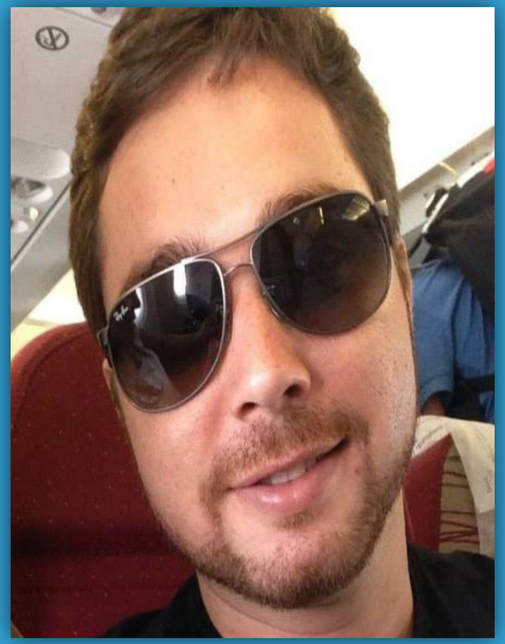
COMPUTAÇÃO CIENTÍFICA

PYTHON



- Graduado em Engenharia de Computação
- Mestre em Engenharia de Computação
- Coordenador
    - Núcleo de Estudos e Pesquisa
    - Laboratório Aberto
    - Graduação em Análise e Desenvolvimento de Sistemas

NumPy

Desmistificando Numpy

INTRODUÇÃO A NUMPY

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Biblioteca Numpy

NumPy *é o pacote fundamental para computação científica em Python. Disponibiliza formas de representar, armazenar e manipular matrizes multidimensionais por meio de operações matemáticas, lógica, álgebra linear, classificação, seleção e operações de E/S.*

NumPy

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Instalação Numpy

❖ Instalação com Conda:

```
# Best practice, use an environment rather than install in the base env
conda create -n my-env
conda activate my-env
# If you want to install from conda-forge
conda config --env --add channels conda-forge
# The actual install command
conda install numpy
```

❖ Instalação com Pip:

```
pip install numpy
```

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

❖ Importando a biblioteca Numpy:

```
import numpy as np
```

```
>>> a = np.arange(6)
>>> a2 = a[np.newaxis, :]
>>> a2.shape
(1, 6)
```

❖ List vs Array Numpy

- Um array é uma estrutura de dados central da biblioteca NumPy.
- O NumPy oferece uma enorme variedade de maneiras rápidas e eficientes de criar matrizes e manipular dados numéricos dentro delas
- Embora uma lista Python possa conter diferentes tipos de dados em uma única lista, todos os elementos em uma matriz NumPy devem ser homogêneos.
- O NumPy usa muito menos memória para armazenar dados e fornece um mecanismo para especificar os tipos de dados

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

❖ **O que é um array:**

- Um array é uma grid de valores no formato de dados brutos de mesmo tipo (homogêneos) com um mecanismos de indexação diversificado para localização de elementos.

- Os atributos de um array refletem informações intrínsecas à própria matriz

- ndarray.ndim
- ndarray.shape
- ndarray.size
- ndarray.dtype
- ndarray.itemsize
- ndarray.data

```
>>> import numpy as np
>>> a = np.array([2,3,4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int64')
>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
```

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
>>> a.itemsize
8
>>> a.size
15
>>> type(a)
<class 'numpy.ndarray'>
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<class 'numpy.ndarray'>
```

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

❖ Como criar um array básico:

Para criar uma matriz NumPy, você pode usar a função np.array ().

```
>>> import numpy as np
>>> a = np.array([1, 2, 3])
```

❖ Array com elementos 0:

```
>>> np.zeros(2)
array([0., 0.])
```

❖ Array com elementos 1:

```
>>> np.ones(2)
array([1., 1.])
```

❖ Especificando tipo de dado:

```
>>> x = np.ones(2, dtype=np.int64)
>>> x
array([1, 1])
```

❖ Array com elementos vazios:

```
>>> # Create an empty array with 2 elements
>>> np.empty(2)
array([ 3.14, 42.  ])  # may vary
```

❖ Array com valores dentro de um intervalo:

```
>>> np.arange(4)
array([0, 1, 2, 3])
```

```
>>> np.arange(2, 9, 2)
array([2, 4, 6, 8])
```

❖ Array com valores linearmente espaçados:

```
>>> np.linspace(0, 10, num=5)
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

❖ Ordenando elementos:

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

```
>>> np.sort(arr)
array([1, 2, 3, 4, 5, 6, 7, 8])
```

❖ Concatenando elementos:

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

```
>>> np.concatenate((a, b))
array([1, 2, 3, 4, 5, 6, 7, 8])
```

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

```
>>> np.concatenate((x, y), axis=0)
array([[1, 2],
       [3, 4],
       [5, 6]])
```

❖ Conhecendo a forma e tamanho:

```
>>> array_example = np.array([[[0, 1, 2, 3],
...                            [4, 5, 6, 7]],
...
...                           [[0, 1, 2, 3],
...                            [4, 5, 6, 7]],
...
...                           [[0 ,1 ,2, 3],
...                            [4, 5, 6, 7]]])
```

```
>>> array_example.ndim
3
```

```
>>> array_example.size
24
```

```
>>> array_example.shape
(3, 2, 4)
```

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

❖ Remodelando um array:

```
>>> a = np.arange(6)
>>> print(a)
[0 1 2 3 4 5]
```

```
>>> b = a.reshape(3, 2)
>>> print(b)
[[0 1]
 [2 3]
 [4 5]]
```

```
>>> numpy.reshape(a, newshape=(1, 6), order='C')
array([[0, 1, 2, 3, 4, 5]])
```

❖ Como converte array 1D dentro de 2D:

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

```
>>> a2 = a[np.newaxis, :]
>>> a2.shape
(1, 6)
```

```
>>> row_vector = a[np.newaxis, :]
>>> row_vector.shape
(1, 6)
```

```
>>> col_vector = a[:, np.newaxis]
>>> col_vector.shape
(6, 1)
```

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

```
>>> b = np.expand_dims(a, axis=1)
>>> b.shape
(6, 1)
```

```
>>> c = np.expand_dims(a, axis=0)
>>> c.shape
(1, 6)
```

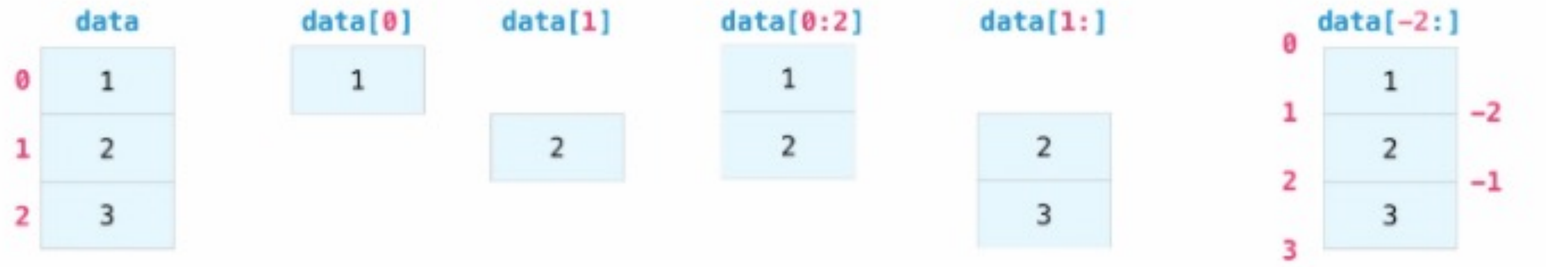
INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

- Indexando e fatiando arrays

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

>>> data[1]
2
>>> data[0:2]
array([1, 2])
>>> data[1:]
array([2, 3])
>>> data[-2:]
array([2, 3])
```



INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

❖ Indexando e fatiando arrays por expressões:

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

```
>>> print(a[a < 5])
[1 2 3 4]
```

```
>>> five_up = (a >= 5)
>>> print(a[five_up])
[ 5  6  7  8  9 10 11 12]
```

```
>>> divisible_by_2 = a[a%2==0]
>>> print(divisible_by_2)
[ 2  4  6  8 10 12]
```

```
>>> c = a[(a > 2) & (a < 11)]
>>> print(c)
[ 3  4  5  6  7  8  9 10]
```

```
>>> five_up = (a > 5) | (a == 5)
>>> print(five_up)
[[False False False False]
 [ True  True  True  True]
 [ True  True  True True]]
```

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

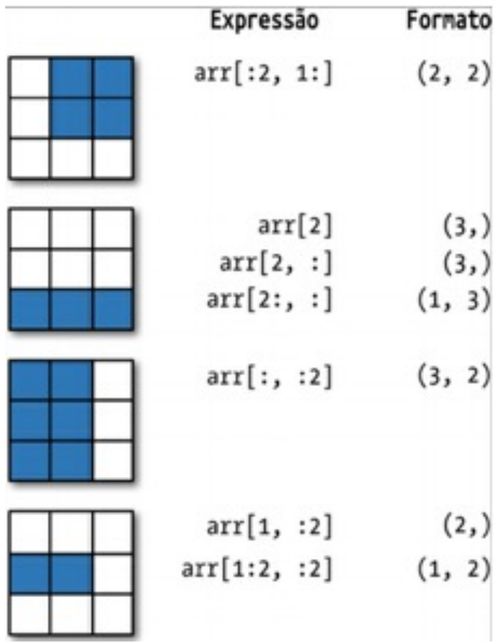
Comandos Numpy

❖ Indexando limites aos arrays:

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

```
>>> arr1 = a[3:8]
>>> arr1
array([4, 5, 6, 7, 8])
```

```
>>> b2 = a.copy()
```



Expressão	Formato
arr[:2, 1:]	(2, 2)
arr[2]	(3,)
arr[2, :]	(3,)
arr[2:, :]	(1, 3)
arr[:, :2]	(3, 2)
arr[1, :2]	(2,)
arr[1:2, :2]	(1, 2)

❖ Empilhando arrays:

```
>>> a1 = np.array([[1, 1],
...                [2, 2]])

>>> a2 = np.array([[3, 3],
...                [4, 4]])
```

```
>>> np.vstack((a1, a2))
array([[1, 1],
       [2, 2],
       [3, 3],
       [4, 4]])
```

```
>>> np.hstack((a1, a2))
array([[1, 1, 3, 3],
       [2, 2, 4, 4]])
```

❖ Split arrays:

```
>>> x = np.arange(1, 25).reshape(2, 12)
>>> x
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]])
```

