

Notas de aula de Python

Curso livre de programação

Encontro 5 - Matrizes

Prof. Louis Augusto

`louis.augusto@ifsc.edu.br`



**INSTITUTO FEDERAL
SANTA CATARINA**

Instituto Federal de Santa Catarina
Campus Florianópolis

1 Matrizes Nativas

- Matriz é um vetor de vetores
- Percorrimento de matrizes
- Matrizes via compreensão de listas

2 Matrizes no numpy

- Conversão de matrizes nativas para numpy
- Criação de matrizes no numpy
- Operações matriciais com numpy

1 Matrizes Nativas

- **Matriz é um vetor de vetores**
- Percorrimento de matrizes
- Matrizes via compreensão de listas

2 Matrizes no numpy

- Conversão de matrizes nativas para numpy
- Criação de matrizes no numpy
- Operações matriciais com numpy

Matriz como vetor de vetores

As matrizes em Python são vistas como um vetor de vetores, logo tudo o que é válido para vetores permanece válido para matrizes.

```
# -*- coding: utf-8 -*-
#Criação de vetores
vA = [] # vetor vazio
vB = [int]* 5 # Vetor de inteiros de 5 posições
vC = [None]*10 # Vetor vazio de 10 posições
v9 = [9]*12 # Vetor completado com 9's em suas 12 posições

#Criação de matrizes
mA = [ None ] * 4 #Inicia a matriz como um vetor
for i in range (0 , 4): #Insere 4 linhas
    mA [ i ] = [ None ] * 3
    #Em cada linha insere 3 colunas.

#matriz 2x3 inserida diretamente.
mB = [[1,2,3],[ 'a', 'b', 'c' ]]
```

Matriz como vetor de vetores

As matrizes em Python são vistas como um vetor de vetores, logo tudo o que é válido para vetores permanece válido para matrizes.

```
# -*- coding: utf-8 -*-
#Criação de vetores
vA = [] # vetor vazio
vB = [int]* 5 # Vetor de inteiros de 5 posições
vC = [None]*10 # Vetor vazio de 10 posições
v9 = [9]*12 # Vetor completado com 9's em suas 12 posições

#Criação de matrizes
mA = [ None ] * 4 #Inicia a matriz como um vetor
for i in range (0 , 4): #Insere 4 linhas
    mA [ i ] = [ None ] * 3
    #Em cada linha insere 3 colunas.

#matriz 2x3 inserida diretamente.
mB = [[1,2,3],[ 'a', 'b', 'c' ]]
```

Matriz como vetor de vetores

As matrizes em Python são vistas como um vetor de vetores, logo tudo o que é válido para vetores permanece válido para matrizes.

```
# -*- coding: utf-8 -*-
#Criação de vetores
vA = [] # vetor vazio
vB = [int]* 5 # Vetor de inteiros de 5 posições
vC = [None]*10 # Vetor vazio de 10 posições
v9 = [9]*12 # Vetor completado com 9's em suas 12 posições

#Criação de matrizes
mA = [ None ] * 4 #Inicia a matriz como um vetor
for i in range (0 , 4): #Insere 4 linhas
    mA [ i ] = [ None ] * 3
    #Em cada linha insere 3 colunas.

#matriz 2x3 inserida diretamente.
mB = [[1,2,3],[ 'a', 'b', 'c' ]]
```

1 Matrizes Nativas

- Matriz é um vetor de vetores
- **Percorrimento de matrizes**
- Matrizes via compreensão de listas

2 Matrizes no numpy

- Conversão de matrizes nativas para numpy
- Criação de matrizes no numpy
- Operações matriciais com numpy

Percorrimento de matrizes

Exercício: Construa uma matriz 4x3 e a preencha as posições com números consecutivos de 1 a 12.

Complete a matriz por linhas e por colunas.

