

centro
universitário



Programação Científica

Prof. Dr. Danilo H. Perico

Numpy



- **NumPy** é um pacote para a linguagem Python que suporta **arrays** e matrizes multidimensionais, possuindo uma larga coleção de funções matemáticas para trabalhar com estas estruturas
- **NumPy** foi criado em 2005 por *Travis Oliphant*
- É um projeto comunitário, multiplataforma, de código livre

<https://numpy.org/>



Apesar de existir desde 2005, um Artigo sobre o NumPy só foi publicado em **16 Setembro de 2020**, na *Nature*

<https://www.nature.com/articles/s41586-020-2649-2>

Review

Array programming with NumPy

<https://doi.org/10.1038/s41586-020-2649-2>

Received: 21 February 2020

Accepted: 17 June 2020

Published online: 16 September 2020

Open access

Check for updates

Charles R. Harris¹, K. Jarrod Millman^{2,3,4}✉, Stéfan J. van der Walt^{2,4,5}✉, Ralf Gommers⁶✉,
Pauli Virtanen^{7,8}, David Cournapeau⁹, Eric Wieser¹⁰, Julian Taylor¹¹, Sebastian Berg⁴,
Nathaniel J. Smith¹², Robert Kern¹³, Matti Picus⁴, Stephan Hoyer¹⁴, Marten H. van Kerkwijk¹⁵,
Matthew Brett^{2,16}, Allan Haldane¹⁷, Jaime Fernández del Río¹⁸, Mark Wiebe^{19,20},
Pearu Peterson^{6,21,22}, Pierre Gérard-Marchant^{23,24}, Kevin Sheppard²⁵, Tyler Reddy²⁶,
Warren Weckesser⁴, Hameer Abbas⁶, Christoph Gohlke²⁷ & Travis E. Oliphant⁶

Array programming provides a powerful, compact and expressive syntax for accessing, manipulating and operating on data in vectors, matrices and higher-dimensional arrays. NumPy is the primary array programming library for the Python language. It has an essential role in research analysis pipelines in fields as diverse as physics, chemistry, astronomy, geoscience, biology, psychology, materials science, engineering, finance and economics. For example, in astronomy, NumPy was an important part of the software stack used in the discovery of gravitational waves¹ and in the first imaging of a black hole². Here we review how a few fundamental array concepts lead to a simple and powerful programming paradigm for organizing, exploring and analysing scientific data. NumPy is the foundation upon which the scientific Python ecosystem is constructed. It is so pervasive that several projects, targeting audiences with specialized needs, have developed their own NumPy-like



- **NumPy** é bastante útil para executar várias tarefas matemáticas como integração numérica, diferenciação, interpolação, extrapolação e muitas outras
- O **NumPy** possui também funções incorporadas para álgebra linear, transformadas de fourier e geração de números aleatórios
- Quando usada em conjunto com **Matplotlib**, por exemplo, pode **substituir** programas como o **Matlab** para o tratamento de tarefas matemáticas



***NumPy is the base of
the Scientific Python
ecosystem***

Application-specific

cesium PyChrono MDAnalysis eht-imaging iris
khmer PsychoPy Qiime2 FiPy deepchem
nibabel mne-python yellowbrick scikit-HEP
PyWavelets librosa SunPy QuTiP yt

Domain-specific

Astropy Biopython NLTK
Astronomy Biology Linguistics
QuantEcon cantera simpeg
Economics Chemistry Geophysics

Technique-specific

scikit-learn scikit-image NetworkX
Machine learning Image processing Network analysis
pandas, statsmodels Statistics

Foundation

SciPy Matplotlib
Algorithms Plots

Python NumPy IPython / Jupyter
Language Arrays Interactive environments
New array implementations

NumPy API ————— Array Protocols -----



Exemplo de Uso - Processamento de Imagens

- Imagens no computador são representadas como Arrays Multidimensionais de números
- NumPy torna-se a escolha mais natural para essa tarefa
- O NumPy, na verdade, fornece algumas excelentes funções de biblioteca para rápida manipulação de imagens
- Alguns exemplos são o espelhamento de uma imagem, a rotação de uma imagem por um determinado ângulo etc.



Exemplo de Uso - Machine Learning

- Ao escrever algoritmos de Machine Learning, supõe-se que se realizem vários cálculos numéricos em Array
 - Por exemplo, multiplicação de Arrays, transposição, adição, etc.
- O NumPy fornece uma excelente biblioteca para cálculos
- Os Arrays NumPy são usados para armazenar os dados de treinamento, bem como os parâmetros dos modelos de Machine Learning



Por que usar?

- Em Python usamos listas para tratar vetores e matrizes, mas elas são lentas para processar
- O **NumPy** fornece um **objeto array** que é até **50x** mais rápido que o Python tradicional
 - O objeto array em NumPy fornece funções de suporte que tornam o trabalho com vetores e matrizes mais fácil
- Os arrays são muito frequentemente usados em ciência de dados, onde velocidade e recursos são muito importantes



Por que usar?

- Os arrays **NumPy** são armazenados em um lugar contínuo na memória - ao contrário das listas, para que os processos possam manipulá-los de forma muito eficiente
 - Esse comportamento é chamado de localidade de referência em ciência da computação.
- Esta é a principal razão pela qual o **NumPy** é mais rápido do que as listas
- Também é otimizado para trabalhar com as mais recentes arquiteturas de CPU



NumPy

Instalação

Já vem instalado no Anaconda!

Se mesmo assim precisar instalar:

- If you use conda, you can install it with:

`conda install numpy`

- If you use pip, you can install it with:

`pip install numpy`

- If you use Linux, you can install via terminal with:

`sudo apt-get install python-numpy`



Primeiros Passos

- Para utilizar a biblioteca NumPy em um programa, é necessário importá-la:

```
import numpy
```

ou:

```
import numpy as np
```

- Podemos exibir a versão instalada:

```
print(np.__version__)
```



NumPy Criando Arrays

- O objeto **array** em NumPy é chamado **ndarray** (*n dimensional array*)
- Podemos criar um objeto **ndarray** usando a função **array()**:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(type(arr))
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```



Criando Arrays

Command

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



NumPy Array

1
2
3



Criando Arrays

- Para criar um array, podemos passar uma lista, tupla ou qualquer objeto semelhante a matriz no método array() e ele será convertido em um objeto do tipo ndarray:

```
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