```
>>> np.hsplit(x, 3)
[array([[1,  2,  3,  4],
        [13, 14, 15, 16]]), array([[ 5,  6,  7,  8],
        [17, 18, 19, 20]]), array([[ 9, 10, 11, 12],
        [21, 22, 23, 24]])]
```

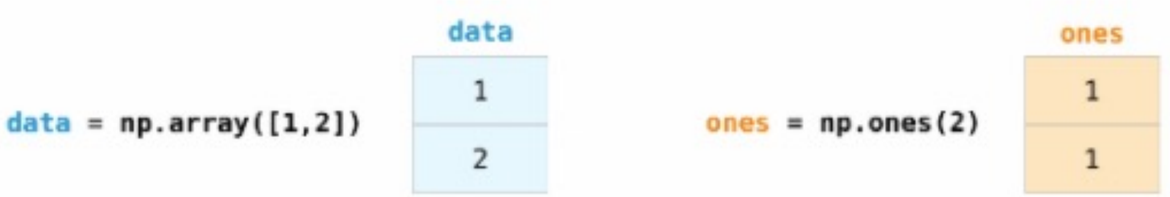
```
>>> np.hsplit(x, (3, 4))
[array([[1, 2, 3],
        [13, 14, 15]]), array([[ 4],
        [16]]), array([[ 5, 6, 7, 8, 9, 10, 11, 12],
        [17, 18, 19, 20, 21, 22, 23, 24]])]
```

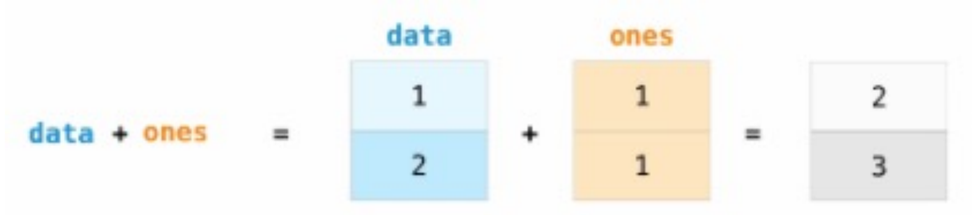
INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

❖ Operações com array:





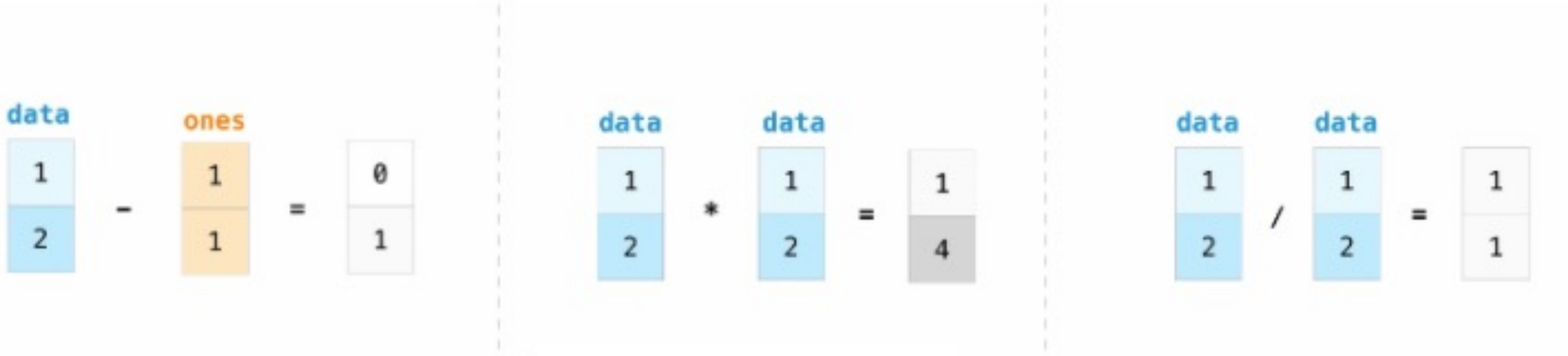

```
>>> data = np.array([1, 2])
>>> ones = np.ones(2, dtype=int)
>>> data + ones
array([2, 3])
```

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

❖ Operações básicas com array:




```
>>> data - ones
array([0, 1])
>>> data * data
array([1, 4])
>>> data / data
array([1., 1.])
```

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

>>> a.sum()
10
```

```
>>> b = np.array([[1, 1], [2, 2]])
```

```
>>> b.sum(axis=0)
array([3, 3])
```

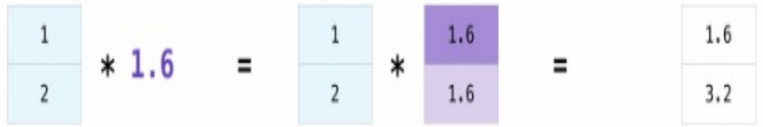
```
>>> b.sum(axis=1)
array([2, 4])
```

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

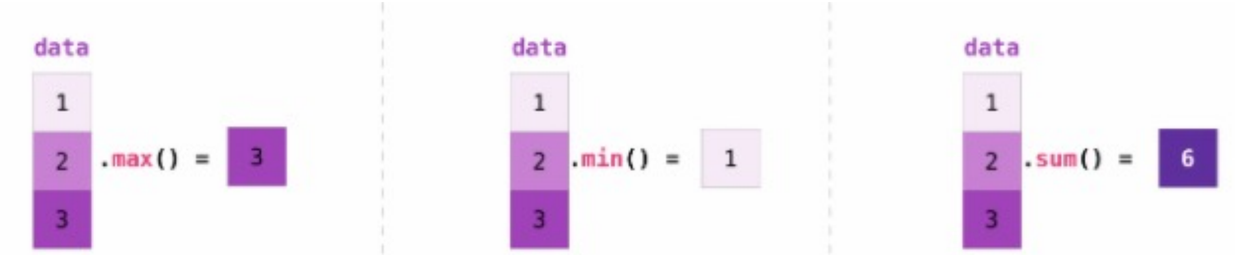
Comandos Numpy

❖ Broadcasting com array:




```
>>> data = np.array([1.0, 2.0])
>>> data * 1.6
array([1.6, 3.2])
```

❖ Mínimos e máximos com array:




```
>>> data.max()
2.0
>>> data.min()
1.0
>>> data.sum()
3.0
```

```
>>> a = np.array([[0.45053314, 0.17296777, 0.34376245, 0.5510652],
...               [0.54627315, 0.05093587, 0.40067661, 0.55645993],
...               [0.12697628, 0.82485143, 0.26590556, 0.56917101]])
```

```
>>> a.sum()
4.8595784
```

```
>>> a.min()
0.05093587
```

```
>>> a.min(axis=0)
array([0.12697628, 0.05093587, 0.26590556, 0.5510652 ])
```

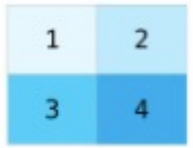
INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

❖ Criando matrizes:

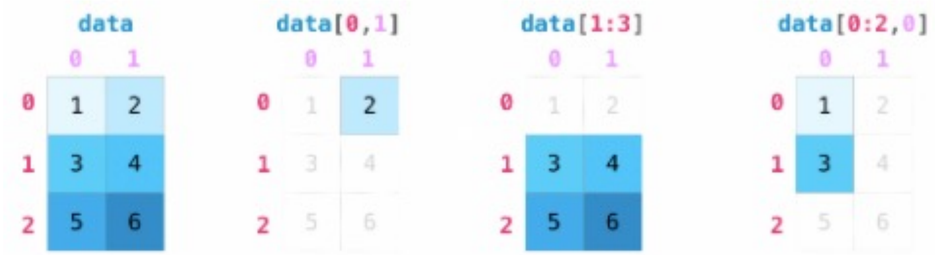
```
np.array([[1,2],[3,4]])
```



1	2
3	4

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

❖ Indexando matrizes:




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

INTRODUÇÃO AO NUMPY

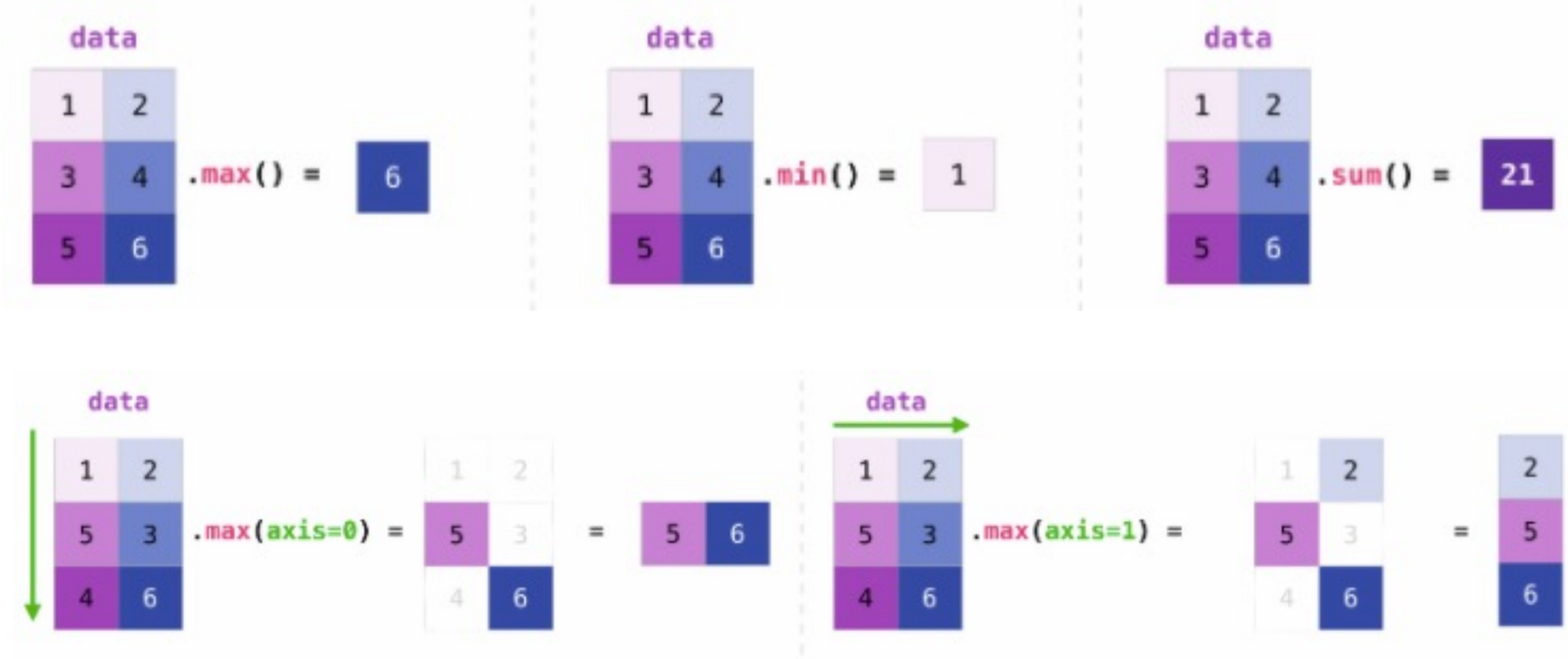
DESMISTIFICANDO NUMPY

Comandos Numpy

❖ Mínimos e máximo com matrizes:

```
>>> data.max()
4
>>> data.min()
1
>>> data.sum()
10
```

```
>>> data.max(axis=0)
array([3, 4])
>>> data.max(axis=1)
array([2, 4])
```



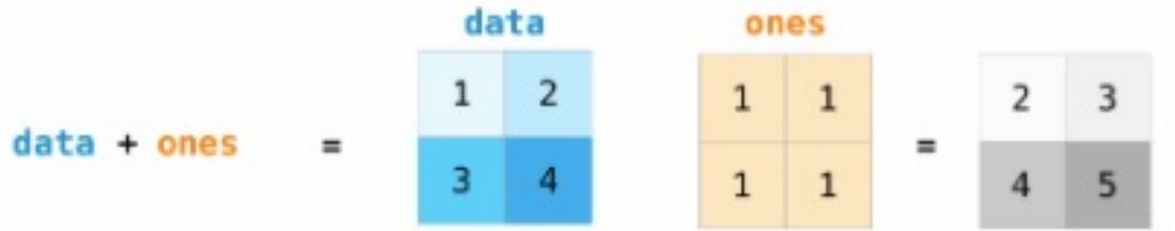
INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

