

Módulo Numpy

Cálculos matriciais em Python podem ser realizados facilmente a partir do módulo **Numpy**. A importação de um módulo **Numpy** se dá da seguinte forma:

```
Python  
1.      import numpy as np
```

O **np**, no código acima, representa uma abreviação do módulo **Numpy**, em resumo, entende-se como “importe o módulo numpy como nome np”. Após isso, para acessar as funcionalidades do módulo, basta digitar **np**.

Módulo Numpy

O módulo Numpy conta com diversas operações entre matrizes e vetores. Este módulo possui sofisticadas funções que permitem a realização, de forma fácil e poderosa, de cálculos de Álgebra Linear e Geometria Analítica em programas escritos na linguagem Python.

Convertendo uma lista em um array Numpy

Inicialmente, para construir uma matriz ou um vetor capaz de ser manipulado pelo módulo **numpy**, precisamos convertê-lo em um *array* do próprio **numpy**. Veja o código abaixo.

```
Python  
1.      import numpy as np  
2.      nome = np.array(lista_base)
```

A variável “*lista_base*” é, inicialmente, uma lista comum que será convertida em um *array* do módulo **numpy**.

Por exemplo, para construir uma matriz A definida matematicamente a seguir:

$$A = [1 \ 2 \ 3]$$

Basta executar os seguintes comandos usando o módulo Numpy:

Python

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

Ou seja, o **numpy** construirá a matriz especificada segundo a definição acima e a armazenará na variável “a”.

A construção de matrizes bidimensionais pode ser feita da mesma forma. Por exemplo, para construir a matriz B abaixo, tem-se o seguinte código Python.

$$B = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$

Python

1. `import numpy as np`
2. `b = np.array([[1, 2], [3, 4]])`

Neste caso o **numpy** criará uma matriz em Python na mesma estrutura de linhas e colunas dada pela definição matemática da matriz B.

A vantagem do uso do **numpy** é o fato de que operações matriciais se tornam tão simples quanto operações com variáveis comuns. Por exemplo, para somar duas matrizes de mesmo tamanho ou multiplicar uma matriz por um escalar, se torna uma simples operação de multiplicação entre duas variáveis.

Exemplos práticos do módulo Numpy

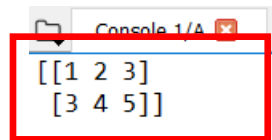
Os exemplos a seguir tratarão algumas das operações e facilidades que o uso do Numpy pode proporcionar.

Exemplo 1) Imprimir uma matriz no console. Observe, que assim que a matriz se torna um array Numpy, pode-se exibir seu conteúdo com um simples comando **print**.

Python

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

Figura 18 - Saída esperada no console



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

Fonte: Print screen da interface do Spyder 3.7

Fonte: Código desenvolvido pelo próprio autor

Exemplo 2) Este exemplo demonstra como somar duas matrizes A e B, dadas por:

$$A = \begin{bmatrix} 1 & 1 & 2 & 2 \end{bmatrix}, \text{ e } B = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix}$$

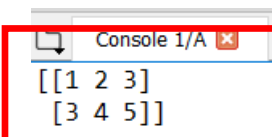
As linhas 2 e 3, do código abaixo, são as linhas que constroem as matrizes A e B manipuláveis pelo **numpy**.

Python

```
1. import numpy as np
2. a = np.array([ [1, 1], [2, 2] ])
3. b = np.array([ [1, 0], [1, 0] ])
4. c = a + b
5. print(c)
```

A linha 4 cria uma nova matriz C resultante da soma de ambas as matrizes. Por fim, na linha 5, pode-se exibir a matriz resultante.

Figura 19 - Saída esperada no console



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

Fonte: Print screen da interface do Spyder 3.7

Fonte: Código desenvolvido pelo próprio autor

Exemplo 3) Usando o módulo Numpy este exemplo demonstra como multiplicar a matriz A pelo escalar 2.

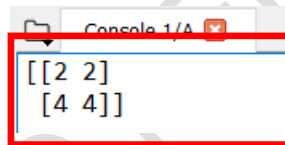
$$A = \begin{bmatrix} 1 & 1 & 2 & 2 \end{bmatrix}$$

Python

```
1. import numpy as np
2. a = np.array([ [1, 1], [2, 2] ])
3. b = a * 2
4. print(b)
```

Observe que na linha 3 um novo *array* **numpy** é criado a partir do resultado a multiplicação de cada um dos elementos de “a” por 2.

Figura 20 - Saída esperada no console



```
[[2 2]
 [4 4]]
```

Fonte: Print screen da interface do Spyder 3.7

Fonte: Código desenvolvido pelo próprio autor

Exemplo 4) Este exemplo demonstra como ocorre leitura de dados do teclado para a montagem de uma matriz a partir do módulo **numpy**.

Python

```
1. import numpy as np
2. a = [ ]
3. for i in range(2):
4.     linha = [ ]
5.     for j in range (2):
6.         x = input ()
7.         linha.append(int(x))
8.     a.append(linha)
9. a = np.array(a)
```

```
10. print(a)
```

Observe que o programa acima cria uma “lista de listas”, referenciado da linha 3 a linha 8, e somente após isso, cria uma representação na forma de *array numpy*.