Completamento por linhas

| | | |
|----|----|----|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| 10 | 11 | 12 |

Completamento por colunas

| | | |
|---|---|----|
| 1 | 5 | 9 |
| 2 | 6 | 10 |
| 3 | 7 | 11 |
| 4 | 8 | 12 |

Alguns exercícios do Beecrowd com matrizes: 1182 a 1190.

Percorrimento de matrizes

Exercício: Construa uma matriz 4x3 e a preencha as posições com números consecutivos de 1 a 12.

Complete a matriz por linhas e por colunas.

Completamento por linhas

| | | |
|----|----|----|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| 10 | 11 | 12 |

Completamento por colunas

| | | |
|---|---|----|
| 1 | 5 | 9 |
| 2 | 6 | 10 |
| 3 | 7 | 11 |
| 4 | 8 | 12 |

Alguns exercícios do Beecrowd com matrizes: 1182 a 1190.

Percorrimento de matrizes

Exercício: Construa uma matriz 4x3 e a preencha as posições com números consecutivos de 1 a 12.

Complete a matriz por linhas e por colunas.

Completamento por linhas

| | | |
|----|----|----|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| 10 | 11 | 12 |

Completamento por colunas

| | | |
|---|---|----|
| 1 | 5 | 9 |
| 2 | 6 | 10 |
| 3 | 7 | 11 |
| 4 | 8 | 12 |

Alguns exercícios do Beecrowd com matrizes: 1182 a 1190.

Percorrimento de matrizes

Exercício: Construa uma matriz 4x3 e a preencha as posições com números consecutivos de 1 a 12.

Complete a matriz por linhas e por colunas.

Completamento por linhas

| | | |
|----|----|----|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| 10 | 11 | 12 |

Completamento por colunas

| | | |
|---|---|----|
| 1 | 5 | 9 |
| 2 | 6 | 10 |
| 3 | 7 | 11 |
| 4 | 8 | 12 |

Alguns exercícios do Beecrowd com matrizes: 1182 a 1190.

1 Matrizes Nativas

- Matriz é um vetor de vetores
- Percorrimento de matrizes
- Matrizes via compreensão de listas

2 Matrizes no numpy

- Conversão de matrizes nativas para numpy
- Criação de matrizes no numpy
- Operações matriciais com numpy

Matrizes via compreensão de listas

Podemos usar compreensão de listas para gerar matrizes também.

Vamos criar a matriz: $A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}$ usando compreensão de listas.

Basta-nos fazer:

```
A = [[i for i in range(1,6)] for j in range (5)]  
print(A)
```

E o que obtemos é uma repetição 5 vezes do vetor $[1, 2, 3, 4, 5]$ no interior de uma lista. Acessamos um valor da matriz como $A[1, 2]$, por exemplo.

1 Matrizes Nativas

- Matriz é um vetor de vetores
- Percorrimento de matrizes
- Matrizes via compreensão de listas

2 Matrizes no numpy

- Conversão de matrizes nativas para numpy
- Criação de matrizes no numpy
- Operações matriciais com numpy

Conversão de matrizes nativas

A maneira mais simples de transformar uma matriz nativa numa matriz numpy é por conversão direta:

Forma mais simples:

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

Podemos passar uma matriz nativa inteira para numpy do mesmo modo:
Considere a matriz nativa anteriormente criada:

```
for i in range(m):  
    for j in range(n):  
        mat[i][j] = j*m+(i+1)
```

Conversão para ndarray:

```
arr1 = np.array(mat1)
```

Observe que a impressão é exatamente igual:

```
for i in range (0 , len ( mat )):  
    for j in range (0 , len ( mat [ i ])):  
        print ( "{0:2d}".format(arr1[i][j]), end = ' ')  
print ()
```

Conversão de matrizes nativas

A maneira mais simples de transformar uma matriz nativa numa matriz numpy é por conversão direta:

Forma mais simples:

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

Podemos passar uma matriz nativa inteira para numpy do mesmo modo:
Considere a matriz nativa anteriormente criada:

```
for i in range(m):  
    for j in range(n):  
        mat[i][j] = j*m+(i+1)
```