```
[1 2 3 4 5]
```

```
arr = np.array([[1, 2], [3, 4]])
print(arr)
```

```
[[1 2]
 [3 4]]
```



Criando Arrays

- Podemos criar um array com elementos sequenciais usando o método `arange()`

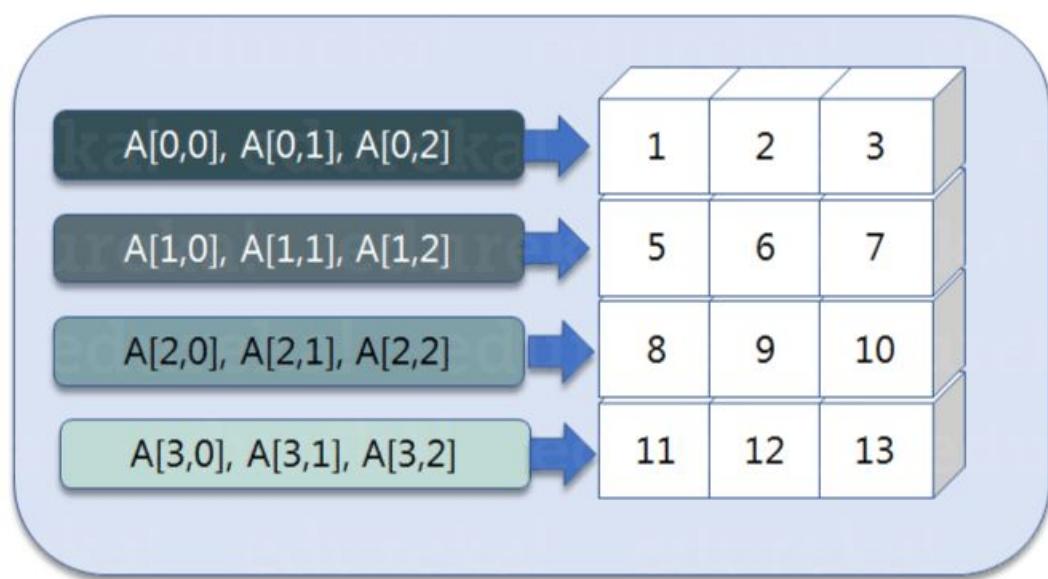
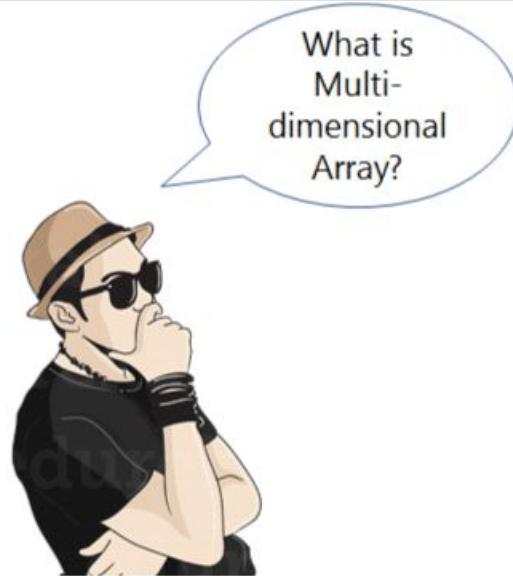
```
arr = np.arange(15)  
print(arr)
```

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



Dimensões em Arrays

edureka!





Dimensões em Arrays

- 0D: Representa um número escalar

```
arr = np.array(42)
print(arr)
```

42

- 1D: Representa um vetor

```
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

[1 2 3 4 5]



Dimensões em Arrays

- 2D:
 - Um array que tem arrays 1D como seus elementos é chamado de array 2D
 - São frequentemente usados para representar matrizes

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



Dimensões em Arrays

- 3D:
 - Um array que tem arrays 2D como seus elementos é chamado de array 3D
 - São frequentemente usados para representar um tensor de 3^a ordem

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

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

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



Dimensões Superiores: Tensores

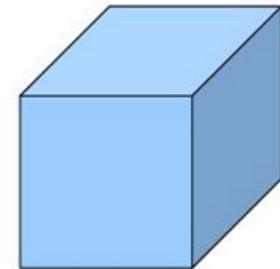
- Arrays multidimensionais podem ser chamados de **tensores***



1d-tensor



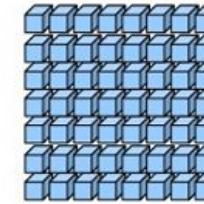
2d-tensor



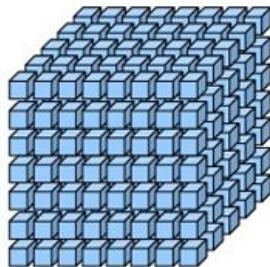
3d-tensor



4d-tensor



5d-tensor



6d-tensor

Descobrindo a dimensão de um array

- O objeto **array** fornece o atributo **ndim** que retorna um inteiro que nos diz quantas dimensões o array tem:

```
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2], [3, 4]], [[5, 6], [7 ,8]]])
e = np.array([[[[1, 2], [3, 4]], [[5, 6], [7 ,8]]], [[[1, 2], [3, 4]], [[5, 6], [7 ,8]]]])
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
print(e.ndim)
```

```
0
1
2
3
4
```



Indíces

- Os elementos dos **arrays** são acessados da mesma forma que nas listas

```
arr = np.array([1, 2, 3, 4])
print(arr[0])
```

1

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print(arr[0][1])
```

2

```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0][1][2])
```

6



Indíces

- Mas, **arrays** de numpy também podem ser acessados utilizando colchetes, com as dimensões separadas por vírgulas:

```
arr = np.array([1, 2, 3, 4])
print(arr[0])
```

1

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print(arr[0, 1])
```

2

```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
```

6



Indíces Negativos

- Igual às listas, os índices negativos indexam os elementos a partir do final do array:

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('Último elemento da segunda lista: ', arr[1, -1])
```

```
Último elemento da segunda lista: 10
```



Slicing - Fatiamento

- Fatiar em Python significa pegar elementos de um dado índice até outro índice
- Para fatiar usamos os índices como:
 - `[start : end]`
 - Se não definirmos o *start*, considera-se início em 0
 - Se não passarmos o *end*, considera-se até o final da dimensão
- Também podemos definir o passo, assim:
 - `[start : end : step]`



NumPy Slicing - Fatiamento

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

```
[2 3 4 5]
```

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
```

```
[5 6 7]
```

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[:4])
```

```
[1 2 3 4]
```

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2])
```

```
[2 4]
```



NumPy Slicing - Fatiamento

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])  
print(arr[1, 1:4])
```

```
[7 8 9]
```

```
print(arr[0:2, 2])
```

```
[3 8]
```

```
print(arr[0:2, 1:4])
```

```
[[2 3 4]  
 [7 8 9]]
```



NumPy Slicing - Fatiamento

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

- Capturando uma linha:
 - `print(arr[1, :])`
- Capturando uma coluna:
 - `print(arr[:, 1])`

```
print(arr[1, :])
```

```
[6 7 8]
```

```
print(arr[:, 1])
```

```
[2 7 8]
```



Slicing - Fatiamento

	<code>data</code>	<code>data[0]</code>	<code>data[1]</code>	<code>data[0:2]</code>	<code>data[1:]</code>	<code>data[-2:]</code>
0	1			1		
1	2		2	2	2	2
2	3				3	3



NumPy

Slicing - Fatiamento

data		
0	1	
0	1	2
1	3	4
2	5	6

data[0,1]		
0	1	
0	1	2
1	3	4
2	5	6

data[1:3]		
0	1	2
0	1	2
1	3	4
2	5	6

data[0:2,0]		
0	1	2
0	1	2
1	3	4
2	5	6



Slicing - Fatiamento e Seleção

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

- Podemos facilmente pegar todos os valores menores do que 5, por exemplo:

```
print(a[a < 5])
```

```
[ 1  2  3  4  4  2 -1]
```



Slicing - Fatiamento e Seleção

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

- Você também pode selecionar, por exemplo, números iguais ou maiores que 5 e usar essa condição para indexar uma matriz.

