Biblioteca Numpy

Prof. Gabriel Rodrigues Caldas de Aquino

gabrielaquino@ic.ufrj.br
Instituto de Computação - Universidade Federal do Rio de Janeiro

Compilado em:
September 23, 2025

NumPy - Introdução

O que é NumPy?

- Pacote fundamental para **computação científica** em Python

Objeto principal: `numpy.ndarray`

- Vetor **n-dimensional** (arrays multidimensionais)
- Características fundamentais:
  - **Tamanho fixo** (definido na criação)
  - **Indexado** por tuplas de inteiros positivos
  - **Homogêneo** - todos elementos do mesmo tipo

Documentação Oficial

Há diversas métodos e atributos disponíveis no NumPy, os quais podem ser consultados na documentação oficial no endereço: `https://numpy.org/doc/stable/index.html`

NumPy - Arrays

Importe o módulo e crie arrays a partir de listas

```
import numpy as np
np.array(lista)
```

Exemplos

```
# Array 1D (vetor)
x = np.array([1, 2, 3])


# Array 2D (matriz)
y = np.array([[1., 0., 0.],
              [0., 1., 0.]])
```

Saída dos Exemplos

```
#Saída:
>>> x
array([1, 2, 3])

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

Tipos de Numpy array

```
>>> type(x)
numpy.ndarray

>>> type(y)
 numpy.ndarray
```

Características

- `x` é um array **1-dimensional** (vetor)
- `y` é um array **2-dimensional** (matriz)
- Ambos são do tipo `numpy.ndarray`

NumPy - O Atributo dtype

Definição

O dtype define o tipo dos elementos armazenados no array NumPy

Exemplo Inicial

```
minhalista = [1, 2, 3, 4, 5]
arr = np.array(minhalista)
print(arr)          # array([1, 2, 3, 4, 5])
print(type(arr))    # <class 'numpy.ndarray'>
print(arr.dtype)    # dtype('int64')
```

Com Número Decimal

```
minhalista = [1, 2, 3, 4, 5.5]
arr = np.array(minhalista)
print(arr.dtype)  # dtype('float64')
```

Com String

```
minhalista = [1, 2, 3, 4, "ola"]
arr = np.array(minhalista)
print(arr.dtype)  # dtype('<U21')
```

NumPy - Propriedades de Arrays

Criação de Array 2D

```
arr2 = np.array([[1, 2],
                 [3, 4]])
```

Propriedades Fundamentais

- `.ndim` - Número de dimensões
- `.shape` - Tupla com tamanho em cada dimensão
- `.size` - Número total de elementos
- `.dtype` - Tipo dos dados

Aplicado ao Exemplo

```
>>> arr2.ndim
2
>>> arr2.shape
(2, 2)
>>> arr2.size
4
>>> arr2.dtype
dtype('int64')
```

NumPy - Propriedades Básicas de Arrays

Propriedade	**Descrição**
`ndarray.ndim`	Número de eixos (dimensões) do array.
`ndarray.shape`	Dimensões do array. Uma tupla de inteiros indicando o tamanho em cada dimensão. Para uma matriz com n linhas e m colunas, shape será (n, m). O comprimento da tupla shape é igual ao número de dimensões (`ndim`).
`ndarray.size`	Número total de elementos do array.
`ndarray.dtype`	Objeto que descreve o tipo dos elementos no array. Podem ser usados tipos padrão do Python ou tipos específicos do NumPy como `numpy.int32`, `numpy.int16` e `numpy.float64`.

NumPy – Arrays com Valores Pré-definidos

Por que usar valores pré-definidos?

- Facilita a **inicialização** de matrizes antes do preenchimento com dados
- Evita **erros** na alocação de memória para grandes arrays
- Útil para **cálculos numéricos** e simulações

Funções de Criação

Função	**Descrição**
`np.zeros(shape)`	Cria array preenchido com **zeros**
`np.ones(shape)`	Cria array preenchido com **uns**
`np.empty(shape)`	Cria array com **valores aleatórios**

NumPy – Arrays 1D com Valores Pré-definidos

Exemplos 1D

```
# Array de 5 zeros
np.zeros(5)

# Array de 3 uns
np.ones(3)

# Array vazio 4 posições
np.empty(4)
```

Dica Importante

Especifique sempre o `dtype` para controle preciso do tipo numérico:
`np.zeros(5, dtype=np.float32)`

NumPy – Arrays 2D com Valores Pré-definidos

Exemplos 2D

```
# Matriz 2x3 de zeros
np.zeros((2,3))

# Matriz 3x3 de uns
np.ones((3,3))

# Matriz 2x2 vazia
np.empty((2,2))
```

Dica Importante

Ao criar arrays multidimensionais, lembre-se de usar dois parênteses:
o primeiro envolve a tupla com as dimensões, o segundo é da chamada da função.

Exemplo correto: `np.ones((3,3), dtype=int)`

Exemplo: Criação de Arrays com Zeros

```
# importando o módulo
import numpy as np

# criando um array com cinco elementos sendo zeros
array_zeros = np.zeros(5)
print(array_zeros)
# Saída: [0. 0. 0. 0. 0.]