Conversão para ndarray:

```
arr1 = np.array(mat1)
```

Observe que a impressão é exatamente igual:

```
for i in range (0 , len ( mat )):  
    for j in range (0 , len ( mat [ i ])):  
        print ( "{0:2d}".format(arr1[i][j]), end = ' ')  
print ()
```


Conversão de matrizes nativas

A maneira mais simples de transformar uma matriz nativa numa matriz numpy é por conversão direta:

Forma mais simples:

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

Podemos passar uma matriz nativa inteira para numpy do mesmo modo:

Considere a matriz nativa anteriormente criada:

```
for i in range(m):
    for j in range(n):
        mat[i][j] = j*m+(i+1)
```

Conversão para ndarray:

```
arr1 = np.array(mat1)
```

Observe que a impressão é exatamente igual:

```
for i in range (0 , len ( mat )):
    for j in range (0 , len ( mat [ i ])):
        print ( "{0:2d}".format(arr1[i][j]), end = ' ')
print ()
```

Conversão de matrizes nativas

A maneira mais simples de transformar uma matriz nativa numa matriz numpy é por conversão direta:

Forma mais simples:

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

Podemos passar uma matriz nativa inteira para numpy do mesmo modo:
Considere a matriz nativa anteriormente criada:

```
for i in range(m):  
    for j in range(n):  
        mat[i][j] = j*m+(i+1)
```

Conversão para ndarray:

```
arr1 = np.array(mat1)
```

Observe que a impressão é exatamente igual:

```
for i in range (0 , len ( mat )):  
    for j in range (0 , len ( mat [ i ])):  
        print ( "{0:2d}".format(arr1[i][j]), end = ' ')  
print ()
```

Conversão de matrizes nativas

A maneira mais simples de transformar uma matriz nativa numa matriz numpy é por conversão direta:

Forma mais simples:

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

Podemos passar uma matriz nativa inteira para numpy do mesmo modo:
Considere a matriz nativa anteriormente criada:

```
for i in range(m):  
    for j in range(n):  
        mat[i][j] = j*m+(i+1)
```

Conversão para ndarray:

```
arr1 = np.array(mat1)
```

Observe que a impressão é exatamente igual:

```
for i in range (0 , len ( mat )):  
    for j in range (0 , len ( mat [ i ])):  
        print ( "{0:2d}".format(arr1[i][j]), end = ' ')  
print ()
```

Conversão de matrizes nativas

A maneira mais simples de transformar uma matriz nativa numa matriz numpy é por conversão direta:

Forma mais simples:

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

Podemos passar uma matriz nativa inteira para numpy do mesmo modo:
Considere a matriz nativa anteriormente criada:

```
for i in range(m):  
    for j in range(n):  
        mat[i][j] = j*m+(i+1)
```

Conversão para ndarray:

```
arr1 = np.array(mat1)
```

Observe que a impressão é exatamente igual:

```
for i in range ( 0 , len ( mat ) ):  
    for j in range ( 0 , len ( mat [ i ] ) ):  
        print ( "{0:2d}".format(arr1[i][j]), end = ' ')  
print ( )
```

1 Matrizes Nativas

- Matriz é um vetor de vetores
- Percorrimento de matrizes
- Matrizes via compreensão de listas

2 Matrizes no numpy

- Conversão de matrizes nativas para numpy
- **Criação de matrizes no numpy**
- Operações matriciais com numpy

Criação de matrizes via numpy

1 Com arange:

```
arr2 = np.arange(1, 25).reshape(6, 4)
```

Cria-se um vetor com arange e depois altera a dimensão com o comando reshape.

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]
 [21 22 23 24]]
```

Gera este resultado.

2 Para saber a dimensão da matriz, usa-se o comando shape.