```
five_up = (a >= 5)
print(a[five_up])
```

```
[5 6 7 8 9]
```



Slicing - Fatiamento e Seleção

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

- Podemos selecionar elementos que satisfazem duas condições usando os operadores lógicos `&` e `|`:

```
c = a[(a > 2) & (a < 11)]
```

```
print(c)
```

```
[3 4 5 6 7 8 9 4]
```



Slicing - Fatiamento e Seleção

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

- Podemos usar também os operadores `&` e `|` para retornar valores booleanos que especificam se os valores em uma matriz atendem ou não a uma determinada condição:

```
five_up = (a > 5) | (a == 5)
print(five_up)
```

```
[[False False False False]
 [ True  True  True  True]
 [ True False False False]]
```



Tipos de Dados

- O NumPy define mais tipos de dados do que os existentes em Python:
 - i - integer
 - b - boolean
 - u - unsigned integer
 - f - float
 - c - complex float
 - m - timedelta
 - M - datetime
 - O - object
 - S - string
 - U - unicode string
 - V - fixed chunk of memory for other type (void)
- Para descobrir o tipo de dado em NumPy use `dtype()`



Tipos de Dados

```
arr = np.array([1, 2, 3, 4])  
print(arr.dtype)
```

int32

int32 - tipo int com 32 bits

```
arr = np.array([1.0, 2.0, 3.0, 4.0])  
print(arr.dtype)
```

float64

float64 - tipo float com 64 bits

```
arr = np.array(['apple', 'banana', 'cherry'])  
print(arr.dtype)
```

<U6

U para Unicode - 6 é o número de bytes da string mais longa



Tipos de Dados

- A função `array()` usada para criar arrays pode ter um argumento opcional - `dtype` - que nos permite definir o tipo de dados esperado dos elementos de matriz:

```
arr = np.array([1, 2, 3, 4], dtype='U')
```

```
print(arr)
```

```
['1' '2' '3' '4']
```

- `arr` é um array de *unicode string (U)* dos números.



Tipos de Dados

- Para os tipos i, u, f, S e U podemos definir seu tamanho
 - i - integer
 - u - unsigned integer
 - f - float
 - S - string
 - U - unicode string
- Para os números, usamos 8, 16, 32, 64 bits
- Para strings, definimos a quantidade de caracteres



Tipos de Dados

```
arr = np.array([1, 2, 3, 4], dtype='int64')
```

```
print(arr)
```

```
[1 2 3 4]
```

```
arr.dtype
```

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

- **arr** é um array de *inteiros com 64 bits*



Tipos de Dados

Exemplos:

- int8 - byte (-128 to 127)
- int16 - integer (-32768 to 32767)
- int32 - integer (-2147483648 to 2147483647)
- int64 - integer (-9223372036854775808 to 9223372036854775807)
- uint8 - unsigned integer (0 to 255)
- uint16 - unsigned integer (0 to 65535)



Tipos de Dados

- Para converter entre tipos, utilizamos a função `astype()`

```
arr = np.array([[0,4,5],[8,7,6]])
print( arr.dtype )
arr = arr.astype(np.int8)
print(arr.dtype)
```

int32

int8



NumPy

Copiando Arrays

- Para copiar um array é necessário utilizar a função `copy()` :

```
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42
print(arr)
print(x)
```

```
[42  2   3   4   5]
[1  2  3  4  5]
```



Copiando Arrays - view

- Quando não usamos a função `copy()`, criamos apenas uma referência para o array original (`view`):

```
arr = np.array([1, 2, 3, 4, 5])
x = arr
arr[0] = 42
print(arr)
print(x)
```

```
[42  2  3  4  5]
[42  2  3  4  5]
```



Array Shape

- O **shape**, ou forma, de um array, é dado pelo número de elementos em cada dimensão
- Os arrays NumPy têm um atributo chamado shape que retorna uma tupla com cada índice tendo o número de elementos correspondentes em cada dimensão:

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

```
(2, 3)
```



Reshaping

- **Reshape** significa mudar a forma de um array
- Com o **reshaping** podemos adicionar ou remover dimensões ou alterar o número de elementos em cada dimensão
- Muito usado em ciência de dados e Machine Learning
- Utiliza-se o método **reshape (newshape)**
 - **newshape** define a nova forma do array
 - Pode ter qualquer forma, desde que o número de elementos do novo array seja igual ao original



Reshaping

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
newarr = arr.reshape(4, 2)
print(arr)
print(newarr)
```

```
[1 2 3 4 5 6 7 8]
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
```



Reshaping

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
newarr = arr.reshape(2, 4)
print(newarr)
```

```
[[1 2 3 4]
 [5 6 7 8]]
```



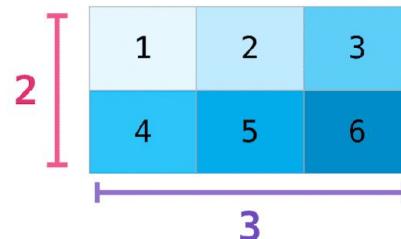
NumPy

Reshaping

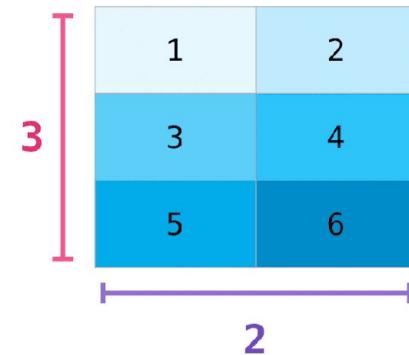
`data`

1
2
3
4
5
6

`data.reshape(2,3)`



`data.reshape(3,2)`





NumPy Reshaping

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
newarr = arr.reshape(2, 2, 2)
print(newarr)
```

```
[[[1 2]
  [3 4]]
```

```
[[5 6]
  [7 8]]]
```



NumPy Reshaping

```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
newarr2 = arr.reshape(8)
print(arr)
print(newarr2)
```

```
[[1 2 3 4]
 [5 6 7 8]]
[1 2 3 4 5 6 7 8]
```



Reshaping

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
newarr = arr.reshape(2, 2, 2)
newarr2 = newarr.reshape(-1)
print(newarr2)
```

```
[1 2 3 4 5 6 7 8]
```

- *Flattening:*

- Reduz qualquer array multidimensional à uma dimensão



Iteração em Array

- Iterar em um array NumPy é semelhante a iterar uma lista
- Utiliza-se um comando de repetição como o **for**:

```
arr = np.array([1, 2, 3])
for x in arr:
    print(x)
```

```
1
2
3
```



Iteração em Array - 2D

- Iterar em um array NumPy é semelhante a iterar uma lista
- Utiliza-se um comando de repetição como o **for**:

```
arr = np.array([[1, 2, 3],[4, 5, 6]])
for x in arr:
    print(x)
```

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



NumPy

Iteração em Array - 2D

- Iterar em um array NumPy é semelhante a iterar uma lista
- Utiliza-se um comando de repetição como o **for**:

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
for x in arr:
    for y in x:
        print(y, end=" ")
    print()
```

```
1 2 3
4 5 6
```



NumPy

Iteração em Array - 3D

- Iterar em um array NumPy é semelhante a iterar uma lista
- Utiliza-se um comando de repetição como o **for**:

```
arr = np.array([[[1,2],[3,4]],[[5, 6],[7,8]]])
for x in arr:
    for y in x:
        for z in y:
            print(z, end=" ")
```

1 2 3 4 5 6 7 8



Busca em Arrays

- Para pesquisar um array, use o método `where()`:

```
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
print(x)

(array([3, 5, 6], dtype=int64),)
```



Busca em Arrays

- Para pesquisar um array, use o método `where()`:

```
arr = np.array([[1, 2, 3, 4], [4, 5, 4, 0]])
x = np.where(arr == 4)
print(x)
print(x[0])
print(x[1])

(array([0, 1, 1], dtype=int64), array([3, 0, 2], dtype=int64))
[0 1 1]
[3 0 2]
```



Ordenação de Arrays

- Ordenar significa colocar elementos em uma sequência ordenada
- Sequência ordenada é qualquer sequência que tem uma ordem correspondente a elementos, como numérico ou alfabético, ascendente ou descendente.
- O objeto array em NumPy tem uma função chamada `sort()`, que classifica um array especificado.



