

Python para Ciência de Dados e Inteligência Artificial

Aula 04: Introdução ao NumPy.

IMT – Instituto Mauá de Tecnologia

Março/2023

Prof. Jones Egydio
jones.egydio@maua.br

Introdução ao NumPy

- O **NumPy** é uma biblioteca de Python que é utilizada para computação científica e análise de dados. Suas principais características incluem:
 - **Array multidimensional:** O NumPy é baseado em arrays multidimensionais, que permitem representar dados em diferentes dimensões – operações vetorizadas;
 - **Funções matemáticas:** O NumPy inclui um grande número de funções matemáticas, como funções trigonométricas, exponenciais, logarítmicas e estatísticas, que permitem realizar operações matemáticas complexas em arrays;
 - **Indexação avançada:** Permite acessar elementos específicos de um array ou selecionar subconjuntos de um array com base em determinadas condições.
 - **Broadcasting:** Usa o conceito de broadcasting, que permite realizar operações em arrays com formas diferentes, tornando o código mais legível e eficiente.
 - **Integração com outras bibliotecas:** O NumPy é integrado com outras bibliotecas de análise de dados, como Pandas e Matplotlib, para tornar a análise de dados mais eficiente.
 - **Alta performance:** O NumPy é construído em C, o que significa que ele é muito rápido e eficiente em termos de uso de recursos do sistema.

Introdução ao NumPy

- O **NumPy** é uma biblioteca de Python que é utilizada para computação científica e análise de dados. Suas principais características incluem:
 - Biblioteca Python usada para trabalhar com álgebra linear, arrays e matrizes;
 - Fornecer um objeto de matriz que é até 50x mais rápido que as listas tradicionais do Python;
 - Array NumPy é armazenados em um local contínuo na memória;
 - É otimizado para funcionar com as arquiteturas de CPU mais recentes;
 - Demandam menos espaço de memória para mesma quantidade de elementos;
 - SIMD - Processamento vetores Single instruction, multiple data (SIMD);
 - Mais versatilidade nas operações com arrays (ex. multiplicação)
 - É codificado parcialmente em Python, mas a maioria das partes que requerem computação rápida são escritas em C ou C++.
 - O objeto array em NumPy é chamado ndarray.
 - O ndarray fornece várias funções de suporte que facilitam muito seu uso.

Introdução ao NumPy

- **Exemplo 01:** Considere a lista:

`L = [1, 2, 3, 4, 5, 6, 7]`

Deseja-se multiplicar cada item da lista por 7, escreva o código em Python para realizar tal tarefa.

NumPy: criando Arrays (1D)

```
import numpy as np

a = np.array([2, 3, 4])
print(a)
print(a.dtype)

b = np.array([1.2, 3.5, 5.1])
print(b)
print(b.dtype)

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

```
print(c*1j)
print(c)
c = np.array([1, 2, 3, 4], dtype=complex )
```

NumPy: criando Arrays (2D)

```
import numpy as np

a = np.array([[11, 12, 13], [21, 22, 23], [31, 32, 33]])
print(a)
print(a.dtype)

b = np.array([[1.2], [3.5], [5.1]])
print(b)
print(b.dtype)

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

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

NumPy: criando Arrays pré-definidos

```
import numpy as np

z = np.zeros((3, 4))
print(z)

o = np.ones((2, 3, 4), dtype=np.int16) # Com definição do tipo de dado
print(o)

i = np.identity(3)
print("\ni = \n", i)

v = np.empty((2, 3))
print(v)
```

```
hctuf(Λ)
Λ = ub*ewbλ((5^3))
```

NumPy: criando Arrays pré-definidos

```
import numpy as np

x1 = np.arange( 10, 30, 5 )
print("\nx1 = \n", x1)

x2 = np.arange( 0, 2, 0.3 ) # Também funciona para incremento real (float)
print("\nx2 = \n", x2)

x3 = np.linspace(0, 2, 9) # 9 elementos de 0 a 2
print("\nx3 = \n", x3)

x4 = np.linspace(0, 2*np.pi, 100)
print("\nx4 = \n", x4)
```

```
print("\nx4 = \n", x4)
x4 = np.linspace(0, 2*np.pi, 100)
```