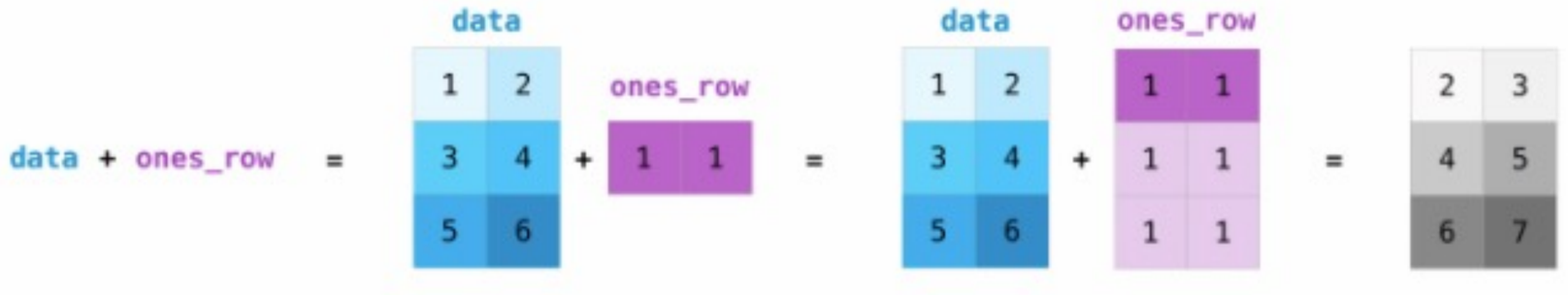
Comandos Numpy

- Operações básicas com matrizes:

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




```
>>> 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]])
```



INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

- Geração de matrizes:

```
>>> np.ones((4, 3, 2))
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.]]])
```


```
>>> np.ones(3)
array([1., 1., 1.])
>>> np.zeros(3)
array([0., 0., 0.])
# the simplest way to generate random numbers
>>> rng = np.random.default_rng(0)
>>> rng.random(3)
array([0.63696169, 0.26978671, 0.04097352])
```

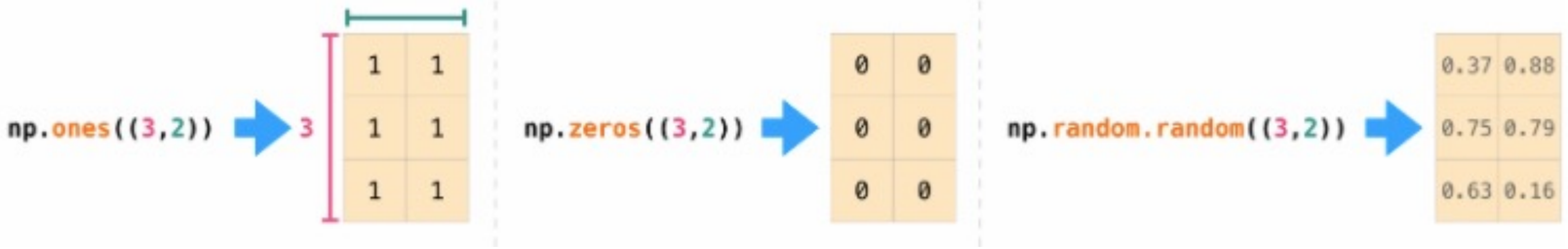
INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

- Geração de matrizes:

```
>>> np.ones((3, 2))
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
>>> np.zeros((3, 2))
array([[0., 0.],
       [0., 0.],
       [0., 0.]])
>>> rng.random((3, 2))
array([[0.01652764, 0.81327024],
       [0.91275558, 0.60663578],
       [0.72949656, 0.54362499]])  # may vary
```




```
>>> rng.integers(5, size=(2, 4))
array([[2, 1, 1, 0],
       [0, 0, 0, 4]])  # may vary
```

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

- Como obter itens únicos e contar:

```
>>> a = np.array([11, 11, 12, 13, 14, 15, 16, 17, 12, 13, 11, 14, 18, 19, 20])
```

```
>>> unique_values = np.unique(a)
>>> print(unique_values)
[11 12 13 14 15 16 17 18 19 20]
```

```
>>> unique_values, indices_list = np.unique(a, return_index=True)
>>> print(indices_list)
[ 0  2  3  4  5  6  7 12 13 14]
```

```
>>> unique_values, occurrence_count = np.unique(a, return_counts=True)
>>> print(occurrence_count)
[3 2 2 2 1 1 1 1 1 1]
```

```
>>> a_2d = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [1, 2, 3, 4]])
```

```
>>> unique_values = np.unique(a_2d)
>>> print(unique_values)
[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

```
>>> unique_rows = np.unique(a_2d, axis=0)
>>> print(unique_rows)
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

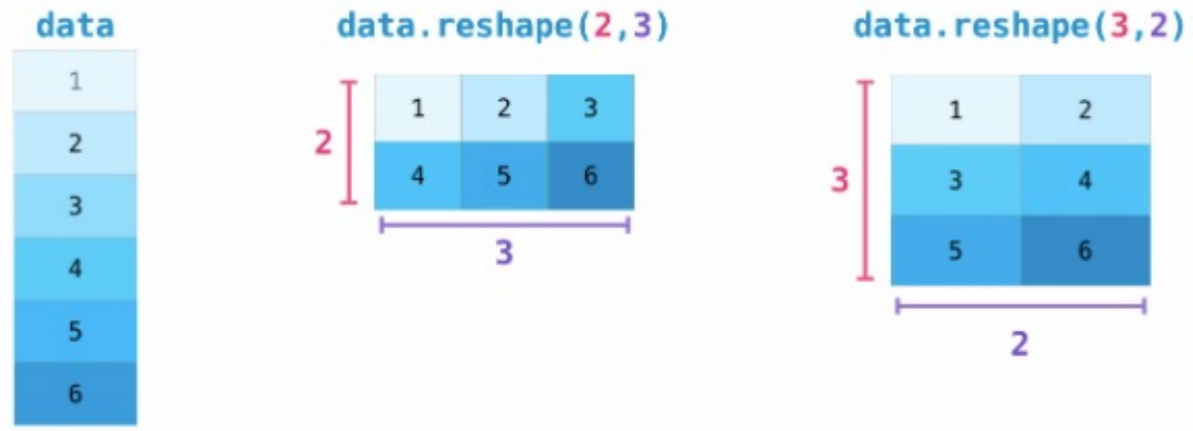
```
>>> unique_rows, indices, occurrence_count = np.unique(
...      a_2d, axis=0, return_counts=True, return_index=True)
>>> print(unique_rows)
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
>>> print(indices)
[0 1 2]
>>> print(occurrence_count)
[2 1 1]
```

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

❖ Transpondo e remodelando uma matriz:




```
>>> data.reshape(2, 3)
array([[1, 2, 3],
       [4, 5, 6]])
>>> data.reshape(3, 2)
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
>>> arr = np.arange(6).reshape((2, 3))
>>> arr
array([[0, 1, 2],
       [3, 4, 5]])
```

```
>>> arr.transpose()
array([[0, 3],
       [1, 4],
       [2, 5]])
```

```
>>> arr.T
array([[0, 3],
       [1, 4],
       [2, 5]])
```

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

❖ Remodelagem e achatamento de vetores multidimensionais:

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

```
>>> x.flatten()
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

$$MeanSquareError = \frac{1}{n}\sum_{i=1}^{n}(Y\_prediction_i - Y_i)^2$$

```
>>> a1 = x.flatten()
>>> a1[0] = 99
>>> print(x)  # Original array
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
>>> print(a1)  # New array
[99  2  3  4  5  6  7  8  9 10 11 12]
```

```
>>> a2 = x.ravel()
>>> a2[0] = 98
>>> print(x)  # Original array
[[98  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
>>> print(a2)  # New array
[98  2  3  4  5  6  7  8  9 10 11 12]
```

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

- Como salvar ou carregar objetos Numpy:

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

```
>>> np.save('filename', a)
```

```
>>> b = np.load('filename.npy')
```

```
>>> csv_arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
>>> np.savetxt('new_file.csv', csv_arr)
```

```
>>> np.loadtxt('new_file.csv')
array([1., 2., 3., 4., 5., 6., 7., 8.])
```

INTRODUÇÃO AO NUMPY

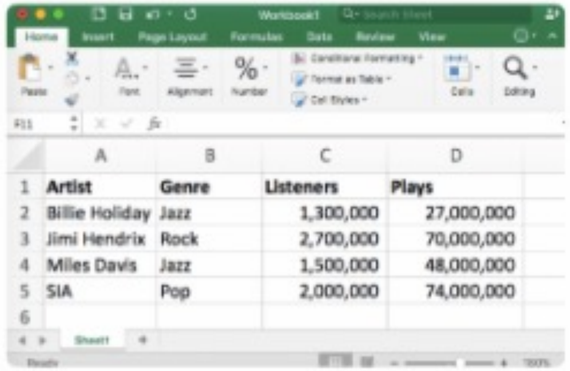
DESMISTIFICANDO NUMPY

Comandos Numpy

❖ Importando e exportando um CSV:

```
>>> import pandas as pd

>>> # If all of your columns are the same type:
>>> x = pd.read_csv('music.csv', header=0).values
>>> print(x)
[['Billie Holiday' 'Jazz' 1300000 27000000]
 ['Jimmie Hendrix' 'Rock' 2700000 70000000]
 ['Miles Davis' 'Jazz' 1500000 48000000]
 ['SIA' 'Pop' 2000000 74000000]]