NumPy Ordenação de Arrays

```
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))
```

```
[0 1 2 3]
```

```
arr = np.array(['banana', 'cherry', 'apple'])
print(np.sort(arr))
```

```
['apple' 'banana' 'cherry']
```

```
arr = np.array([[3, 2, 4], [5, 0, 1]])
print(np.sort(arr))
```

```
[[2 3 4]
 [0 1 5]]
```



Operações entre Arrays

- O NumPy define diversas operações matemáticas entre matrizes, otimizadas para velocidade de processamento
- Todas as operações básicas são definidas:
 - Soma, subtração, multiplicação e divisão escalar de vetores
 - Multiplicação vetorial
 - Multiplicação de matrizes

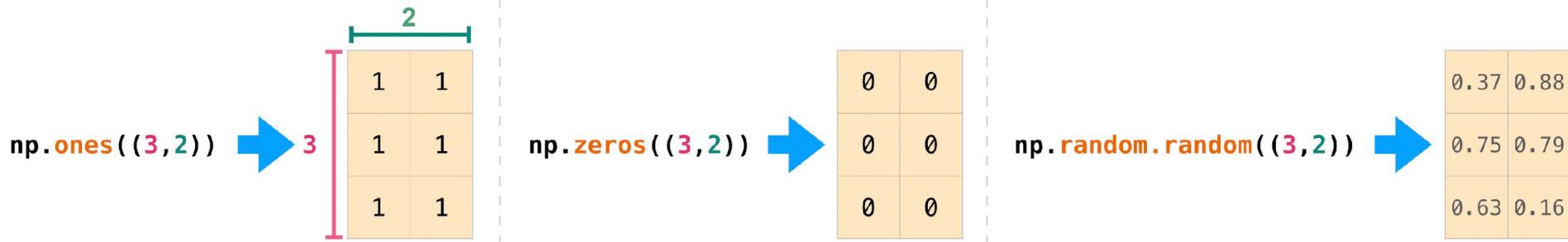


Criação de Matrizes

```
np.ones((3, 2))
```

```
np.zeros((3, 2))
```

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





NumPy Soma Escalar

```
data = np.array([1, 2])
ones = np.ones(2, dtype=int)
data + ones
```

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

$$\text{data} + \text{ones} = \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline \end{array} + \begin{array}{|c|} \hline 1 \\ \hline 1 \\ \hline \end{array} = \begin{array}{|c|} \hline 2 \\ \hline 3 \\ \hline \end{array}$$



NumPy

Subtração, Multiplicação e Divisão Escalar

```
data = np.array([1, 2])
ones = np.ones(2, dtype=int)
```

data - ones

$$\begin{array}{c} \text{data} \\ \hline 1 \\ 2 \end{array} \quad - \quad \begin{array}{c} \text{ones} \\ \hline 1 \\ 1 \end{array} \quad = \quad \begin{array}{c} 0 \\ \hline 1 \end{array}$$

data * data

$$\begin{array}{c} \text{data} \\ \hline 1 \\ 2 \end{array} \quad * \quad \begin{array}{c} \text{data} \\ \hline 1 \\ 2 \end{array} \quad = \quad \begin{array}{c} 1 \\ \hline 4 \end{array}$$

data / data

$$\begin{array}{c} \text{data} \\ \hline 1 \\ 2 \end{array} \quad / \quad \begin{array}{c} \text{data} \\ \hline 1 \\ 2 \end{array} \quad = \quad \begin{array}{c} 1 \\ \hline 1 \end{array}$$



Broadcast: Multiplicação por um escalar

```
data = np.array([1.0, 2.0])  
data * 1.6
```

```
array([1.6, 3.2])
```

1
2

* 1.6 =

1
2

* 1.6 =

1.6
3.2



NumPy Soma de Matrizes

```
data = np.array([[1, 2], [3, 4]])  
ones = np.array([[1, 1], [1, 1]])  
data + ones
```

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

$$\text{data} + \text{ones} = \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} + \begin{array}{|c|c|}\hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array} = \begin{array}{|c|c|}\hline 2 & 3 \\ \hline 4 & 5 \\ \hline \end{array}$$



NumPy Soma de Matrizes - Tamanhos Diferentes

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

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

$$\begin{array}{c} \text{data} \\ \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline 5 & 6 \\ \hline \end{array} \end{array} + \begin{array}{c} \text{ones_row} \\ \begin{array}{|c|c|} \hline 1 & 1 \\ \hline \end{array} \end{array} = \begin{array}{c} \text{data} \\ \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline 5 & 6 \\ \hline \end{array} \end{array} + \begin{array}{c} \text{ones_row} \\ \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array} \end{array} = \begin{array}{|c|c|} \hline 2 & 3 \\ \hline 4 & 5 \\ \hline 6 & 7 \\ \hline \end{array}$$



NumPy

Multiplicação de Matrizes

```
data = np.array([[1, 2], [3, 4]])
data2 = np.array([[5, 6], [7, 8]])
data * data2
```

```
array([[ 5, 12],
       [21, 32]])
```

$$\text{data} \quad \quad \quad \text{data2}$$
$$\begin{matrix} \text{data} \\ \hline \begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} \end{matrix} \quad \quad \quad \begin{matrix} \text{data2} \\ \hline \begin{matrix} 5 & 6 \\ 7 & 8 \end{matrix} \end{matrix} = \begin{matrix} \begin{matrix} 5 & 12 \\ 21 & 32 \end{matrix} \end{matrix}$$
$$\text{data} * \text{data2} =$$



Produto Escalar

- O produto escalar é uma operação definida entre dois vetores que fornece um número real (*também chamado "escalar"*) como resultado
 - É o produto interno padrão do espaço euclidiano
- Algebricamente, o produto escalar de dois vetores é formado pela multiplicação de seus componentes correspondentes e pela soma dos produtos resultantes.
 - Geometricamente, é o produto das magnitudes euclidianas dos dois vetores e o cosseno do ângulo entre eles.



Produto Escalar

$$\sum_{i=0}^{n-1} x_i * y_i = x_0 * y_0 + x_1 * y_1 + \dots + x_n * y_n$$

$$[a \ b] \cdot \begin{bmatrix} x \\ y \end{bmatrix} = a.x + b.y$$

$$[a \ b \ c] \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = a.x + b.y + c.z$$

$$[a \ b \ c \ d] \cdot \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} = a.x + b.y + c.z + d.t$$



Produto Escalar

- A função chamada `dot()` é usada para realizarmos o produto escalar.

```
data = np.array([1, 2])
data2 = np.array([3, 4])
x = data.dot(data2)
print(x)
```

11



Produto Escalar

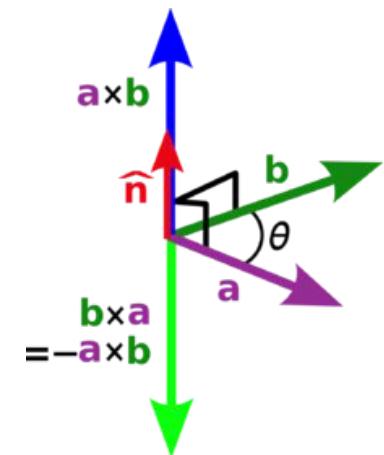
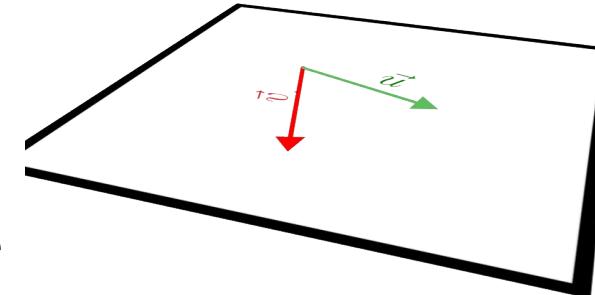
```
data = np.array([1, 2, 3])
data2 = np.array([4, 5, 6])
x = data.dot(data2)
print(x)
```

32



Produto Vetorial

- O produto vetorial é uma operação sobre dois vetores em um espaço tridimensional e é denotado por \times
 - Dados dois vetores independentes linearmente a e b , o produto vetorial $a \times b$ é um vetor perpendicular ao vetor a e ao vetor b e é a normal do plano contendo os dois vetores.
- Seu resultado difere do produto escalar por ser também um vetor, ao invés de um escalar.