# criando uma matriz 3x4 preenchida com zeros
matriz_zeros = np.zeros((3, 4))
print(matriz_zeros)
# Saída: [[0. 0. 0. 0.]
          [0. 0. 0. 0.]
          [0. 0. 0. 0.]]
```

Exemplo: Criação de Arrays com np.empty

```
# importando o módulo
import numpy as np

# criando uma matriz 3x3 sem inicialização garantida
array_empty = np.empty((3, 3))
print(array_empty)
# Saída: [[4.67296746e-307 1.69121096e-306 1.37959131e-306]
          [1.11261162e-306 1.11260619e-306 9.34609790e-307]
          [8.45559303e-307 9.34600963e-307 1.37959740e-306]]
```

Observação

Os valores podem ser aleatórios, pois `np.empty` não inicializa os elementos, apenas aloca espaço na memória.

Criando Arrays com `np.arange()`

Array 1D (Vetor)

```
# Vetor de 0 a 8
v = np.arange(9)
print(v)
# [0 1 2 3 4 5 6 7 8]

# Vetor de 0 a 8 pulando de 2 em 2
v2 = np.arange(0, 9, 2)
print(v2)
# [0 2 4 6 8]
```

Dica

`np.arange(início, fim, passo)` cria um vetor com espaçamento definido.
O valor final **não é incluído**. O passo pode ser positivo ou negativo!

Criando Matrizes com `reshape()`

Matriz 2x2

```
# Criar e redimensionar
m2x2 = np.arange(1,5).reshape(2,2)
print(m2x2)
# [[1 2]
#  [3 4]]
```

Matriz 3x3

```
# Criar e redimensionar
m3x3 = np.arange(9).reshape(3,3)
print(m3x3)
# [[0 1 2]
#  [3 4 5]
#  [6 7 8]]
```

Dica

- `reshape(linhas, colunas)` para transformar um array 1D em matriz.
- **O número total de elementos deve ser compatível**

NumPy – Entendendo Eixos (Axes)

O que são eixos em NumPy?

Em arrays multidimensionais, os eixos representam as direções ao longo das quais as operações podem ser aplicadas.

- `axis=0`: opera coluna por coluna
  Saída: `[coluna1, coluna2, coluna3]`
- `axis=1`: opera linha por linha
  Saída: `[linha1, linha2, linha3]`

Dica Visual

Pense que o `axis` é a dimensão que será "reduzida".
`axis=0` colapsa cada coluna.
`axis=1` colapsa cada linha.

NumPy – Exemplo Prático com Eixos

Soma com axis=0 e axis=1

```
import numpy as np
matriz = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

# Soma por colunas (axis=0)
print(np.sum(matriz, axis=0))
# Saída: [12 15 18]

# Soma por linhas (axis=1)
print(np.sum(matriz, axis=1))
# Saída: [ 6 15 24]
```

Operações ao Longo dos Eixos em NumPy

Operação	**axis = 0 (colunas)**	**axis = 1 (linhas)**
`np.sum()`	Soma por coluna	Soma por linha
`np.mean()`	Média por coluna	Média por linha
`np.max()`	Máximo por coluna	Máximo por linha
`np.min()`	Mínimo por coluna	Mínimo por linha

Exemplo: Matriz de vendas

- Linhas representam lojas
- Colunas representam produtos (Lápis, borracha, caderno)

Codigo

```
vendas = np.array([
    [10, 20, 30],  # Loja 1
    [15, 25, 35],  # Loja 2
    [12, 18, 28],  # Loja 3
    [8, 22, 26],   # Loja 4
])
np.sum(vendas, axis=0) # Soma por coluna (produtos)
# [45 85 119]
np.sum(vendas, axis=1) # Soma por linha (lojas)
# [60 75 58 56]
np.mean(vendas, axis=0) # Média por produto (em todas as lojas)
# [11.25 21.25 29.75]
```

Operações Aritméticas em NumPy – Adição Escalar (Vetor)

Adição Escalar

```
import numpy as np

A = np.arange(1, 10)
print(A)
# [1 2 3 4 5 6 7 8 9]

print(A + 5)  # Adição escalar
# [ 6  7  8  9 10 11 12 13 14]
```

Operações Aritméticas em NumPy – Matriz + Escalar

Adição Escalar em Matrizes