>>> # You can also simply select the columns you need:
>>> x = pd.read_csv('music.csv', usecols=['Artist', 'Plays']).values
>>> print(x)
[['Billie Holiday' 27000000]
 ['Jimmie Hendrix' 70000000]
 ['Miles Davis' 48000000]
 ['SIA' 74000000]]
```

music.csv



	A	B	C	D
1	Artist	Genre	Listeners	Plays
2	Billie Holiday	Jazz	1,300,000	27,000,000
3	Jimi Hendrix	Rock	2,700,000	70,000,000
4	Miles Davis	Jazz	1,500,000	48,000,000
5	SIA	Pop	2,000,000	74,000,000
6				

pandas.read_csv('music.csv')

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

❖ Funções universais:

Tabela 4.3 – Ufuncs unárias

Função	Descrição
`abs, fabs`	Calcula o valor absoluto de inteiros, números de ponto flutuante e valores complexos para todos os elementos
`sqrt`	Calcula a raiz quadrada de cada elemento (equivalente a `arr ** 0.5`)
`square`	Calcula o quadrado de cada elemento (equivalente a `arr ** 2`)
`exp`	Calcula o exponencial ex de cada elemento
`log, log10, log2, log1p`	Logaritmo natural (base *e*), log na base 10, log na base 2 e log (1 + x), respectivamente
`sign`	Calcula o sinal de cada elemento: 1 (positivo), 0 (zero) ou –1 (negativo)
`ceil`	Calcula o teto de cada elemento (isto é, o menor inteiro maior ou igual ao número)
`floor`	Calcula o piso de cada elemento (isto é, o maior inteiro menor ou igual ao elemento)
`rint`	Arredonda os elementos para o inteiro mais próximo, preservando o `dtype`
`modf`	Devolve as partes fracionária e inteira do array como um array separado
`isnan`	Devolve um array booleano indicando se cada valor é `NaN` (Not a Number)
`isfinite, isinf`	Devolve um array booleano indicando se cada elemento é finito (não `inf`, não `NaN`) ou infinito, respectivamente
`cos, cosh, sin, sinh, tan, tanh`	Funções trigonométricas regulares e hiperbólicas
`arccos, arccosh, arcsin, arcsinh, arctan, arctanh`	Funções trigonométricas inversas
`logical_not`	Calcula o valor-verdade de `not x` para todos os elementos (equivalente a `~arr`).

Tabela 4.4 – Funções universais binárias

Função	Descrição
`add`	Soma elementos correspondentes em arrays
`subtract`	Subtrai elementos do segundo array do primeiro
`multiply`	Multiplica elementos do array
`divide, floor_divide`	Faz a divisão ou a divisão pelo piso (truncando o resto)
`power`	Eleva os elementos do primeiro array às potências indicadas no segundo array
`maximum, fmax`	Máximo para todos os elementos; `fmax` ignora `NaN`
`minimum, fmin`	Mínimo para todos os elementos; `fmin` ignora `NaN`
`mod`	Módulo para todos os elementos (resto da divisão)
`copysign`	Copia o sinal dos valores do segundo argumento para os valores do primeiro argumento
`greater, greater_equal, less, less_equal, equal, not_equal`	Faz uma comparação para todos os elementos, produzindo um array booleano
	(equivalente aos operadores infixos `>, >=, <, <=, ==, !=`)
`logical_and, logical_or, logical_xor`	Calcula o valor-verdade da operação lógica (equivalente aos operadores infixos `&, |, ^`) para todos os elementos

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

- Funções universais:

Tabela 4.5 – Métodos estatísticos básicos de arrays

Método	Descrição
`sum`	Soma de todos os elementos do array ou ao longo de um eixo; arrays de tamanho zero têm soma igual a 0
`mean`	Média aritmética; arrays de tamanho zero têm média `NaN`
`std, var`	Desvio-padrão e variância, respectivamente, com graus opcionais de ajuste de liberdade (denominador default `n`)
`min, max`	Mínimo e máximo
`argmin, argmax`	Índices dos elementos mínimo e máximo, respectivamente
`cumsum`	Soma cumulativa dos elementos, começando de 0
`cumprod`	Produto cumulativo dos elementos, começando de 1

Tabela 4.6 – Operações de conjunto em arrays

Método	Descrição
`unique(x)`	Calcula os elementos únicos ordenados de `x`
`intersect1d(x, y)`	Calcula os elementos comuns ordenados em `x` e `y`
`union1d(x, y)`	Calcula a união ordenada dos elementos
`in1d(x, y)`	Calcula um array booleano indicando se cada elemento de `x` está contido em `y`
`setdiff1d(x, y)`	Diferença entre conjuntos, isto é, elementos em `x` que não estão em `y`
`setxor1d(x, y)`	Diferença simétrica entre conjuntos: elementos que estão em apenas um dos arrays, mas não em ambos

Tabela 4.7 – Funções comumente usadas de numpy.linalg

Função	Descrição
`diag`	Devolve os elementos diagonais (ou fora da diagonal) de uma matriz quadrada como um array 1D, ou converte um array 1D em uma matriz quadrada, com zeros fora da diagonal
`dot`	Multiplicação de matrizes
`trace`	Calcula a soma dos elementos da diagonal
`det`	Calcula o determinante da matriz
`eig`	Calcula os autovalores (valores próprios) e os autovetores de uma matriz quadrada
`inv`	Calcula a inversa de uma matriz quadrada
`pinv`	Calcula a pseudoinversa de Moore-Penrose de uma matriz
`qr`	Calcula a decomposição QR
`svd`	Calcula a SVD (Singular Value Decomposition, ou Decomposição de Valor Singular)
`solve`	Resolve o sistema linear Ax = b para x, em que A é uma matriz quadrada
`lstsq`	Calcula a solução de quadrados mínimos para Ax = b

INTRODUÇÃO AO NUMPY

DESMISTIFICANDO NUMPY

Comandos Numpy

- Funções universais:

Tabela 4.8 – Lista parcial de funções de numpy.random

Função	Descrição
`seed`	Fornece uma semente ao gerador de números aleatórios
`permutation`	Devolve uma permutação aleatória de uma sequência ou um intervalo permutado
`shuffle`	Permuta aleatoriamente uma sequência in-place
`rand`	Sorteia amostras de uma distribuição uniforme
`randint`	Sorteia inteiros aleatórios de um dado intervalo de valores menores para maiores
`randn`	Sorteia amostras de uma distribuição normal com média 0 e desvio-padrão 1 (interface do tipo MATLAB)
`binomial`	Sorteia amostras de uma distribuição binomial
`normal`	Sorteia amostras de uma distribuição normal (gaussiana)
`beta`	Sorteia amostras de uma distribuição beta
`chisquare`	Sorteia amostras de uma distribuição qui-quadrada
`gamma`	Sorteia amostras de uma distribuição gama
`uniform`	Sorteia amostras de uma distribuição uniforme [0, 1)

Desmistificando MATPLOTLIB

INTRODUÇÃO A MATPLOTLIB

INTRODUÇÃO AO MATPLOTLIB

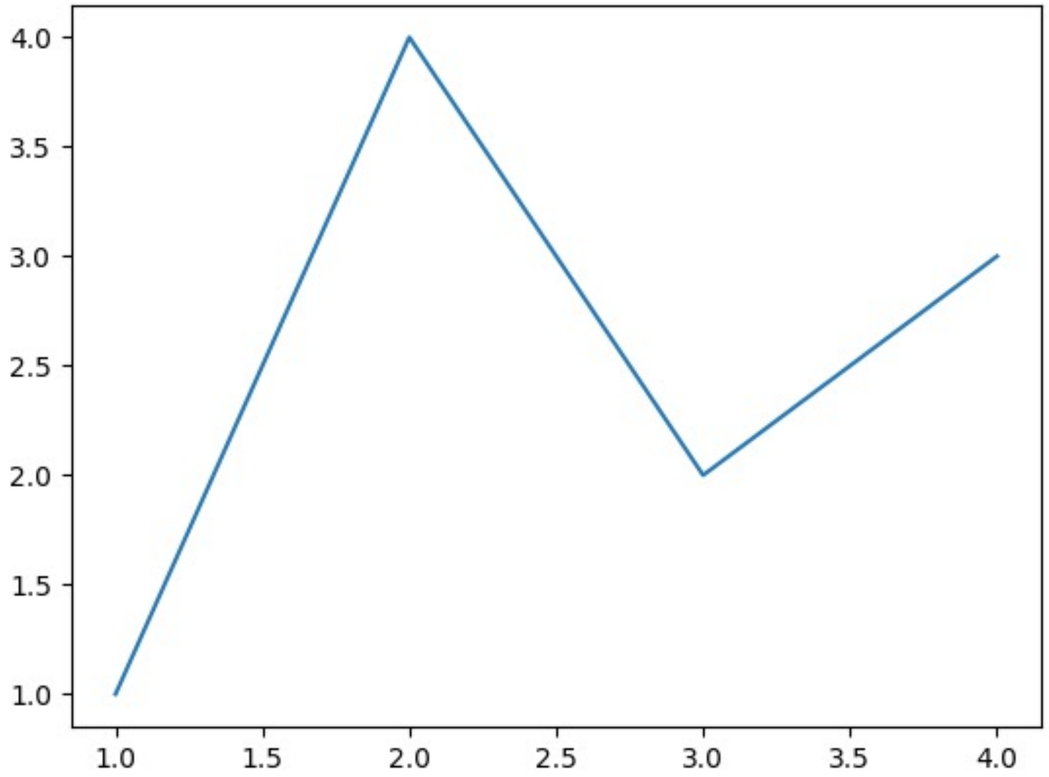
DESMISTIFICANDO MATPLOTLIB

Comandos MATPLOTLIB

❖ Um exemplo do MATPLOTLIB:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
plt.plot([1, 2, 3, 4], [1, 4, 2, 3])  # Matplotlib plot.
```



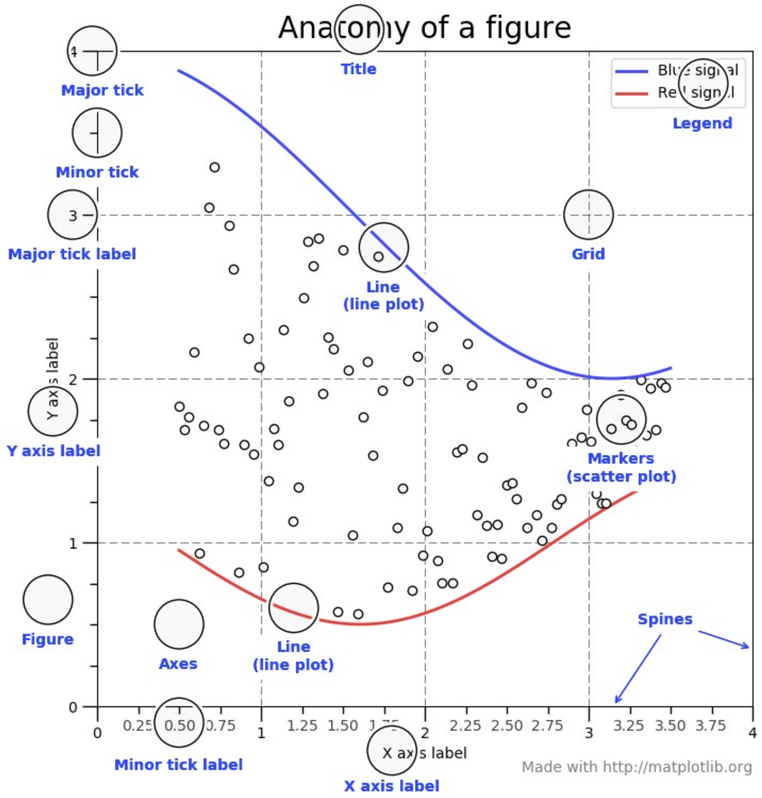

```
fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])  # Plot some data on the axes.
```

INTRODUÇÃO AO MATPLOTLIB

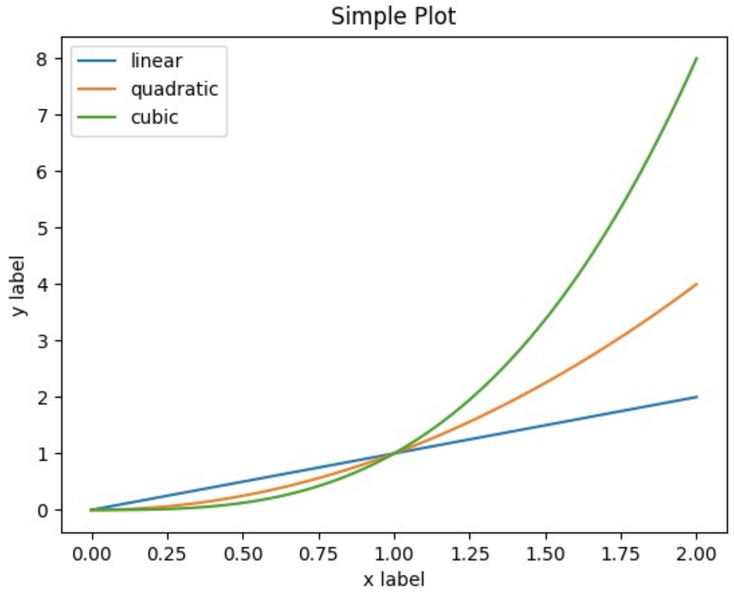
DESMISTIFICANDO MATPLOTLIB

Comandos MATPLOTLIB

- Partes de um Figure:




```python
x = np.linspace(0, 2, 100)

# Note that even in the OO-style, we use `.pyplot.figure` to create the figure.
fig, ax = plt.subplots()  # Create a figure and an axes.
ax.plot(x, x, label='linear')  # Plot some data on the axes.
ax.plot(x, x**2, label='quadratic')  # Plot more data on the axes...
ax.plot(x, x**3, label='cubic')  # ... and some more.
ax.set_xlabel('x label')  # Add an x-label to the axes.
ax.set_ylabel('y label')  # Add a y-label to the axes.
ax.set_title("Simple Plot")  # Add a title to the axes.
ax.legend()  # Add a legend.
```