NumPy: criando Arrays pré-definidos

```
import numpy as np

r1 = np.random.rand(2, 3)
print("\nr1 = \n", r1)

r2 = np.random.randint(1, 100, r1.shape)
print("\nr2 = \n", r2)
```

```
import numpy as np
r2 = np.random.randint(1, 100, r1.shape)
```

NumPy: acesso a elementos em um Array

```
import numpy as np

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

M = np.array([[11, 12, 13], [21, 22, 23], [31, 32, 33]])
print(M)

print(v[0])
print(v[-1])
print(M[0]) # será que era isso que você queria?
print(M[0,0])
print(M[2][1])
print(M[-1][-2])
```

```
print(M[-1][-2])
print(M[0][0])
print(M[2][1])
```

NumPy: Funções e Métodos

```
import numpy as np

A = np.array([[2, 3, 5], [1, 9, 7]])

print("len(A) = ", len(A)) # Sempre retorna o número de linhas
print("len(A[0, :]) = ", len(A[0, :])) # Retorna o número de colunas
print("A.max() = ", A.max())
print("A.min() = ", A.min())
print("A.sum() = ", A.sum())
print("A.sum(axis = 0) = ", A.sum(axis = 0))
print("A.sum(axis = 1) = ", A.sum(axis = 1))
print("A.prod() = ", A.prod())
print("np.where(A > 4) = ", np.where(A > 4, A, 0)) # Como isso funciona?
```

```
print("np.where(A > 4) = ", np.where(A > 4, A, 0)) # Como isso funciona?
print("A.prod() = ", A.prod())
print("A.sum(axis = 1) = ", A.sum(axis = 1))
```

NumPy: operações básicas

Broadcasting

```
import numpy as np

a = np.array([20, 30, 40, 50])
b = np.arange(1, 5)

print("a + b =", a + b)
print("a - b =", a - b)
print("a * b =", a * b) # Cuidado não é o prooduto de duas Matrizes
print("a / b =", a / b)
print("a @ b =", a @ b)
print("a.dot(b) =", a.dot(b)) # Produto escalar
print("a > b =", a > b)
```

```
print("a > p =", a > p)
print("a.dot(p) =", a.dot(p)) # Produto escalar
print("a @ p =", a @ p)
```

NumPy: operações básicas

Produtos de matrizes

```
import numpy as np

# Cuidado com a dimensão dos arrays na multiplicação de matrizes
A = np.array([[2, 3, 7], [1, 7, 5]])
B = np.array([[1, 5], [7, 8], [9, 8]])

print("\nA @ B =\n", A @ B)
print("\nA @ B.T =\n", A @ B.T) # B.T é a matriz transposta de B
```

```
print("\nA @ B.T =\n", A @ B.T) # B.T é a matriz transposta de B
print("\nA @ B =\n", A @ B)
```

NumPy: indexação e fatiamento

Arrays com uma dimensão

```
import numpy as np

a = np.arange(10)**3

print("a =", a)
print("\na[2] =", a[2])
print("a[2:5] =", a[2:5])

a[:6:2] = 1000 # O que isso faz?
print("a =", a)
print("a[ : :-1] =", a[ : :-1])

for i in a: # o Array continua sendo iterativo
    print(i**(1/3))
```

```
print(i**(1/3))
for i in a: # o Array continua sendo iterativo
```

NumPy: indexação e fatiamento

Arrays com multidimensão

```
import numpy as np

def f(x,y):
    return 10*x+y

b = np.fromfunction(f, (5,4), dtype=int)
print("b =\n", b)
print("\nb[2,3] =", b[2,3])
print("\nb[0:5, 1] =", b[0:5, 1])
print("\nb[:,1] =", b[:,1]) # Qual a diferença?
print("\nb[1:3,:]\n", b[1:3,:])
```

```
b[1:3,:]\n", b[1:3,:])
b[:,1] =", b[:,1]) # Qual a diferença?
b[0:5, 1] =", b[0:5, 1])
```

NumPy: estatística

Análise exploratória simples

```
import numpy as np

A = np.random.rand(10)

print(A)
print(np.mean(A) )
print(np.std(A) )
print(np.var(A) )
```

```
print(np.var(A) )
print(np.std(A) )
```




Referências bibliográficas

MENEZES, N. N. C., Introdução à programação com Python, 3ª Edição. São Paulo: Editora Novatec, 2019.

Notas de aula: Prof. Anderson Harayashini Moreira, pós-graduação em CD e IA, IMT, 2022.