```
print("Dimensão da matriz: ", arr2.shape)
```

3 Matriz identidade: `matid3 = np.identity(3, dtype=float)`

4 Matriz somente com 1's: `array_2d = np.ones((2,3), dtype=int)`

OBS: O tipo, em zeros ou ones pode ser alterado para float, outro int de maior ou menor palavra.

Criação de matrizes via numpy

1 Com arange:

```
arr2 = np.arange(1, 25).reshape(6, 4)
```

Cria-se um vetor com arange e depois altera a dimensão com o comando reshape.

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]
 [21 22 23 24]]
```

Gera este resultado.

2 Para saber a dimensão da matriz, usa-se o comando shape.

```
print("Dimensão da matriz: ", arr2.shape)
```

3 Matriz identidade: `matid3 = np.identity(3, dtype=float)`

4 Matriz somente com 1's: `array_2d = np.ones((2,3), dtype=int)`

OBS: O tipo, em zeros ou ones pode ser alterado para float, outro int de maior ou menor palavra.

Criação de matrizes via numpy

1 Com arange:

```
arr2 = np.arange(1, 25).reshape(6, 4)
```

Cria-se um vetor com arange e depois altera a dimensão com o comando reshape.

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]
 [21 22 23 24]]
```

Gera este resultado.

2 Para saber a dimensão da matriz, usa-se o comando shape.

```
print("Dimensão da matriz: ", arr2.shape)
```

3 Matriz identidade: `matid3 = np.identity(3, dtype=float)`

4 Matriz somente com 1's: `array_2d = np.ones((2,3), dtype=int)`

OBS: O tipo, em zeros ou ones pode ser alterado para float, outro int de maior ou menor palavra.

Criação de matrizes via numpy

1 Com arange:

```
arr2 = np.arange(1, 25).reshape(6, 4)
```

Cria-se um vetor com arange e depois altera a dimensão com o comando reshape.

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]
 [21 22 23 24]]
```

Gera este resultado.

2 Para saber a dimensão da matriz, usa-se o comando shape.

```
print("Dimensão da matriz: ", arr2.shape)
```

3 Matriz identidade: `matid3 = np.identity(3, dtype=float)`

4 Matriz somente com 1's: `array_2d = np.ones((2,3), dtype=int)`

OBS: O tipo, em zeros ou ones pode ser alterado para float, outro int de maior ou menor palavra.

Criação de matrizes via numpy

1 Com arange:

```
arr2 = np.arange(1, 25).reshape(6, 4)
```

Cria-se um vetor com arange e depois altera a dimensão com o comando reshape.

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]
 [21 22 23 24]]
```

Gera este resultado.

2 Para saber a dimensão da matriz, usa-se o comando shape.

```
print("Dimensão da matriz: ", arr2.shape)
```

3 Matriz identidade: `matid3 = np.identity(3, dtype=float)`

4 Matriz somente com 1's: `array_2d = np.ones((2,3), dtype=int)`

OBS: O tipo, em zeros ou ones pode ser alterado para float, outro int de maior ou menor palavra.

Criação de matrizes via numpy

1 Com arange:

```
arr2 = np.arange(1, 25).reshape(6, 4)
```

Cria-se um vetor com arange e depois altera a dimensão com o comando reshape.

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]
 [21 22 23 24]]
```

Gera este resultado.

2 Para saber a dimensão da matriz, usa-se o comando shape.

```
print("Dimensão da matriz: ", arr2.shape)
```

3 Matriz identidade: `matid3 = np.identity(3, dtype=float)`

4 Matriz somente com 1's: `array_2d = np.ones((2,3), dtype=int)`

OBS: O tipo, em zeros ou ones pode ser alterado para float, outro int de maior ou menor palavra.