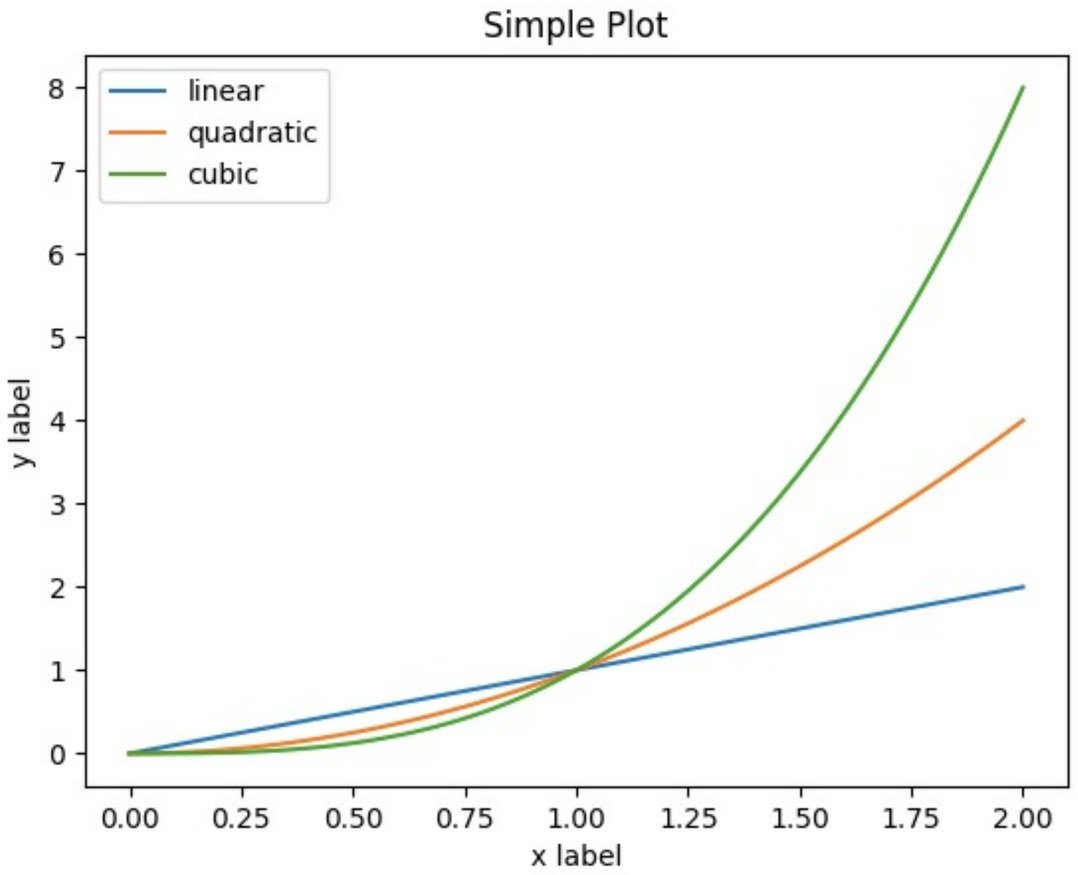
INTRODUÇÃO AO MATPLOTLIB

DESMISTIFICANDO MATPLOTLIB

Comandos MATPLOTLIB

- Partes de um Figure:

```
x = np.linspace(0, 2, 100)

plt.plot(x, x, label='linear')  # Plot some data on the (implicit) axes.
plt.plot(x, x**2, label='quadratic')  # etc.
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
```



INTRODUÇÃO AO MATPLOTLIB

DESMISTIFICANDO MATPLOTLIB

Comandos MATPLOTLIB

- Partes de um Figure:

```
data1, data2, data3, data4 = np.random.randn(4, 100)
fig, ax = plt.subplots(1, 1)
my_plotter(ax, data1, data2, {'marker': 'x'})
```

```
fig, (ax1, ax2) = plt.subplots(1, 2)
my_plotter(ax1, data1, data2, {'marker': 'x'})
my_plotter(ax2, data3, data4, {'marker': 'o'})
```

```
def my_plotter(ax, data1, data2, param_dict):
    """
    A helper function to make a graph

    Parameters
    ----------
    ax : Axes
        The axes to draw to

    data1 : array
       The x data

    data2 : array
       The y data

    param_dict : dict
       Dictionary of kwargs to pass to ax.plot

    Returns
    -------
    out : list
        list of artists added
    """
    out = ax.plot(data1, data2, **param_dict)
    return out
```

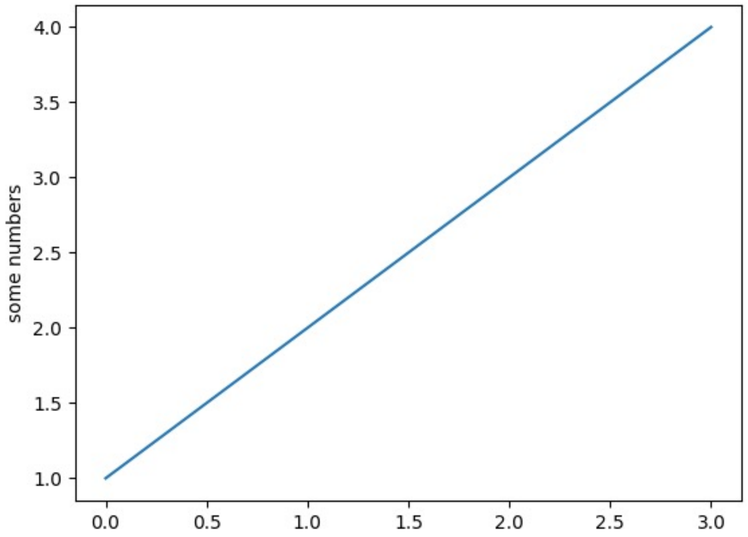
INTRODUÇÃO AO MATPLOTLIB

DESMISTIFICANDO MATPLOTLIB

Comandos MATPLOTLIB

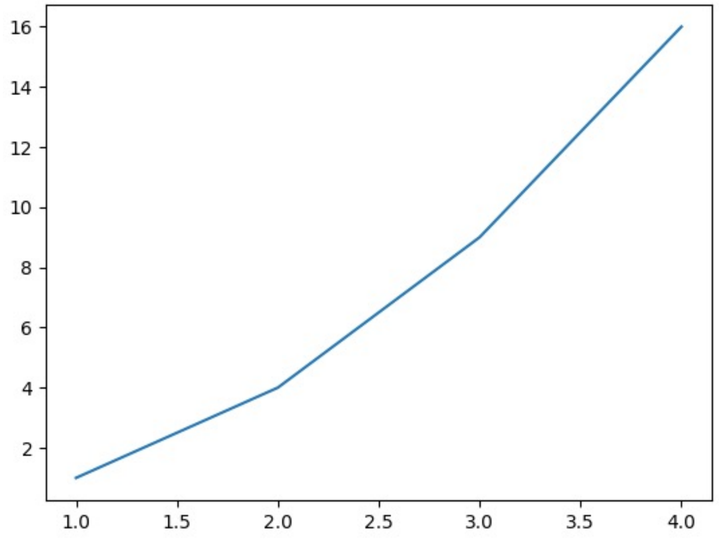
- Com um único vetor:

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```



- Com um dois vetores:

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
```



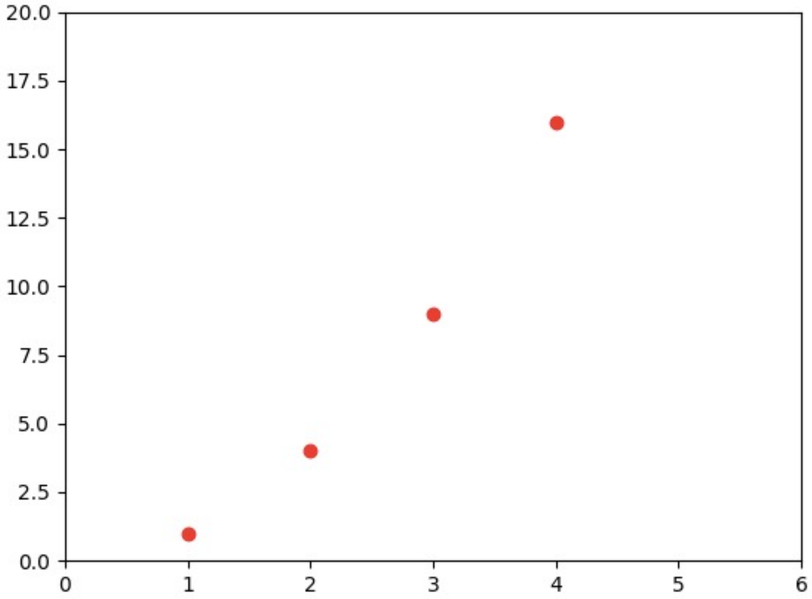
INTRODUÇÃO AO MATPLOTLIB

DESMISTIFICANDO MATPLOTLIB

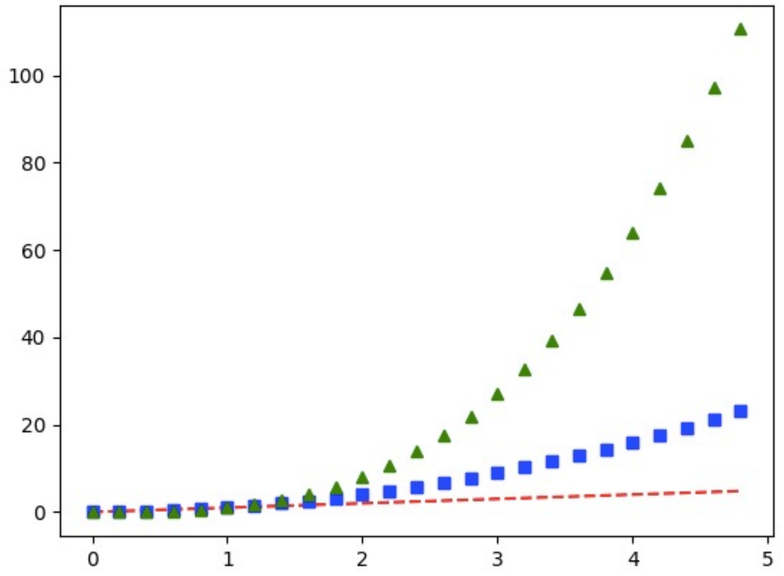
Comandos MATPLOTLIB

❖ Formatando o estilo do seu plot:

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```




```
import numpy as np

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



❖ Propriedades da linha:

```
plt.plot(x, y, linewidth=2.0)
```

```
line, = plt.plot(x, y, '-')
line.set_antialiased(False) # turn off antialiasing
```

```
lines = plt.plot(x1, y1, x2, y2)
# use keyword args
plt.setp(lines, color='r', linewidth=2.0)
# or MATLAB style string value pairs
plt.setp(lines, 'color', 'r', 'linewidth', 2.0)
```