NumPy Produto Vetorial

- Em NumPy, o produto vetorial é obtido com a função *cross()*

```
data = np.array([1, 2, 3])
data2 = np.array([4, 5, 6])
x = np.cross(data, data2)
print(x)
```

```
[ -3  6 -3]
```



NumPy Produto Vetorial

```
data = np.array([[1,2,3],[4,5,6],[1,0,0]])
data2 = np.array([[4,5,6],[4,5,6],[0,1,0]])
x = np.cross(data, data2)
print(x)
```

```
[[ -3   6  -3]
 [  0   0   0]
 [  0   0   1]]
```



Multiplicação de Matrizes

$$\begin{matrix} & m \\ \left| \begin{matrix} & & \\ & & \\ & & \\ & & \\ & & \\ & & \end{matrix} \right. & \cdot & \left| \begin{matrix} & & & \\ & & & \\ & & & \\ & & & \\ & & & \end{matrix} \right. & = & \left| \begin{matrix} & & & \\ & & & \\ & & & \\ & & & \\ & & & \end{matrix} \right. \\ | & & m & & | & n \\ A & \cdot & B & = & C \end{matrix}$$

número de colunas de A = número de linhas de B



Multiplicação de Matrizes

- A função `dot()` é também usada para realizarmos a multiplicação de matrizes

```
data = np.array([[1, 2], [3, 4]])
data2 = np.array([[5, 6], [7, 8]])
x = data.dot(data2)
print(x)
```

```
[[19 22]
 [43 50]]
```



NumPy

Matriz Transposta

```
a = np.array([[1, 2], [3, 4]])  
a.transpose()
```

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

```
a = np.array([[1, 2], [3, 4]])  
a.T
```

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

data

1	2
3	4
5	6

data.T

1	3	5
2	4	6



Inversão de Matriz

- A matriz inversa de uma matriz é tal que se for multiplicada pela matriz original, resulta em matriz identidade
- Usamos a função `numpy.linalg.inv()` para calcular a inversa de uma matriz



NumPy Inversão de Matriz

```
x = np.array([[1,2],[3,4]])
y = np.linalg.inv(x)
print (x)
print()
print (y)
print()
print (np.dot(x,y))
```

```
[[1 2]
 [3 4]]
```

```
[[ -2.   1. ]
 [ 1.5 -0.5]]
```

```
[[1.00000000e+00 1.11022302e-16]
 [0.00000000e+00 1.00000000e+00]]
```



Máximo, mínimo, soma

data

1
2
3

.max() = 3

data

1
2
3

.min() = 1

data

1
2
3

.sum() = 6



Máximo, mínimo, soma em Matrizes

data

1	2
3	4
5	6

.max() = 6

data

1	2
3	4
5	6

.min() = 1

data

1	2
3	4
5	6

.sum() = 21



Máximo, mínimo, soma em Matrizes

`data.max(axis=0)` // vai olhar para cada coluna

`data.max(axis=1)` // vai olhar para cada linha

data

1	2
5	3
4	6

1	2
5	3
4	6

$$\cdot \text{max}(\text{axis}=0) = \begin{matrix} 5 \\ 6 \end{matrix}$$

data

1	2
5	3
4	6

$$\cdot \text{max}(\text{axis}=1) = \begin{matrix} 5 \\ 6 \end{matrix}$$

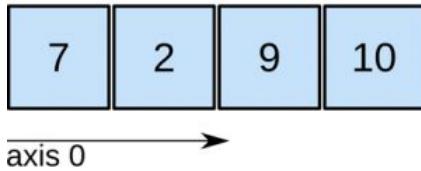
1	2
5	3
4	6

$$= \begin{matrix} 5 \\ 6 \end{matrix}$$



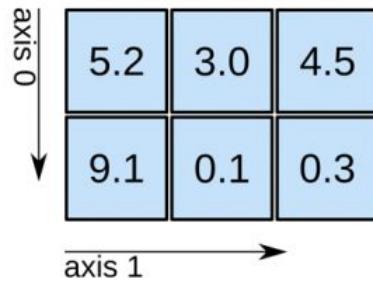
NumPy Axis

1D array



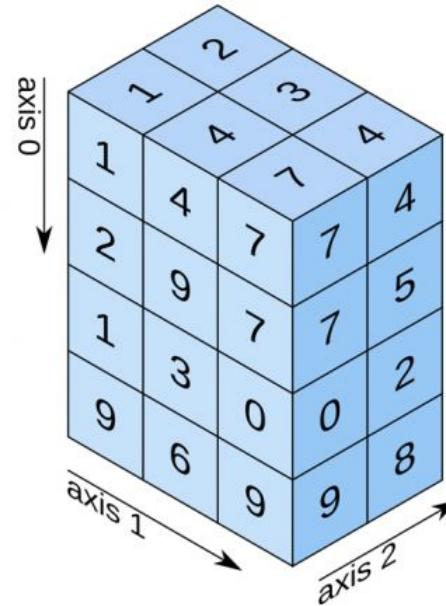
shape: (4,)

2D array



shape: (2, 3)

3D array



shape: (4, 3, 2)



Exercício 1

Utilizando NumPy, crie a Matriz A e a matriz B, conforme abaixo:

$$A = \begin{bmatrix} 12 & 9 & 4 & 1 \\ 11 & 5 & 8 & 1 \\ 1 & 2 & 3 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 5 \\ 1 & 7 \\ 1 & 9 \\ 1 & 1 \end{bmatrix}$$

Crie, então, a Matriz C, sendo $C = A \cdot B$

Calcule e mostre também a média e o desvio padrão das linhas da Matriz C e a média e o desvio padrão das colunas da Matriz C.



Exercício 2

Utilizando NumPy, descubra a solução para o sistema:

$$\begin{cases} 3x + 2y = 10 \\ x - y = 2 \end{cases}$$



Números Aleatórios

- NumPy possui o módulo ***random*** para a geração de números aleatórios
- Ele possui duas funções básicas:
 - ***rand ()***: retorna um número real entre 0 e 1
 - ***randint (n)***: retorna um número real entre 0 e n