Outras funções importantes da Numpy

Além das operações já citadas, o **numpy** apresenta diversas operações matriciais muito importantes, seguem algumas mais utilizadas

Tabela 1 – Exemplos de funções do módulo numpy.

| | |
|--------------------|--|
| zeros(col) | Cria uma matriz de uma linha por col colunas preenchidas por zeros. |
| zeros((lin, col)) | Cria uma matriz com lin linhas por col colunas preenchidas de zeros. |
| eye (n) | Cria uma matriz identidade de n linhas por n colunas. |
| linalg.det(matriz) | Calcula o determinante da matriz quadrada. |
| linalg.inv(matriz) | Calcula a matriz inversa de uma matriz quadrada |
| m1.dot(m2) | Calcula o produto matricial entre as matrizes m1 e m2. |
| transpose(matriz) | Calcula a transposta de uma matriz. |

Fonte: próprio autor.

Exemplos práticos do módulo numpy

Os exemplos que seguem demonstram a utilização de outras funções disponíveis no módulo **numpy**. Lembre-se que o módulo é extenso e possui muitas funcionalidades, as que aqui estão são as utilizadas no escopo desta apostila.

Exemplo 1) Calculando o determinante da seguinte matriz A, dada por:

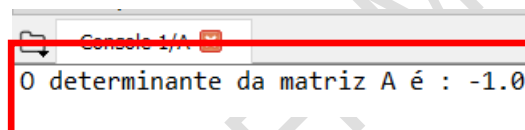
$$A = \begin{bmatrix} 1 & 2 & 2 & 3 \end{bmatrix}$$

Python

```
1. import numpy as np
2. a = np.array([ [1, 2], [2, 3] ])
3. det = np.linalg.det(a)
4. print ("O determinante da matriz A é:", det)
```

A linha 3 do código acima realiza tal operação usando uma função específica do módulo **numpy**.

Figura 21 - Saída esperada no console



```
O determinante da matriz A é : -1.0
```

Fonte: Print screen da interface do Spyder 3.7

Fonte: Código desenvolvido pelo próprio autor

Exemplo 2) Construindo uma matriz preenchida de zeros e uma matriz identidade.

Python

```
1. import numpy as np
2. a = np.zeros((3, 3))
3. b = np.eye(3)
4. print(a)
5. print(b)
```

Figura 22 - Explorador de variáveis

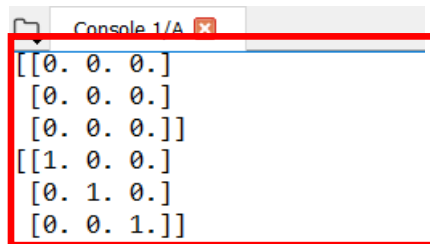
| Explorador de variáveis | | | |
|-------------------------|---------|---------|--|
| Nome | Tipo | Tamanho | |
| a | float64 | (3, 3) | $\begin{bmatrix} 0. & 0. & 0. \\ 0. & 0. & 0. \\ 0. & 0. & 0. \end{bmatrix}$ |
| b | float64 | (3, 3) | $\begin{bmatrix} 1. & 0. & 0. \\ 0. & 1. & 0. \\ 0. & 0. & 1. \end{bmatrix}$ |

Fonte: *Print screen* da interface do Spyder 3.7

Fonte: Código desenvolvido pelo próprio autor

Observe que na linha 2, acima, o programa cria uma matriz 3x3 preenchida com zeros. Já na linha 3, o programa cria uma segunda matriz identidade de tamanho 3.

Figura 23 - Saída esperada no console



```
[[0. 0. 0.]  
[0. 0. 0.]  
[0. 0. 0.]  
[[1. 0. 0.]  
[0. 1. 0.]  
[0. 0. 1.]
```

Fonte: *Print screen* da interface do Spyder 3.7

Fonte: Código desenvolvido pelo próprio autor

Exemplo 3) Calculando o produto matricial entre as seguintes matrizes:

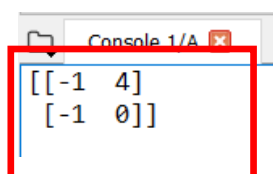
$$A = [1 \ 2 \ 1 \ 0], \text{ e } B = [-1 \ 0 \ 0 \ 2]$$

Python

```
1. import numpy as np  
2. a = np.array([ [1, 2], [1, 0] ])  
3. b = np.array([ [-1, 0], [0, 2] ])  
4. c = a.dot(b)  
5. print(c)
```

O produto entre as duas matrizes é calculado na linha 4, neste caso A por B, e o resultado é armazenado na variável C, a qual é um *array numpy*.

Figura 24 - Saída esperada no console



```
[[ -1  4]  
[ -1  0]]
```

Fonte: *Print screen* da interface do Spyder 3.7

Fonte: Código desenvolvido pelo próprio autor

IMPORTANTE: Caso desejar realizar uma multiplicação de B por A, basta alterar o comando do exemplo 3, linha 4, por `c = b.dot(a)`