1 Matrizes Nativas

- Matriz é um vetor de vetores
- Percorrimento de matrizes
- Matrizes via compreensão de listas

2 Matrizes no numpy

- Conversão de matrizes nativas para numpy
- Criação de matrizes no numpy
- Operações matriciais com numpy

Operações Matriciais

O numpy opera ponto a ponto as posições da matriz. Deve-se ter cuidado portanto com:

```
Arr1 = np.random.rand(3,3) - gera uma matriz 3x3 com valores aleatórios entre 0 e 1.
```

```
matid3 = np.identity(3, dtype=float) - Matriz identidade 3x3.
```

Definindo `Arr4 = Arr1*matid3` vamos obter uma matriz com zeros fora da diagonal principal e com os valores da diagonal principal de `Arr1`. Ou seja, multiplicação ponto a ponto.

O mesmo ocorre com a função `multiply`, do numpy.

```
D = np.multiply(A,B)
```

Para obter multiplicação matricial usamos o comando `dot`(nativo), ou `matmul` (numpy):

```
import numpy as np
A = np.array([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = A.dot(B)
D = np.matmul(A,B)
print(C)
print(D)
```

Operações Matriciais

O numpy opera ponto a ponto as posições da matriz. Deve-se ter cuidado portanto com:

`Arr1 = np.random.rand(3, 3)` - gera uma matriz 3x3 com valores aleatórios entre 0 e 1.

`matid3 = np.identity(3, dtype=float)` - Matriz identidade 3x3.

Definindo `Arr4 = Arr1*matid3` vamos obter uma matriz com zeros fora da diagonal principal e com os valores da diagonal principal de `Arr1`. Ou seja, multiplicação ponto a ponto.

O mesmo ocorre com a função `multiply`, do numpy.

`D = np.multiply(A,B)`

Para obter multiplicação matricial usamos o comando `dot`(nativo), ou `matmul` (numpy):

```
import numpy as np
A = np.array([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = A.dot(B)
D = np.matmul(A,B)
print(C)
print(D)
```

Operações Matriciais

O numpy opera ponto a ponto as posições da matriz. Deve-se ter cuidado portanto com:

`Arr1 = np.random.rand(3, 3)` - gera uma matriz 3x3 com valores aleatórios entre 0 e 1.

`matid3 = np.identity(3, dtype=float)` - Matriz identidade 3x3.

Definindo `Arr4 = Arr1*matid3` vamos obter uma matriz com zeros fora da diagonal principal e com os valores da diagonal principal de `Arr1`. Ou seja, multiplicação ponto a ponto.

O mesmo ocorre com a função `multiply`, do numpy.

`D = np.multiply(A,B)`

Para obter multiplicação matricial usamos o comando `dot`(nativo), ou `matmul` (numpy):

```
import numpy as np
A = np.array([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = A.dot(B)
D = np.matmul(A,B)
print(C)
print(D)
```

Operações Matriciais

O numpy opera ponto a ponto as posições da matriz. Deve-se ter cuidado portanto com:

`Arr1 = np.random.rand(3, 3)` - gera uma matriz 3x3 com valores aleatórios entre 0 e 1.

`matid3 = np.identity(3, dtype=float)` - Matriz identidade 3x3.

Definindo `Arr4 = Arr1*matid3` vamos obter uma matriz com zeros fora da diagonal principal e com os valores da diagonal principal de `Arr1`. Ou seja, multiplicação ponto a ponto.

O mesmo ocorre com a função `multiply`, do numpy.

`D = np.multiply(A,B)`

Para obter multiplicação matricial usamos o comando `dot`(nativo), ou `matmul` (numpy):

```
import numpy as np
A = np.array([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = A.dot(B)
D = np.matmul(A,B)
print(C)
print(D)
```


Operações Matriciais

O numpy opera ponto a ponto as posições da matriz. Deve-se ter cuidado portanto com:

`Arr1 = np.random.rand(3, 3)` - gera uma matriz 3x3 com valores aleatórios entre 0 e 1.

`matid3 = np.identity(3, dtype=float)` - Matriz identidade 3x3.

Definindo `Arr4 = Arr1*matid3` vamos obter uma matriz com zeros fora da diagonal principal e com os valores da diagonal principal de `Arr1`. Ou seja, multiplicação ponto a ponto.