Números Aleatórios

- NumPy possui diversos tipos de distribuições probabilísticas

Distributions

<code>beta(a, b[, size])</code>	Draw samples from a Beta distribution.
<code>binomial(n, p[, size])</code>	Draw samples from a binomial distribution.
<code>chisquare(df[, size])</code>	Draw samples from a chi-square distribution.
<code>dirichlet(alpha[, size])</code>	Draw samples from the Dirichlet distribution.
<code>exponential([scale, size])</code>	Draw samples from an exponential distribution.
<code>f(dfnum, dfden[, size])</code>	Draw samples from an F distribution.
<code>gamma(shape[, scale, size])</code>	Draw samples from a Gamma distribution.
<code>geometric(p[, size])</code>	Draw samples from the geometric distribution.
<code>gumbel([loc, scale, size])</code>	Draw samples from a Gumbel distribution.
<code>hypergeometric/ngood, nbad, nsample[, size]</code>	Draw samples from a Hypergeometric distribution.
<code>laplace([loc, scale, size])</code>	Draw samples from the Laplace or double exponential distribution with specified location (or mean) and scale (decay).
<code>logistic([loc, scale, size])</code>	Draw samples from a logistic distribution.
<code>lognormal([mean, sigma, size])</code>	Draw samples from a log-normal distribution.
<code>logseries(p[, size])</code>	Draw samples from a logarithmic series distribution.
<code>multinomial(n, pvals[, size])</code>	Draw samples from a multinomial distribution.
<code>multivariate_hypergeometric(colors, nsample[, size])</code>	Generate variates from a multivariate hypergeometric distribution.
<code>multivariate_normal(mean, cov[, size, ...])</code>	Draw random samples from a multivariate normal distribution.
<code>negative_binomial(n, p[, size])</code>	Draw samples from a negative binomial distribution.
<code>noncentral_chisquare(df, nonc[, size])</code>	Draw samples from a noncentral chi-square distribution.
<code>noncentral_f(dfnum, dfden, nonc[, size])</code>	Draw samples from the noncentral F distribution.
<code>normal([loc, scale, size])</code>	Draw random samples from a normal (Gaussian) distribution.
<code>pareto(a[, size])</code>	Draw samples from a Pareto II or Lomax distribution with specified shape.
<code>poisson([lam, size])</code>	Draw samples from a Poisson distribution.
<code>power(a[, size])</code>	Draws samples in [0, 1] from a power distribution with positive exponent a - 1.
<code>rayleigh([scale, size])</code>	Draw samples from a Rayleigh distribution.
<code>standard_cauchy([size])</code>	Draw samples from a standard Cauchy distribution with mode = 0.
<code>standard_exponential([size, dtype, method, out])</code>	Draw samples from the standard exponential distribution.
<code>standard_gamma(shape[, size, dtype, out])</code>	Draw samples from a standard Gamma distribution.
<code>standard_normal([size, dtype, out])</code>	Draw samples from a standard Normal distribution (mean=0, stddev=1).
<code>standard_t(df[, size])</code>	Draw samples from a standard Student's t distribution with df degrees of freedom.
<code>triangular(left, mode, right[, size])</code>	Draw samples from the triangular distribution over the interval [left, right].
<code>uniform([low, high, size])</code>	Draw samples from a uniform distribution.
<code>vonmises(mu, kappa[, size])</code>	Draw samples from a von Mises distribution.
<code>wald(mean, scale[, size])</code>	Draw samples from a Wald, or inverse Gaussian, distribution.
<code>weibull(a[, size])</code>	Draw samples from a Weibull distribution.
<code>zipf(a[, size])</code>	Draw samples from a Zipf distribution.

SciPy



<https://scipy.org/>

- **SciPy** oferece uma grande variedade de métodos de álgebra linear numérica em seu módulo *scipy.linalg*
- O NumPy tem também algumas funções de álgebra linear, o *numpy.linalg*
- Ambos os pacotes NumPy e SciPy são compatíveis, mas o Scipy tem seu foco em métodos de computação científica e é mais abrangente, enquanto o foco do NumPy está no tipo de dados array e fornece apenas alguns métodos de álgebra linear por conveniência



Apesar de existir desde 2001, um Artigo sobre o SciPy foi recentemente publicado na ***Nature*** (Fevereiro de 2020)

<https://www.nature.com/articles/s41592-019-0686-2>



OPEN
SciPy 1.0: fundamental algorithms for scientific computing in Python

Pauli Virtanen¹, Ralf Gommers^{1,2*}, Travis E. Oliphant^{2,3,4,5,6}, Matt Haberland^{1,7,8*}, Tyler Reddy^{1,9*}, David Cournapeau¹⁰, Evgeni Burovski¹¹, Pearu Peterson^{12,13}, Warren Weckesser¹⁴, Jonathan Bright¹⁵, Stéfan J. van der Walt¹⁴, Matthew Brett¹⁶, Joshua Wilson¹⁷, K. Jarrod Millman^{14,18}, Nikolay Mayorov¹⁹, Andrew R. J. Nelson^{1,20}, Eric Jones⁵, Robert Kern⁵, Eric Larson²¹, C J Carey²², İlhan Polat²³, Yu Feng²⁴, Eric W. Moore²⁵, Jake VanderPlas²⁶, Denis Laxalde^{10,27}, Josef Perktold²⁸, Robert Cimrman²⁹, Ian Henriksen^{6,30,31}, E. A. Quintero³², Charles R. Harris^{33,34}, Anne M. Archibald³⁵, Antônio H. Ribeiro^{10,36}, Fabian Pedregosa³⁷, Paul van Mulbregt^{10,38} and SciPy 1.0 Contributors³⁹

SciPy is an open-source scientific computing library for the Python programming language. Since its initial release in 2001, SciPy has become a de facto standard for leveraging scientific algorithms in Python, with over 600 unique code contributors, thousands of dependent packages, over 100,000 dependent repositories and millions of downloads per year. In this work, we provide an overview of the capabilities and development practices of SciPy 1.0 and highlight some recent technical developments.



Primeiros Passos

- Para utilizar o SciPy em um programa, é necessário importar:

```
import scipy
```

- No caso de uso específico do módulo de álgebra linear:

```
import scipy.linalg as sl
```



Alguns métodos

- Alguns métodos comuns do SciPy incluem:

det	Determinante da Matriz
eig	Autovalores e Autovetores
inv	Matriz Inversa
solve	Solução de um sistema linear geral: $Ax = b$
lstsq	Solução de mínimos quadrados



Exemplo

- Descubra a solução para o sistema:

$$\begin{cases} 3x + 2y = 10 \\ x - y = 2 \end{cases}$$



Exemplo

$$\begin{cases} 3x + 2y = 10 \\ x - y = 2 \end{cases}$$

```
from scipy import linalg

coef = np.array([[3,2],[1,-1]])
b = np.array([[10], [2]])

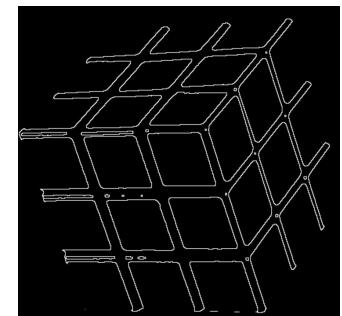
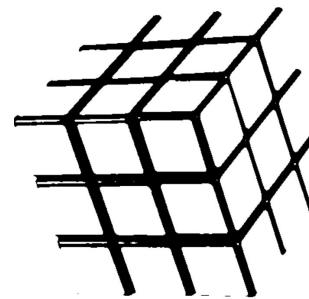
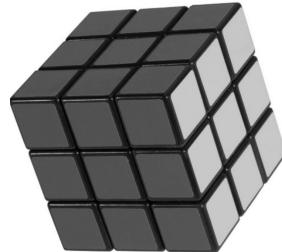
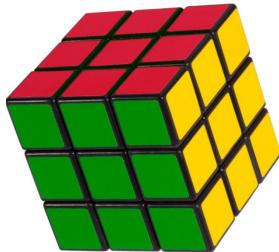
x = linalg.solve(coef, b)
print(x)
```

```
[[2.8]
 [0.8]]
```