```
import numpy as np

B = np.random.randint(0, 10, (3,3))
print(B)
# [[4 3 1]
#  [6 0 7]
#  [9 3 3]]

print(B + 5)
# [[ 9  8  6]
#  [11  5 12]
#  [14  8  8]]
```

Exemplo: Estoque em Papelarias

- Cada linha representa uma loja;
- Cada coluna representa um item: lápis, borracha e caderno.

Código

```
import numpy as np
# Quantidade atual em 3 lojas
estoque = np.array([[10, 5, 2],
                    [3,  8, 4],
                    [6,  2, 7]])

novo_estoque = estoque + 5 # Reposição de 5 unidades em cada item

print(novo_estoque)
# [[15 10  7]
#  [ 8 13  9]
#  [11  7 12]]
```

Operações entre Vetores em NumPy - Soma

Vetor A

```
import numpy as np

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

Vetor B

```
B = np.array([10, 20, 30, 40, 50])
print(B)
# [10 20 30 40 50]
```

Soma A + B

```
print(A + B)
# [11 22 33 44 55]
```

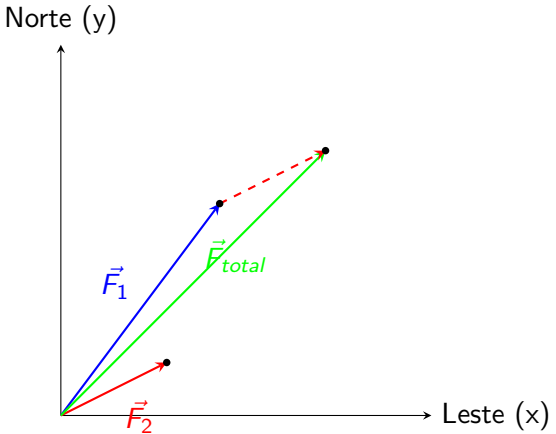
Soma de Vetores com NumPy – Exemplo

Forças

```
import numpy as np

F1 = np.array([3, 4])
F2 = np.array([2, 1])

F_total = F1 + F2
print(F_total)
# [5 5]
```



Operações entre Matrizes em NumPy - Soma

Matriz B

```
B = np.random.randint(0, 10, (3,3))
print(B)
# [[4 3 1]
#  [6 0 7]
#  [9 3 3]]
```

Matriz C

```
C = np.random.randint(0, 10, (3,3))
print(C)
# [[3 7 1]
#  [6 4 5]
#  [0 1 7]]
```

Soma B + C

```
print(B + C)
# [[ 7 10  2]
#  [12  4 12]
#  [ 9  4 10]]
```

Operações entre Matrizes – Compras em uma Loja

Cada linha é um cliente e cada coluna um item comprado (lápis, caneta, caderno).

Compras no Dia 1

```
dia1 = np.array([
    [1, 2, 0],  # Cliente 1
    [0, 1, 3],  # Cliente 2
    [2, 0, 1]   # Cliente 3
])
```

Compras no Dia 2

```
dia2 = np.array([
    [0, 1, 2],
    [1, 0, 1],
    [1, 2, 1]
])
```

Total de Compras (Dia 1 + Dia 2)

```
total = dia1 + dia2
print(total)
# [[1 3 2]
#  [1 1 4]
#  [3 2 2]]
```

Multiplicação Escalar de Vetor

Código NumPy – Multiplicação Escalar

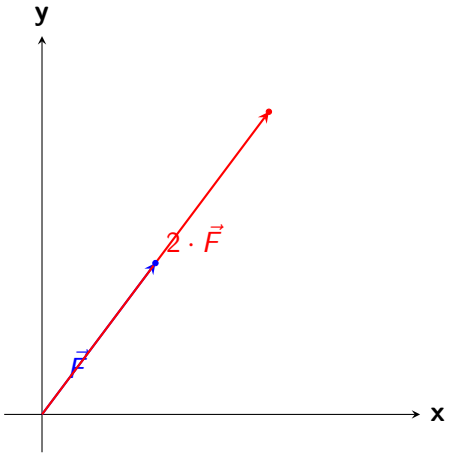
```
import numpy as np

F = np.array([3, 4])
print(F)
#[3 4]

# Multiplicando por 2
print(2 * F)
#[6 8]
```

Vetores

$$\vec{F} = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \quad 2 \cdot \vec{F} = \begin{bmatrix} 6 \\ 8 \end{bmatrix}$$



NumPy - Multiplicação Element-wise

Exemplo: Receita por produto (quantidade × preço)

```
import numpy as np

Quantidade_produtos = np.array([10, 5, 2])
Preco_produtos = np.array([1, 0.5, 20])

# Gasto por produto
print(Quantidade_produtos * Preco_produtos)
# [10.   2.5 40. ]
```

Multiplicação de cada elemento

Cenário

Cada linha representa uma loja.Cada coluna representa um produto (Lápis, Borracha, Caderno). Queremos saber a receita por produto em cada loja (quantidade $\times$ preço unitário).