O mesmo ocorre com a função `multiply`, do numpy.

```
D = np.multiply(A,B)
```

Para obter multiplicação matricial usamos o comando `dot`(nativo), ou `matmul` (numpy):

```
import numpy as np
A = np.array([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = A.dot(B)
D = np.matmul(A,B)
print(C)
print(D)
```

Operações Matriciais

O numpy opera ponto a ponto as posições da matriz. Deve-se ter cuidado portanto com:

`Arr1 = np.random.rand(3,3)` - gera uma matriz 3x3 com valores aleatórios entre 0 e 1.

`matid3 = np.identity(3, dtype=float)` - Matriz identidade 3x3.

Definindo `Arr4 = Arr1*matid3` vamos obter uma matriz com zeros fora da diagonal principal e com os valores da diagonal principal de `Arr1`. Ou seja, multiplicação ponto a ponto.

O mesmo ocorre com a função `multiply`, do numpy.

`D = np.multiply(A,B)`

Para obter multiplicação matricial usamos o comando `dot(nativo)`, ou `matmul` (numpy):

```
import numpy as np
A = np.array([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = A.dot(B)
D = np.matmul(A,B)
print(C)
print(D)
```

Operações Matriciais

O numpy opera ponto a ponto as posições da matriz. Deve-se ter cuidado portanto com:

`Arr1 = np.random.rand(3,3)` - gera uma matriz 3x3 com valores aleatórios entre 0 e 1.

`matid3 = np.identity(3, dtype=float)` - Matriz identidade 3x3.

Definindo `Arr4 = Arr1*matid3` vamos obter uma matriz com zeros fora da diagonal principal e com os valores da diagonal principal de `Arr1`. Ou seja, multiplicação ponto a ponto.

O mesmo ocorre com a função `multiply`, do numpy.

`D = np.multiply(A,B)`

Para obter multiplicação matricial usamos o comando `dot`(nativo), ou `matmul` (numpy):

```
import numpy as np
A = np.array([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = A.dot(B)
D = np.matmul(A,B)
print(C)
print(D)
```

Operações Matriciais

O numpy opera ponto a ponto as posições da matriz. Deve-se ter cuidado portanto com:

```
Arr1 = np.random.rand(3,3) - gera uma matriz 3x3 com valores aleatórios entre 0 e 1.
```

```
matid3 = np.identity(3, dtype=float) - Matriz identidade 3x3.
```

Definindo `Arr4 = Arr1*matid3` vamos obter uma matriz com zeros fora da diagonal principal e com os valores da diagonal principal de `Arr1`. Ou seja, multiplicação ponto a ponto.

O mesmo ocorre com a função `multiply`, do numpy.

```
D = np.multiply(A,B)
```

Para obter multiplicação matricial usamos o comando `dot`(nativo), ou `matmul` (numpy):

```
import numpy as np
A = np.array([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = A.dot(B)
D = np.matmul(A,B)
print(C)
print(D)
```

Operações Matriciais

As mesmas operações de corte em vetores permanecem válidas para matrizes.

```
import numpy as np
A = np.array([[1, 4, 5, 12, 14],
              [-5, 8, 9, 0, 17],
              [-6, 7, 11, 19, 21]])
```

Observe diferentes saídas para a matriz: `print(A[:2, :3])`: as duas primeiras linhas e três primeiras colunas.

`print(A[1])`: primeira linha.

`print(A[:, 2])`: segunda coluna. **Cuidado, a impressão é feita como linha, mas é uma coluna.**

Operações Matriciais

As mesmas operações de corte em vetores permanecem válidas para matrizes.

```
import numpy as np
A = np.array([[1, 4, 5, 12, 14],
              [-5, 8, 9, 0, 17],
              [-6, 7, 11, 19, 21]])
```

Observe diferentes saídas para a matriz: `print(A[:2, :3])`: as duas primeiras linhas e três primeiras colunas.