Entrega

Exercício

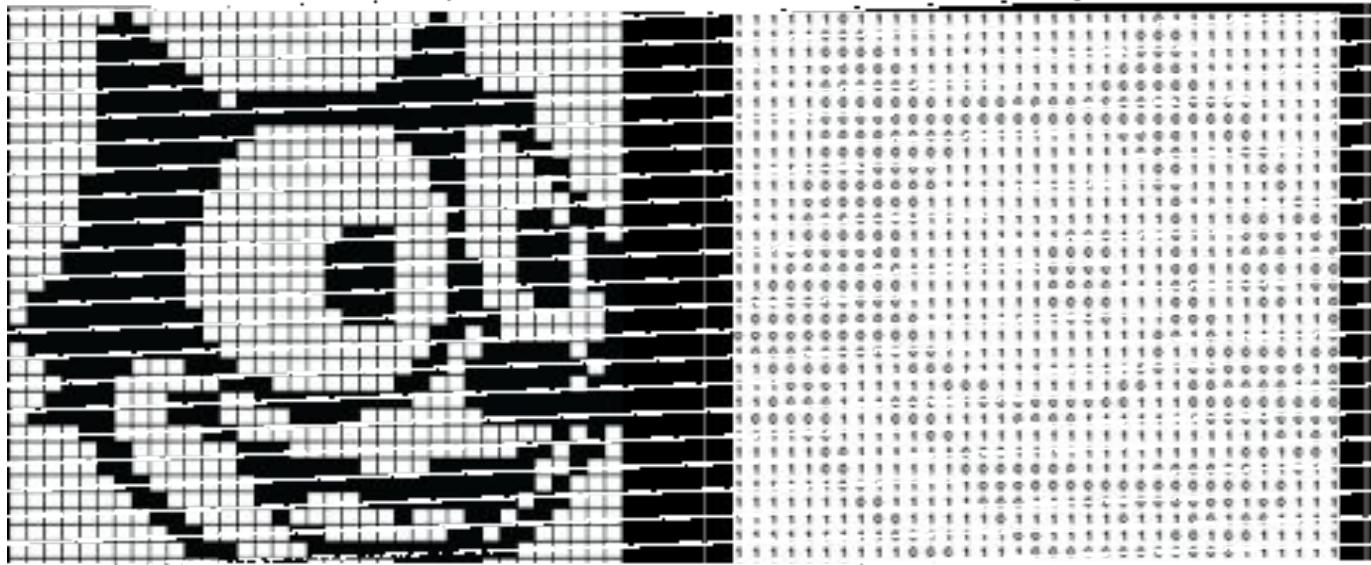
- Dada uma imagem colorida (matriz r, g, b) converter para escala de cinza, converter para binária (preto e branco), utilizar o filtro da média e detectar as bordas utilizando o detector Sobel ou Laplaciano



Imagen???

Representação da imagem

- Uma imagem é composta por, pelo menos, uma matriz de pixels



Representação da imagem

- Uma imagem é composta por, pelo menos, uma matriz de pixels
- A resolução da imagem determina o nível de detalhamento da informação visual: quanto maior a matriz (maior quantidade de pixels), melhor a qualidade da imagem



Representação da imagem

- Por exemplo, uma imagem em escala de cinza:
 - A imagem é representada em uma matriz de inteiros, onde cada célula é um inteiro, sem sinal, de 8 bits que pode conter valores de **0** (preto) até **255** (branco)

Escala de Cinza

Representação da imagem

- **Valor do pixel da imagem colorida em RGB:**
 - Normalmente **3 matrizes** de inteiros, sem sinal: 256 valores
 - A junção das 3 matrizes produz a imagem colorida com capacidade de reprodução de **16,7 milhões de cores**, sendo que os 8 bits tem capacidade para 256 valores e elevando a 3 temos $256^3 = 16,7$ milhões.

Red

Green

Blue

Sistema de coordenadas e manipulação de pixels

- A posição $(0,0)$ está na linha zero e coluna zero.

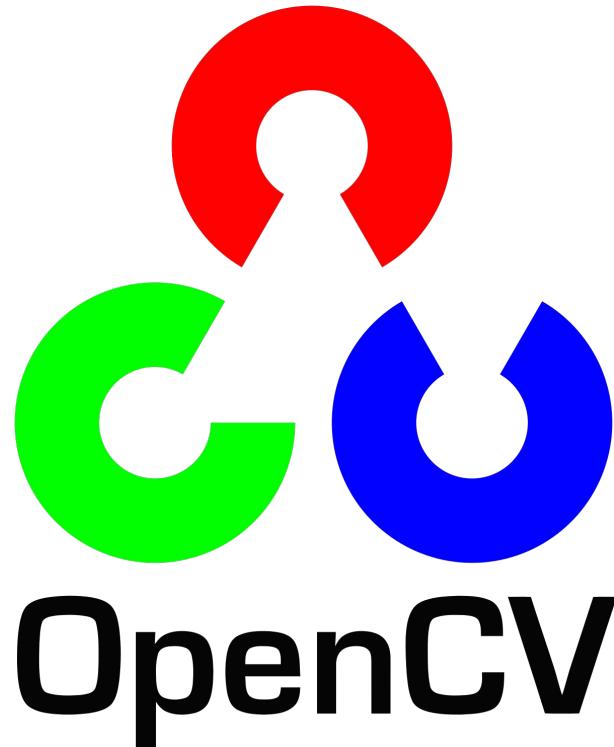
Sistema de coordenadas e manipulação de pixels

- Como manipular pixels e trabalhar com imagens e visão computacional de uma forma eficiente?



Sistema de coordenadas e manipulação de pixels

- Como manipular pixels e trabalhar com imagens e visão computacional de uma forma eficiente?





- O **OpenCV** (*Open Source Computer Vision Library*) é uma biblioteca de software de visão computacional de código aberto, totalmente livre ao uso acadêmico e comercial
- OpenCV representa uma imagem como um **array NumPy** composto por inteiros que representam os pixels e a intensidade

<https://opencv.org/>



- A biblioteca é amplamente utilizada em empresas, grupos de pesquisa e órgãos governamentais
- Originalmente feita para C/C++, mas hoje da suporte também para Python e Java



- Para o Exercício, vamos verificar algumas funções
- Como as imagens são matrizes, muitas operações são feitas como convoluções

Convolução

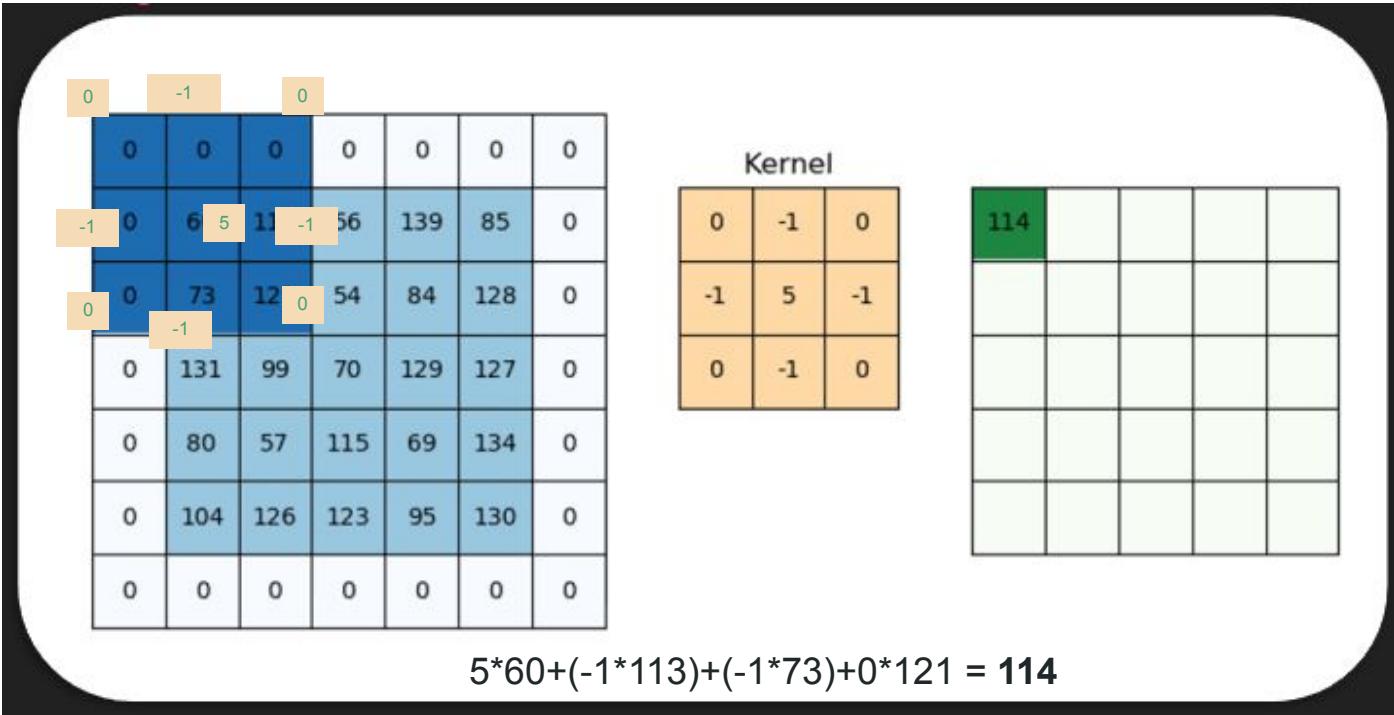
0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