Quantidade Vendida (B)

```
B = np.array([[4, 3, 1],
              [6, 0, 7],
              [9, 3, 3]])
```

Preço Unitário (C)

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

Receita por Loja e Produto (B * C)

```
print(B * C)
# [[12 21  1]
#  [36  0 35]
#  [ 0  3 21]]
```

Multiplicação Matricial com `np.dot()`

Matriz B (3x3)

```
B = np.array([[4, 3, 1],
              [6, 0, 7],
              [9, 3, 3]])
```

Matriz C (3x3)

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

Resultado `np.dot(B, C)`

```
print(np.dot(B, C))
# [[ 30  41  26]
#  [ 18  55  41]
#  [ 45  78  45]]
```

Diferença Fundamental

- `B * C`: Multiplicação elemento a elemento
- `np.dot(B,C)`: Multiplicação de matrizes

Explicação: Multiplicação Matricial com `np.dot()`

Dado que temos duas matrizes quadradas **B** e **C**, ambas de dimensão 3×3. Estamos comparando duas formas distintas de multiplicação em NumPy: a multiplicação elemento a elemento (`*`) e a multiplicação matricial (`np.dot()`).

A diferença das duas operações:
Diferente da multiplicação elemento a elemento (que faz $B[i][j] \times C[i][j]$), a multiplicação matricial segue a regra:
Para cada elemento da matriz resultante, fazemos o produto escalar da linha i de B pela coluna j de C.

Exemplo prático:
Para calcular o valor na posição (0,0) do resultado:

$$4 \times 3 + 3 \times 6 + 1 \times 0 = 12 + 18 + 0 = 30$$

Esse processo se repete para cada posição da matriz resultante.

Transposição de Matrizes em NumPy

Matriz Original

```
B = np.array([[4, 3, 1],
              [6, 0, 7],
              [9, 3, 3]])
```

Método 1: Atributo .T

```
print(B.T)
# [[4 6 9]
#  [3 0 3]
#  [1 7 3]]
```

Método 2: Função transpose()

```
print(B.transpose())
# [[4 6 9]
#  [3 0 3]
#  [1 7 3]]
```

Resolver Sistema Linear (np.linalg.solve(a,b))

Considere o sistema linear:

$$\begin{cases} 2x + 3y - z = 5 \\ 4x + y + 2z = 6 \\ -3x + 2y + z = -4 \end{cases}$$

Representamos na forma matricial $Ax = b$:

$$A = \begin{bmatrix} 2 & 3 & -1 \\ 4 & 1 & 2 \\ -3 & 2 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 5 \\ 6 \\ -4 \end{bmatrix}$$

Código Python para resolver usando NumPy:

```
import numpy as np
A = np.array([[2, 3, -1],
              [4, 1, 2],
              [-3, 2, 1]])
b = np.array([5, 6, -4])
x = np.linalg.solve(A, b)
print("Solução: ", x)
```

Resolva

Considere o seguinte sistema linear:

$$\begin{cases} 3x_1 - 2x_2 + 4x_3 + x_4 - x_5 + 2x_6 = 7 \\ -2x_1 + x_2 - x_3 + 3x_4 + 5x_5 - x_6 = -3 \\ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 10 \\ 4x_1 - x_2 + 2x_3 - x_4 + 3x_5 - 2x_6 = 2 \\ -3x_1 + 5x_2 - x_3 + 2x_4 - 4x_5 + x_6 = 5 \\ 2x_1 + 3x_2 - 2x_3 + x_4 + x_5 - 3x_6 = -1 \end{cases}$$

Forma Matricial

Representamos o sistema como $Ax = b$:

$$A = \begin{bmatrix} 3 & -2 & 4 & 1 & -1 & 2 \\ -2 & 1 & -1 & 3 & 5 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & -1 & 2 & -1 & 3 & -2 \\ -3 & 5 & -1 & 2 & -4 & 1 \\ 2 & 3 & -2 & 1 & 1 & -3 \end{bmatrix}, \quad b = \begin{bmatrix} 7 \\ -3 \\ 10 \\ 2 \\ 5 \\ -1 \end{bmatrix}$$

Resolvendo com NumPy

Código Python para resolver o sistema com NumPy:

```
import numpy as np

A = np.array([
    [3, -2, 4, 1, -1, 2],
    [-2, 1, -1, 3, 5, -1],
    [1, 1, 1, 1, 1, 1],
    [4, -1, 2, -1, 3, -2],
    [-3, 5, -1, 2, -4, 1],
    [2, 3, -2, 1, 1, -3]
])

b = np.array([7, -3, 10, 2, 5, -1])

x = np.linalg.solve(A, b)
print("Solução:", x)
```