`print(A[1])`: primeira linha.

`print(A[:, 2])`: segunda coluna. **Cuidado, a impressão é feita como linha, mas é uma coluna.**

Operações Matriciais

As mesmas operações de corte em vetores permanecem válidas para matrizes.

```
import numpy as np
A = np.array([[1, 4, 5, 12, 14],
              [-5, 8, 9, 0, 17],
              [-6, 7, 11, 19, 21]])
```

Observe diferentes saídas para a matriz: `print(A[:2, :3])`: as duas primeiras linhas e três primeiras colunas.

`print(A[1])`: primeira linha.

`print(A[:, 2])`: segunda coluna. Cuidado, a impressão é feita como linha, mas é uma coluna.

Operações Matriciais

As mesmas operações de corte em vetores permanecem válidas para matrizes.

```
import numpy as np
A = np.array([[1, 4, 5, 12, 14],
              [-5, 8, 9, 0, 17],
              [-6, 7, 11, 19, 21]])
```

Observe diferentes saídas para a matriz: `print(A[:2, :3])`: as duas primeiras linhas e três primeiras colunas.

`print(A[1])`: primeira linha.

`print(A[:,2])`: segunda coluna. Cuidado, a impressão é feita como linha, mas é uma coluna.

Operações Matriciais

As mesmas operações de corte em vetores permanecem válidas para matrizes.

```
import numpy as np
A = np.array([[1, 4, 5, 12, 14],
              [-5, 8, 9, 0, 17],
              [-6, 7, 11, 19, 21]])
```

Observe diferentes saídas para a matriz: `print(A[:2, :3])`: as duas primeiras linhas e três primeiras colunas.

`print(A[1])`: primeira linha.

`print(A[:, 2])`: segunda coluna. **Cuidado, a impressão é feita como linha, mas é uma coluna.**

Operações Matriciais




As mesmas operações de corte em vetores permanecem válidas para matrizes.




```
import numpy as np
A = np.array([[1, 4, 5, 12, 14],
              [-5, 8, 9, 0, 17],
              [-6, 7, 11, 19, 21]])
```




Observe diferentes saídas para a matriz: `print(A[:2, :3])`: as duas primeiras linhas e três primeiras colunas.

`print(A[1])`: primeira linha.

`print(A[:, 2])`: segunda coluna. **Cuidado, a impressão é feita como linha, mas é uma coluna.**

-  Flávio Ulhoa Coelho e Mary Lilian Lourenço,
Um curso de Álgebra Linear
LedUsp - São Paulo.
Editora da Universidade de São Paulo.
-  Isabel Cabral, Cecília Perdigão e Carlos Saiago,
Álgebra Linear, 2ª edição.
Escolar Editora, Lisboa, 2010.
-  Seymour Lipschutz e Marc Lipson,
Álgebra Linear
Coleção Schaum, 3ª edição.
Editora Bookman, Porto Alegre 2004.

-  Flávio Ulhoa Coelho e Mary Lilian Lourenço,
Um curso de Álgebra Linear
LedUsp - São Paulo.
Editora da Universidade de São Paulo.
-  Isabel Cabral, Cecília Perdigão e Carlos Saiago,
Álgebra Linear, 2ª edição.
Escolar Editora, Lisboa, 2010.
-  Seymour Lipschutz e Marc Lipson,
Álgebra Linear
Coleção Schaum, 3ª edição.
Editora Bookman, Porto Alegre 2004.

-  Flavio Ulhoa Coelho e Mary Lilian Lourenço,
Um curso de Álgebra Linear
LedUsp - São Paulo.
Editora da Universidade de São Paulo.
-  Isabel Cabral, Cecília Perdigão e Carlos Saiago,
Álgebra Linear, 2ª edição.
Escolar Editora, Lisboa, 2010.
-  Seymour Lipschutz e Marc Lipson,
Álgebra Linear
Coleção Schaum, 3ª edição.
Editora Bookman, Porto Alegre 2004.