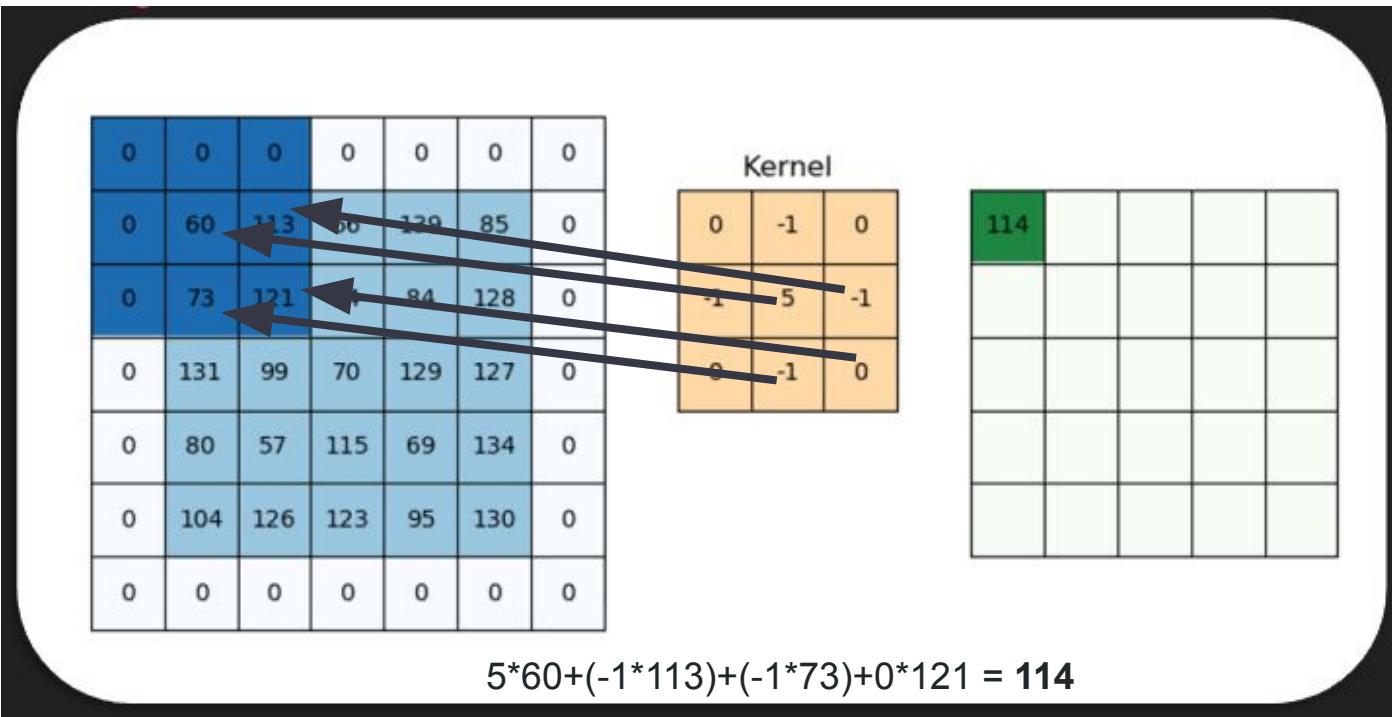
0	-1	0
-1	5	-1
0	-1	0

114			

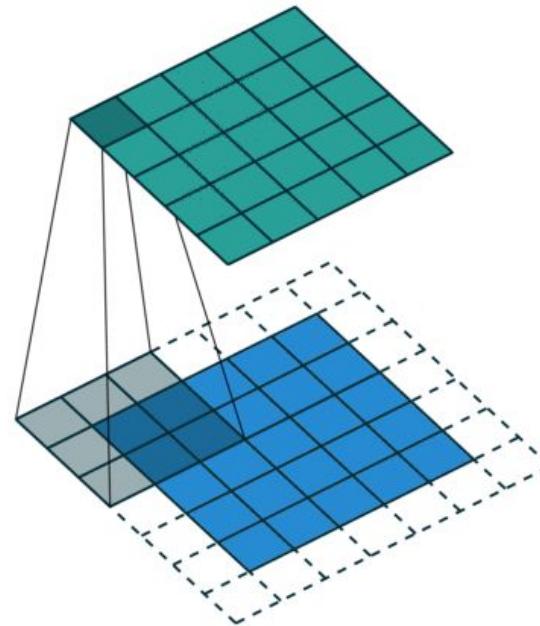
Convolução



Convolução



Convolução



Convolução

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114				

Conversão de Imagem Colorida para Escala de Cinza

$$L = 0.299 * R + 0.587 * G + 0.114 * B$$

- R = red
- G = green
- B = blue

Suavização: Filtro da Média

$1/9$

1	1	1
1	1	1
1	1	1

Kernel – 3x3

Kernel = No filtro da média, a somatória dos elementos deve dar 1

6	4	5	6	8
9	0	4	8	5
3	2	3	4	2
9	2	3	6	1
7	8	9	0	4

Imagem

Detectores de Borda

- **Filtro Sobel**

- De forma simplificada, aplica dois kernels a imagem monocromática:
 - Um kernel para horizontal e outra para vertical

-1	0	+1
-2	0	+2
-1	0	+1

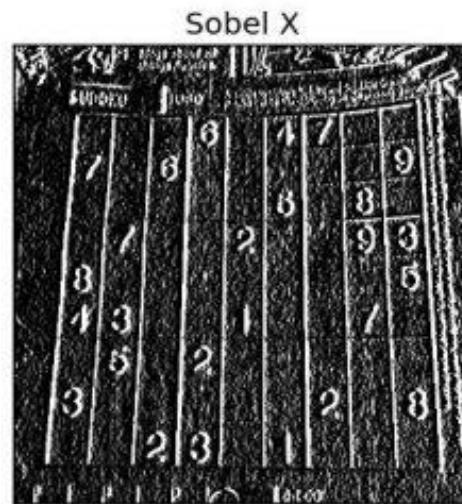
x filter

+1	+2	+1
0	0	0
-1	-2	-1

y filter

Detectores de Borda

- Exemplo de aplicação do filtro Sobel



Detectores de Borda

- **Filtro Laplaciano**

- Aplica a segunda derivada
- Utiliza um kernel na imagem monocromática

Laplaciano simples

0	1	0
1	-4	1
0	1	0

Variante Laplaciano

1	1	1
1	-8	1
1	1	1

Detectores de Borda

- Exemplo de aplicação do filtro Laplaciano