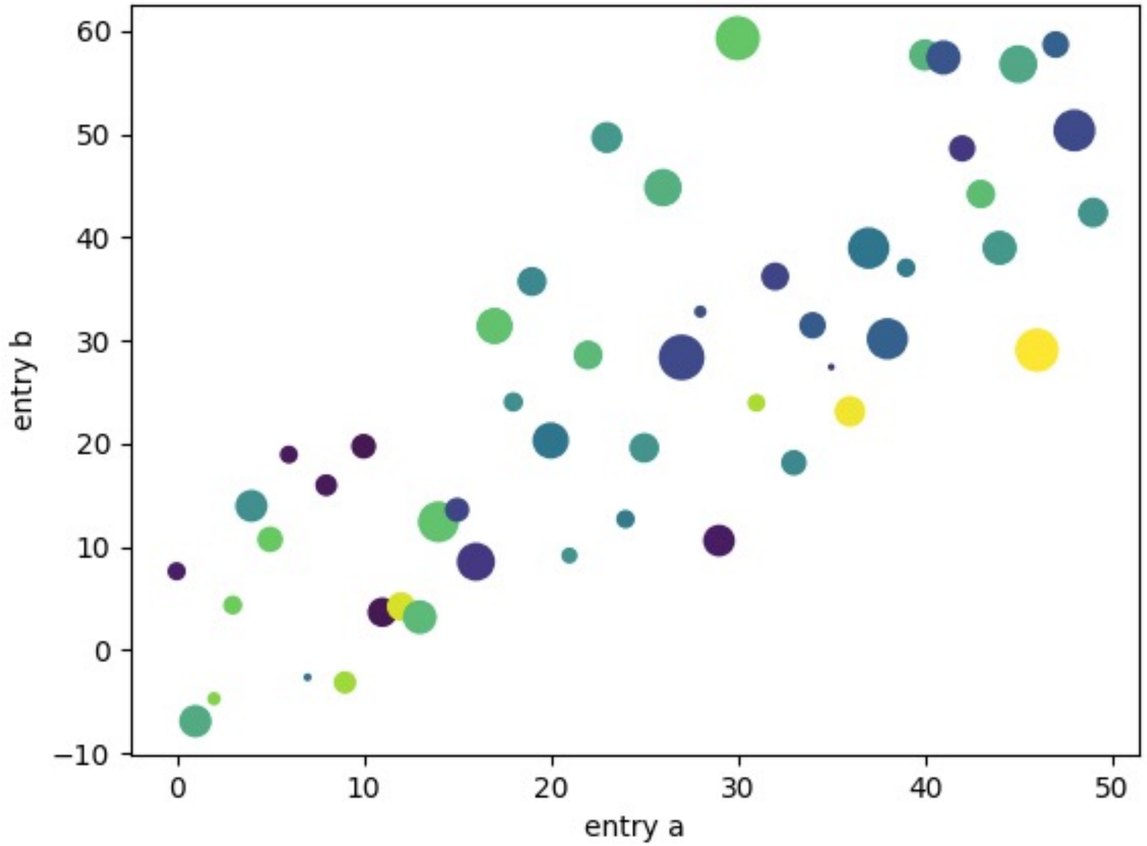
INTRODUÇÃO AO MATPLOTLIB

DESMISTIFICANDO MATPLOTLIB

Comandos MATPLOTLIB

- Plotando com strings de palavras-chave:

```
data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100

plt.scatter('a', 'b', c='c', s='d', data=data)
plt.xlabel('entry a')
plt.ylabel('entry b')
plt.show()
```



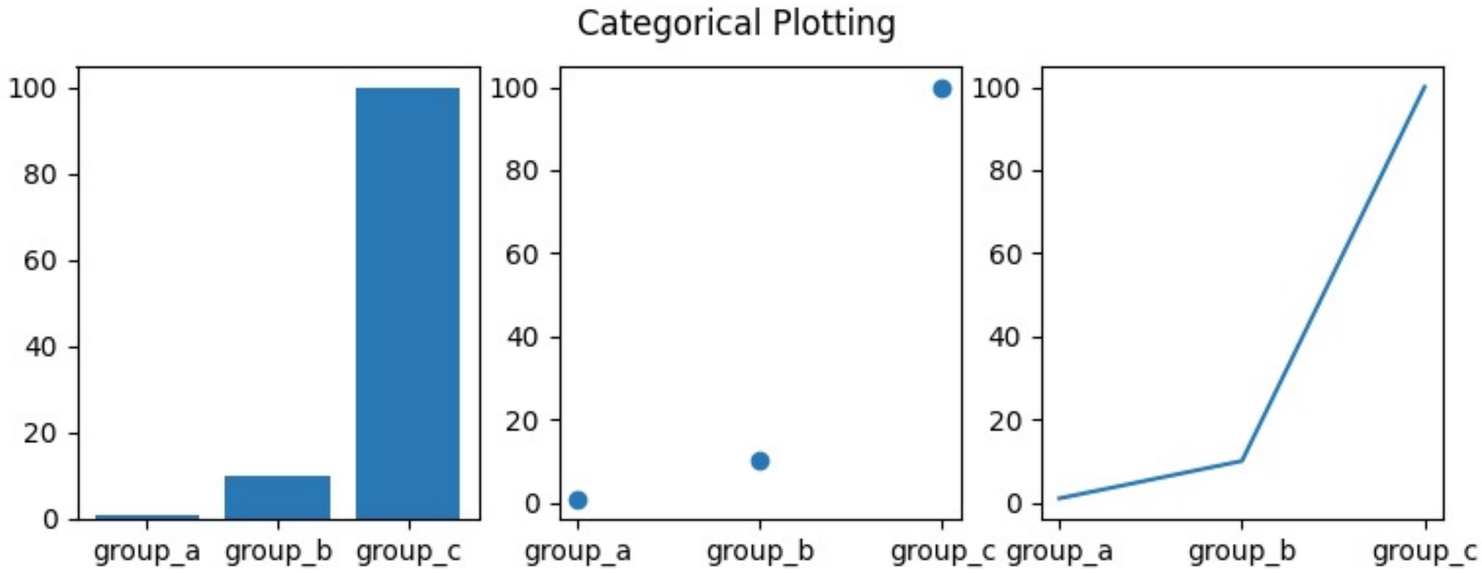
INTRODUÇÃO AO MATPLOTLIB

DESMISTIFICANDO MATPLOTLIB

Comandos MATPLOTLIB

- Plotando com variáveis categóricas:

```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]

plt.figure(figsize=(9, 3))

plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```



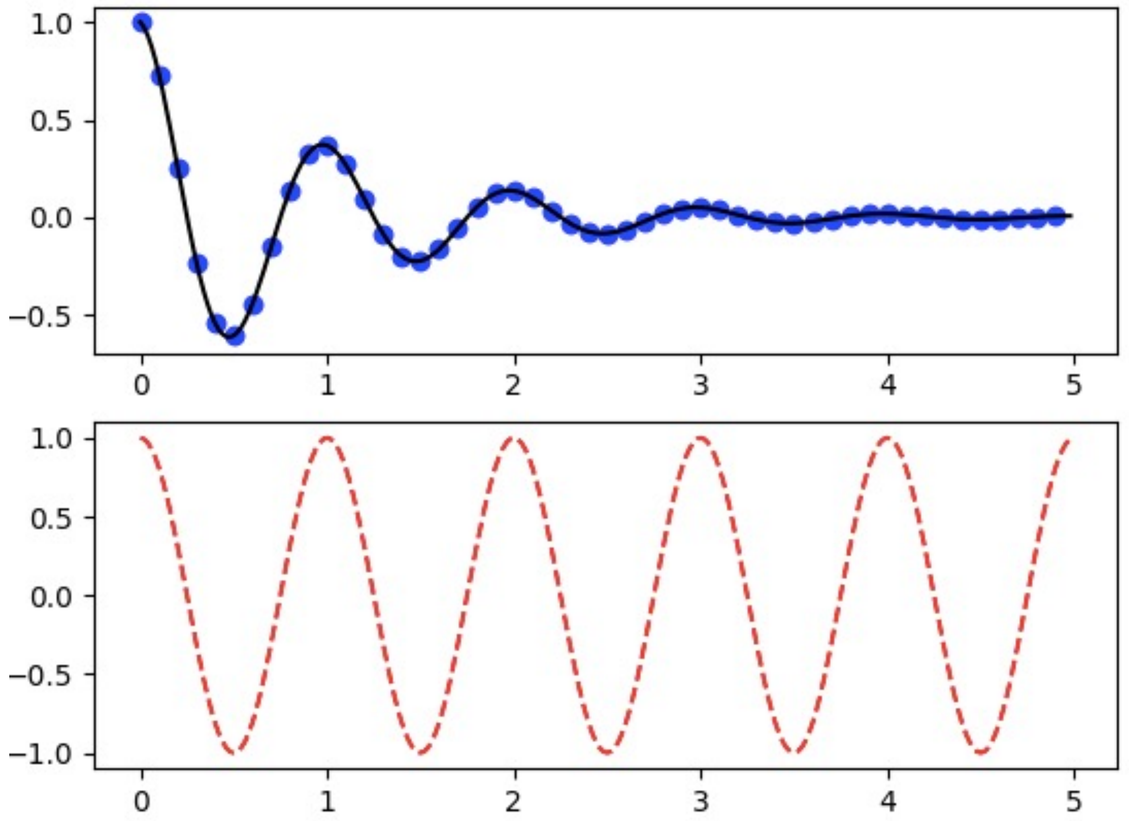
INTRODUÇÃO AO MATPLOTLIB

DESMISTIFICANDO MATPLOTLIB

Comandos MATPLOTLIB

- Plotando com várias figuras e eixos:

```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure()
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```



INTRODUÇÃO AO MATPLOTLIB

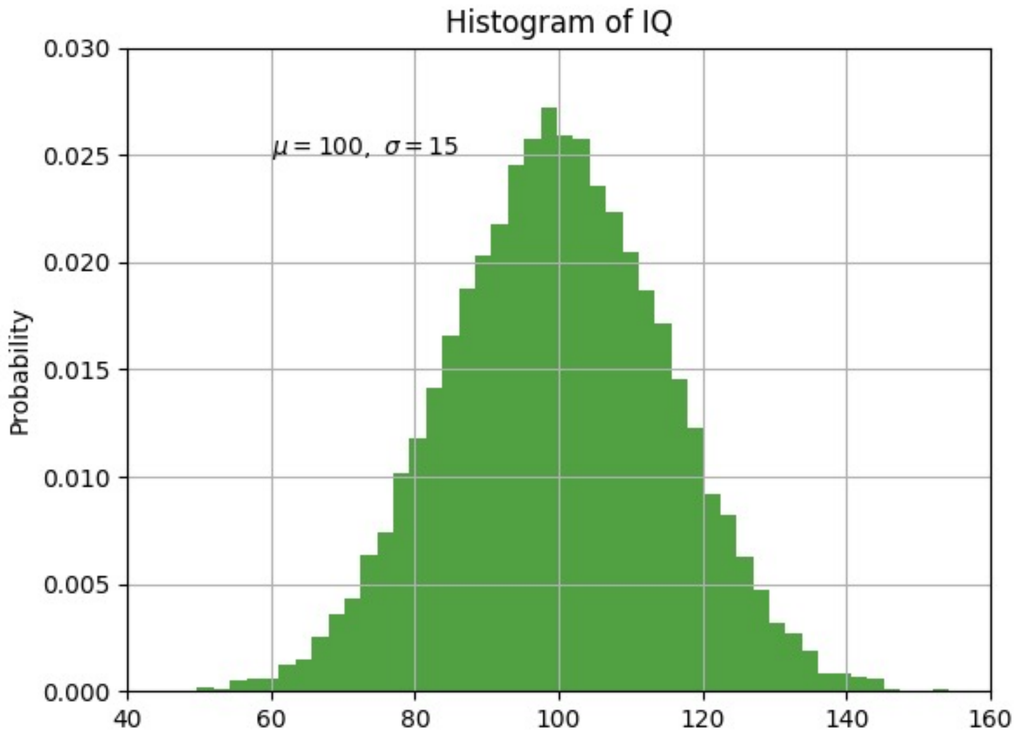
DESMISTIFICANDO MATPLOTLIB

Comandos MATPLOTLIB

- Trabalhando com texto:

```
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, density=1, facecolor='g', alpha=0.75)


plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```

