2 & a_3b_3 & a_3b_4 & a_3b_5 \\ a_4b_1 & a_4b_2 & a_4b_3 & a_4b_4 & a_4b_5 \\ a_5b_1 & a_5b_2 & a_5b_3 & a_5b_4 & a_5b_5 \end{array} \right\}_{(n \times n)}$$

Outer Product

Vectorization

$$a \circ b = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} \circ \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ a_2 b_2 \\ a_3 b_3 \\ a_4 b_4 \\ a_5 b_5 \end{bmatrix}$$

$(n \times 1) \quad (n \times 1) \quad (n \times 1)$

Element wise Product