```
t = plt.xlabel('my data', fontsize=14, color='red')
```



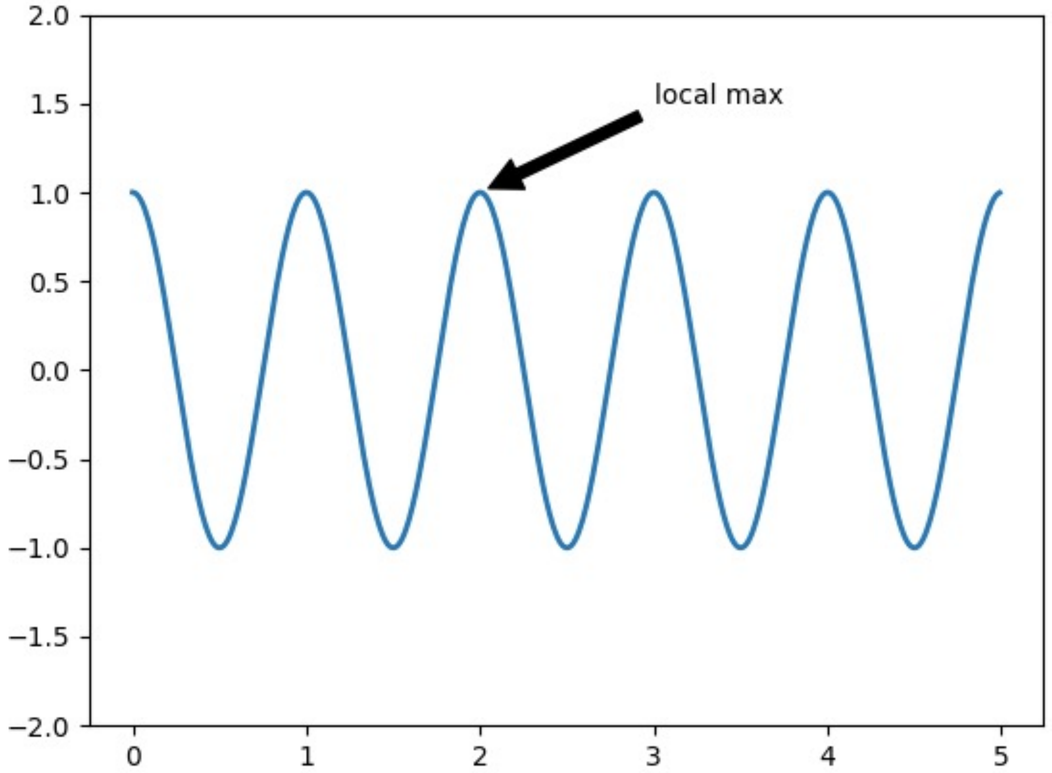
INTRODUÇÃO AO MATPLOTLIB

DESMISTIFICANDO MATPLOTLIB

Comandos MATPLOTLIB

- Anotando texto:

```
ax = plt.subplot()

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='black', shrink=0.05),
             )

plt.ylim(-2, 2)
plt.show()
```



INTRODUÇÃO AO MATPLOTLIB

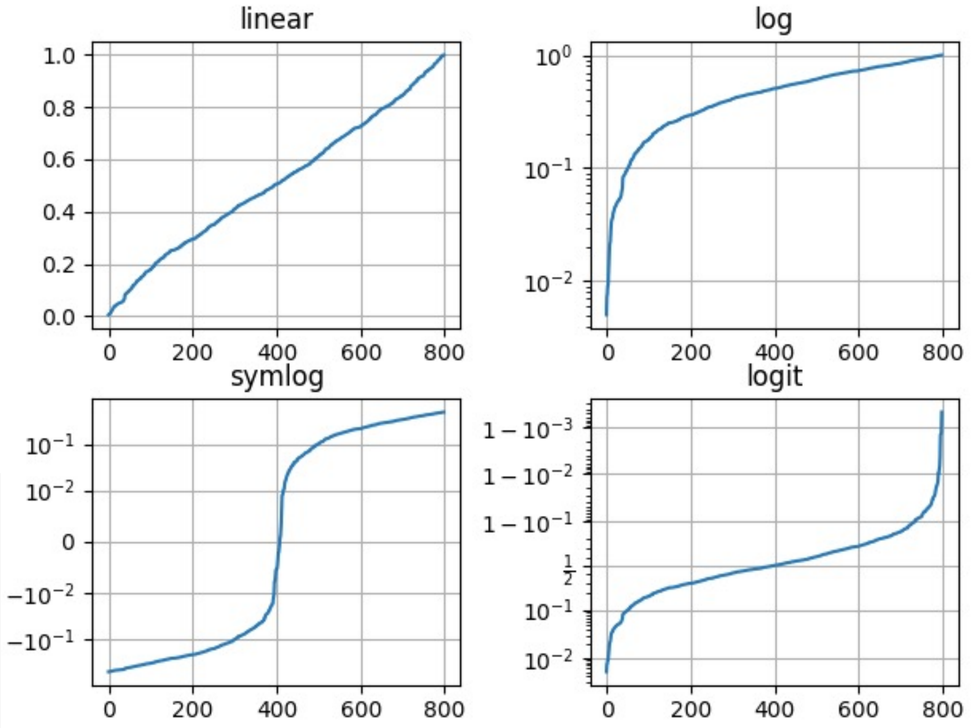
DESMISTIFICANDO MATPLOTLIB

Comandos MATPLOTLIB

- Eixos logarítmicos e não-lineares:

```
# Fixing random state for reproducibility
np.random.seed(19680801)

# make up some data in the open interval (0, 1)
y = np.random.normal(loc=0.5, scale=0.4, size=1000)
y = y[(y > 0) & (y < 1)]
y.sort()
x = np.arange(len(y))

# plot with various axes scales
plt.figure()

# linear
plt.subplot(221)
plt.plot(x, y)
plt.yscale('linear')
plt.title('linear')
plt.grid(True)

# log
plt.subplot(222)
plt.plot(x, y)
plt.yscale('log')
plt.title('log')
plt.grid(True)
```

```
# symmetric log
plt.subplot(223)
plt.plot(x, y - y.mean())
plt.yscale('symlog', linthresh=0.01)
plt.title('symlog')
plt.grid(True)

# logit
plt.subplot(224)
plt.plot(x, y)
plt.yscale('logit')
plt.title('logit')
plt.grid(True)
# Adjust the subplot layout, because the logit one may take more space
# than usual, due to y-tick labels like "1 - 10^{-3}"
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25,
                    wspace=0.35)

plt.show()
```



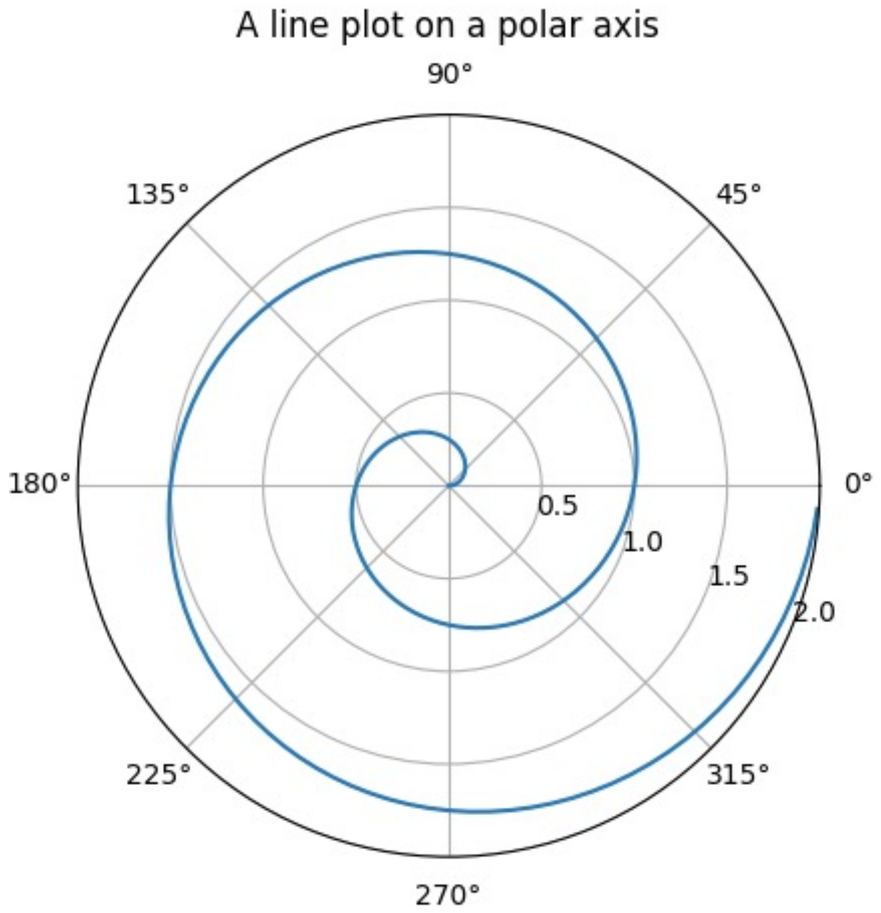
INTRODUÇÃO AO MATPLOTLIB

DESMISTIFICANDO MATPLOTLIB

Comandos MATPLOTLIB

- Gráfico polar:

```
import numpy as np
import matplotlib.pyplot as plt


r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r

fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
ax.plot(theta, r)
ax.set_rmax(2)
ax.set_rticks([0.5, 1, 1.5, 2])  # Less radial ticks
ax.set_rlabel_position(-22.5)  # Move radial labels away from plotted line
ax.grid(True)

ax.set_title("A line plot on a polar axis", va='bottom')
plt.show()
```



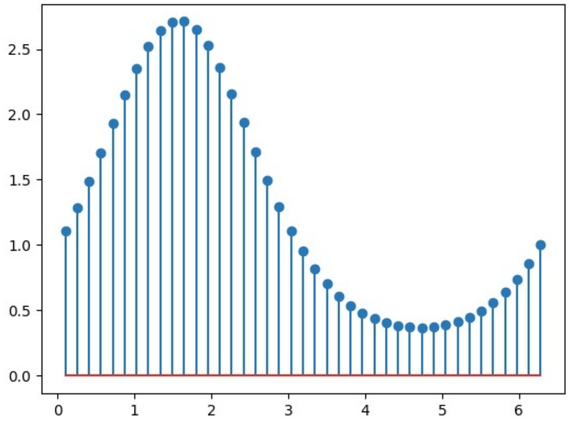
INTRODUÇÃO AO MATPLOTLIB

DESMISTIFICANDO MATPLOTLIB

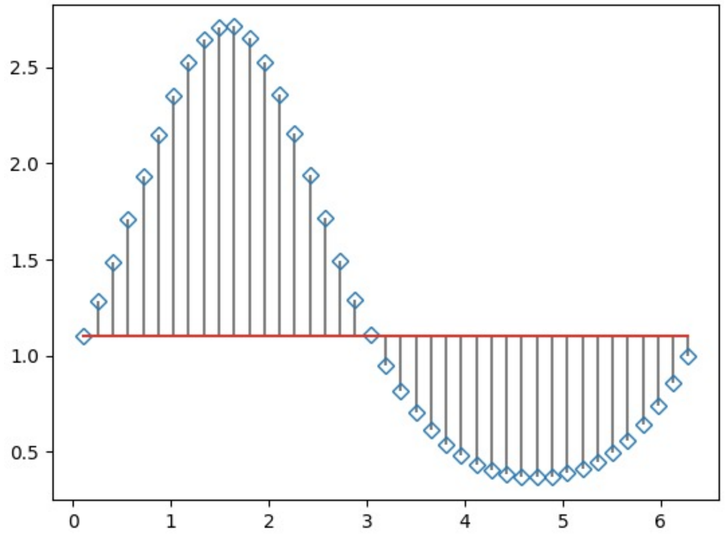
Comandos MATPLOTLIB

- Stem Plot:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0.1, 2 * np.pi, 41)
y = np.exp(np.sin(x))

plt.stem(x, y)
plt.show()
```




```
markerline, stemlines, baseline = plt.stem(
    x, y, linefmt='grey', markerfmt='D', bottom=1.1)
markerline.set_markerfacecolor('none')
plt.show()
```



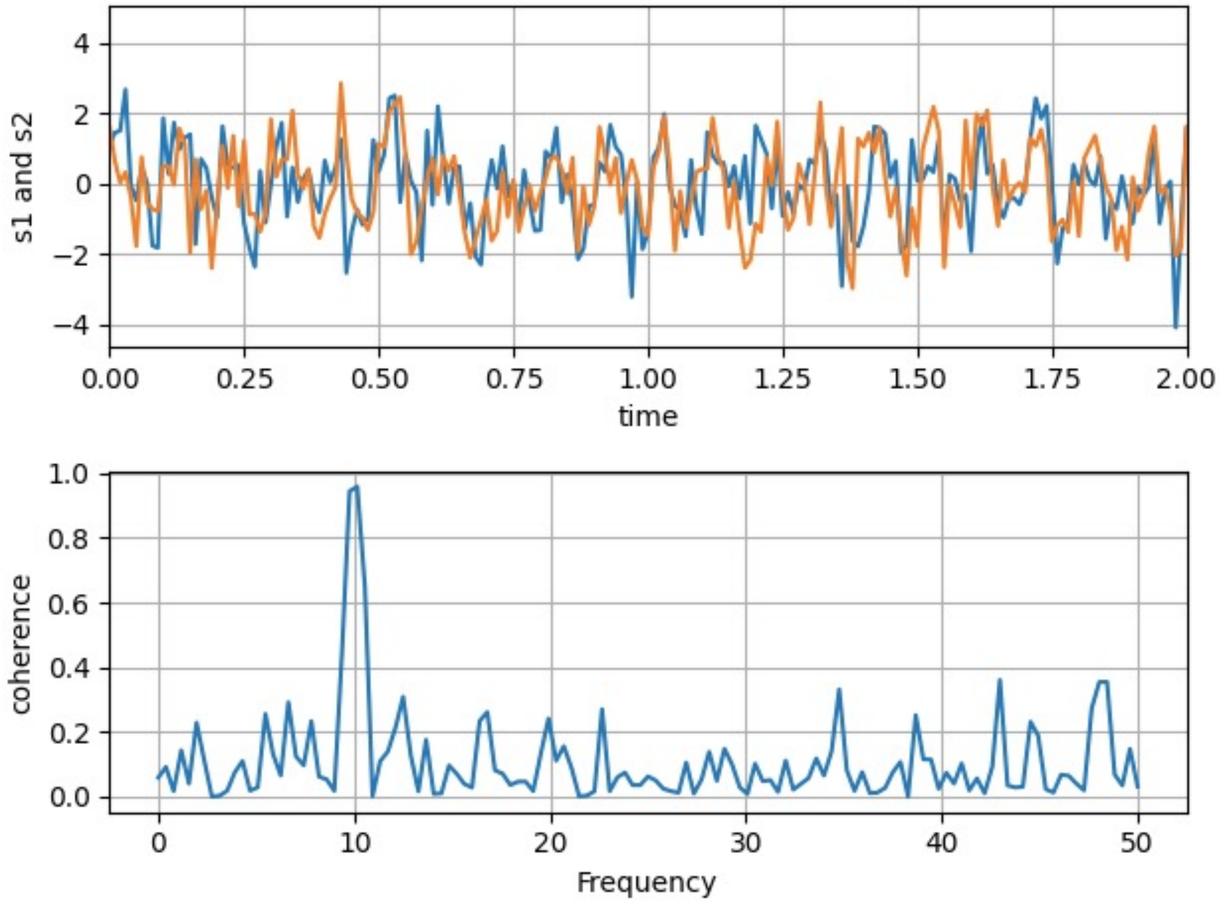
INTRODUÇÃO AO MATPLOTLIB

DESMISTIFICANDO MATPLOTLIB

Comandos MATPLOTLIB

- Stem Plot:

```
import numpy as np
import matplotlib.pyplot as plt

# Fixing random state for reproducibility
np.random.seed(19680801)

dt = 0.01
t = np.arange(0, 30, dt)
nse1 = np.random.randn(len(t))                 # white noise 1
nse2 = np.random.randn(len(t))                 # white noise 2

# Two signals with a coherent part at 10Hz and a random part
s1 = np.sin(2 * np.pi * 10 * t) + nse1
s2 = np.sin(2 * np.pi * 10 * t) + nse2

fig, axs = plt.subplots(2, 1)
axs[0].plot(t, s1, t, s2)
axs[0].set_xlim(0, 2)
axs[0].set_xlabel('time')
axs[0].set_ylabel('s1 and s2')
axs[0].grid(True)

cxy, f = axs[1].cohere(s1, s2, 256, 1. / dt)
axs[1].set_ylabel('coherence')

fig.tight_layout()
plt.show()
```



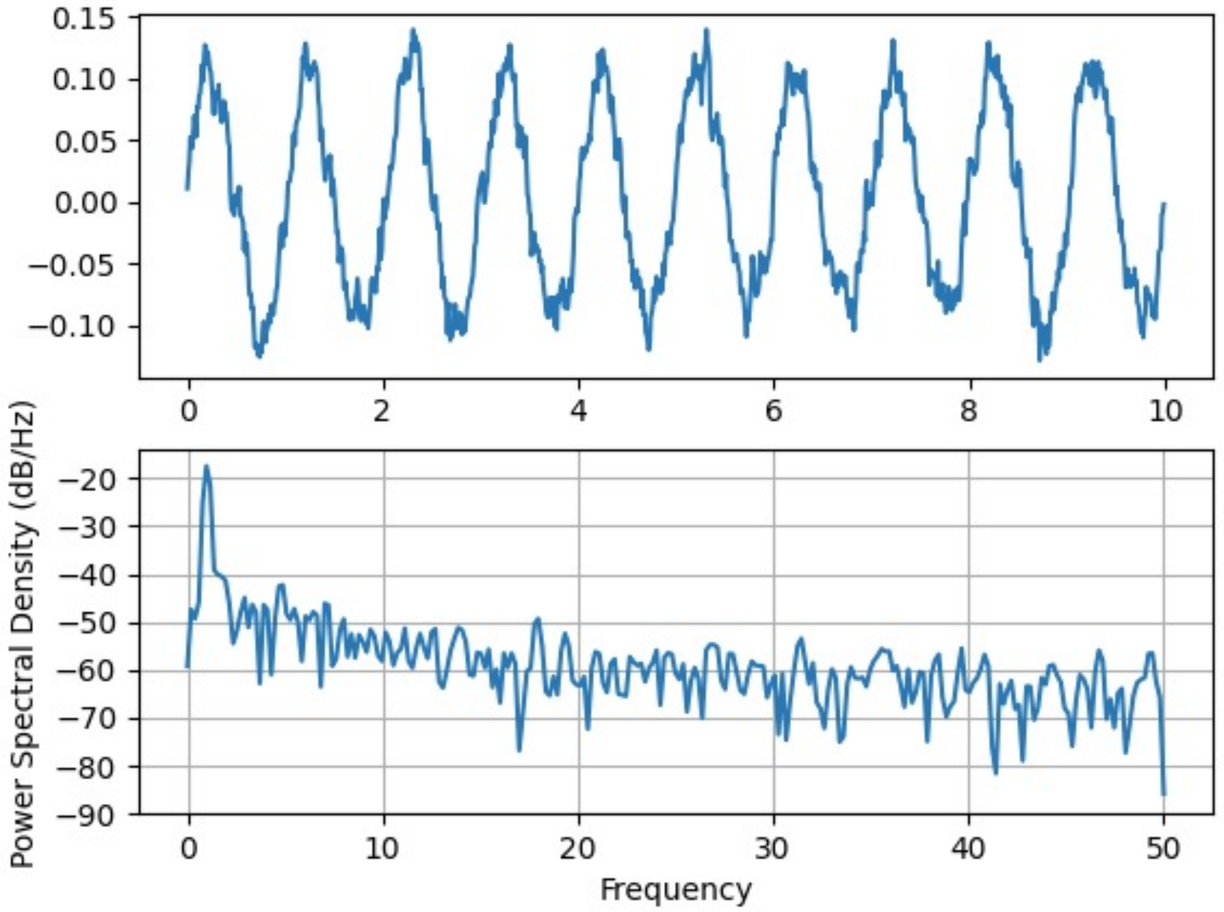
INTRODUÇÃO AO MATPLOTLIB

DESMISTIFICANDO MATPLOTLIB

Comandos MATPLOTLIB

- Coerência de 2 sinais:

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.gridspec as gridspec

# Fixing random state for reproducibility
np.random.seed(19680801)

dt = 0.01
t = np.arange(0, 10, dt)
nse = np.random.randn(len(t))
r = np.exp(-t / 0.05)

cnse = np.convolve(nse, r) * dt
cnse = cnse[:len(t)]
s = 0.1 * np.sin(2 * np.pi * t) + cnse

fig, (ax0, ax1) = plt.subplots(2, 1)
ax0.plot(t, s)
ax1.psd(s, 512, 1 / dt)

plt.show()
```



INTRODUÇÃO AO MATPLOTLIB

DESMISTIFICANDO MATPLOTLIB

Comandos MATPLOTLIB

- Representações espctrais:

```
import matplotlib.pyplot as plt
import numpy as np


np.random.seed(0)

dt = 0.01  # sampling interval
Fs = 1 / dt  # sampling frequency
t = np.arange(0, 10, dt)

# generate noise:
nse = np.random.randn(len(t))
r = np.exp(-t / 0.05)
cnse = np.convolve(nse, r) * dt
cnse = cnse[:len(t)]

s = 0.1 * np.sin(4 * np.pi * t) + cnse  # the signal

fig, axs = plt.subplots(nrows=3, ncols=2, figsize=(7, 7))

# plot time signal:
axs[0, 0].set_title("Signal")
axs[0, 0].plot(t, s, color='C0')
axs[0, 0].set_xlabel("Time")
axs[0, 0].set_ylabel("Amplitude")
```

```
# plot different spectrum types:
axs[1, 0].set_title("Magnitude Spectrum")
axs[1, 0].magnitude_spectrum(s, Fs=Fs, color='C1')

axs[1, 1].set_title("Log. Magnitude Spectrum")
axs[1, 1].magnitude_spectrum(s, Fs=Fs, scale='dB', color='C1')

axs[2, 0].set_title("Phase Spectrum ")
axs[2, 0].phase_spectrum(s, Fs=Fs, color='C2')

axs[2, 1].set_title("Angle Spectrum")
axs[2, 1].angle_spectrum(s, Fs=Fs, color='C2')

axs[0, 1].remove()  # don't display empty ax

fig.tight_layout()
plt.show()
```