TECHMIMO

Autor: Rafael Pereira da Silva

Seguem alguns recados para ajudá-los e para contribuir com o curso:

- Fiquem à vontade para me contatar pelo Linkdin, costumo responder por lá também: https://www.linkedin.com/in/rafael-pereira-da-silva-23890799/ (https://www.linkedin.com/in/rafael-pereira-da-silva-23890799/)
- Fiquem a vontade para compartilharem o certificado do curso no Linkedin. Eu costumo curtir e comentar para dar mais credibilidade
- Vocês podem usar esses notebooks para resolver os exercícios e desafios
- Não se esqueçam de avaliar o curso e dar feedback, eu costumo criar conteúdos baseado nas demandas de vocês
- Se tiverem gostando do curso, recomendem aos amigos, pois isso também ajuda a impulsionar e a crescer a comunidade
- Bons estudos e grande abraços!

Seção 8 - Fundamentos de Numpy

8.1 Arrays

Acesse:

- https://numpy.org/doc/ (https://numpy.org/doc/)
- https://numpy.org/doc/1.18/numpy-user.pdf (https://numpy.org/doc/1.18/numpy-user.pdf)
- https://scipy-lectures.org/intro/numpy/array_object.html#indexing-and-slicing (https://scipy-lectures.org/intro/numpy/array_object.html#indexing-and-slicing)

Sobre

O Numpy é uma biblioteca que permite construirmos *arrays* multidimensionais.

Propriedades de arrays

Propriedade	Retorna	Descrição
ndim	int	número de dimensões (axes)
shape	tupla	quantidade de dados em cada dimensão
size	int	número de elementos
dtype	dtype	tipo das variáveis dos elementos

In [26]:

```
import numpy as np

a_1 = np.array([1,1,1])
a_1.ndim
```

Out[26]:

```
1
```

In [31]:

```
a_2 = np.array([[3,2.,3],[4,5,6]])
a_2
```

Out[31]:

```
array([[ 3.,  2.,  3.],
       [ 4.,  5.,  6.]])
```

In [36]:

```
a_4 = np.arange(40).reshape(2,2,2,5)
a_4.dtype
```

Out[36]:

```
dtype('int32')
```

8.2 Operações com arrays

Algumas funcionalidades do Numpy

Comando	Tipo	Descrição
sin,cos,exp,sqrt	Função	seno,cosseno,exponencial,raíz quadrada
pi,nan,inf	constantes(float)	pi, similar a None, infinito
+,-,*,/	operadores	operações aritimética
ARRAY1.dot(ARRAY2)	método	multiplicação matricial

In [7]:

```
import numpy as np

a = np.sin(np.pi/2)
a
```

Out[7]:

```
1.0
```

In [6]:

```
type(np.pi)
```

Out[6]:

```
float
```

In [9]:

```
type(np.inf)
```

Out[9]:

```
float
```

In [12]:

```
type(np.nan)
```

Out[12]:

```
float
```

In [11]:

```
None
```

In [20]:

```
a_1 = np.array([[1,2,3]])
a_1
```

Out[20]:

```
array([[1, 2, 3]])
```

In [22]:

```
a_2 = np.array([4,5,6])
a_2
```

Out[22]:

```
array([4, 5, 6])
```

In [17]:

```
a_1 - 5
```

Out[17]:

```
array([-4, -3, -2])
```

In [21]:

```
a_1*a_2
```

Out[21]:

```
array([[ 4, 10, 18]])
```

In [24]:

```
a_1.dot(a_2)
```

Out[24]:

```
array([32])
```

In [25]:

```
import sympy as sp

s_1 = sp.Matrix([[1,2,3]])
s_1
```

Out[25]:

```
Matrix([[1, 2, 3]])
```

In [27]:

```
s_2 = sp.Matrix([4,5,6])
s_2
```

Out[27]:

```
Matrix([
[4],
[5],
[6]])
```

In [28]:

```
s_1*s_2
```

Out[28]:

```
Matrix([[32]])
```

In [ ]:

In [ ]:

In [ ]:

8.3 Funções de criação de arrays

Algumas funções numpy para criar arrays

Função	Descrição
zeros(*shape*)	array preenchido com zeros
ones(*shape*)	array preenchido com uns

Função	Descrição
eye(*dimensão*)	matriz identidade
arange(*início, fim, passo*)	cria um array unidimensioanl
linspace(*início, fim, quantidade*)	cria um array unidimensioanl
vstack([*arrays*]) e hstack([*arrays*])	adiciona elementos de um ou mais arrays

Métodos para dimensionar os arrays:

- reshape --> retorna um array com o shape indicado
- resize --> modifica o shape do array em que está sendo aplicado

In [88]:

```
import numpy as np

a_4 = np.ones(6)*11
a_4
```

Out[88]:

```
array([ 11.,  11.,  11.,  11.,  11.,  11.])
```

In [91]:

```
a_5 = np.eye(4)
a_5.ndim
```

Out[91]:

```
2
```

In [96]:

```
a_6 = np.arange(0,10,0.5)
a_6
```

Out[96]:

```
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5,  5. ,
        5.5,  6. ,  6.5,  7. ,  7.5,  8. ,  8.5,  9. ,  9.5])
```

In [98]:

```
a_7 = np.linspace(0,10,12)
a_7.size
```

Out[98]:

```
12
```

In [106]:

```
a_8 = np.ones(8)
a_8.reshape(2,4)
a_8
```

Out[106]:

```
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

In [107]:

```
a_9 = np.ones(8)
a_9.resize(2,4)
a_9

a_10 = np.zeros(8)
a_10.resize(2,4)
a_10
```

Out[107]:

```
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
```

In [108]:

```
a_11 = np.vstack((a_9,a_10))
a_11
```

Out[108]:

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

In [116]:

```
a_12 = np.hstack((a_9,a_10))
a_13 = np.vstack((a_11,a_9))
a_13
```

Out[116]:

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

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
```

8.4 Método copy

Método	Descrição
.copy()	faz a cópia (deep copy) do array. Cuidado! O sinal de igual não executa essa função

Nota - a propriedade **base** retorna o array utilizado para criar o array em questão

In [2]:

```
import numpy as np

a = np.arange(8)
a
```

Out[2]:

```
array([0, 1, 2, 3, 4, 5, 6, 7])
```

In [3]:

```
b = a
b
```

Out[3]:

```
array([0, 1, 2, 3, 4, 5, 6, 7])
```

In [4]:

```
a[0] = 9
a
```

Out[4]:

```
array([9, 1, 2, 3, 4, 5, 6, 7])
```

In [6]:

```
a is b
```

Out[6]:

```
True
```

In [7]:

```
c = a.copy()
c
```

Out[7]:

```
array([9, 1, 2, 3, 4, 5, 6, 7])
```

In [8]:

```
a[0] = 1
a
```

Out[8]:

```
array([1, 1, 2, 3, 4, 5, 6, 7])
```

In [9]:

```
b
```

Out[9]:

```
array([1, 1, 2, 3, 4, 5, 6, 7])
```

In [12]:

```
c is b
```

Out[12]:

```
False
```

In [15]:

```
d = a.reshape(2,4)
d.base is a
```

Out[15]:

```
True
```

In [17]:

```
d.base is a
```

Out[17]:

```
True
```

In [19]:

```
print(a.base)
```

```
None
```

In [ ]:

```

```

8.5 Métodos matemáticos para arrays

Alguns métodos matemáticos para arrays

Método	Descrição
.max() / .min()	retona os valores máximo e mínimo
.argmax() / .argmin()	retorna os índices de valor máximo e mínimo

Método	Descrição
.sum() / .mean() / .std()	soma, média e desvio padrão
.cumsum()	retorna um array com a soma acumulada

In [29]:

```
import numpy as np

my_array = np.arange(8).reshape(2,4)**2
my_array[0][0] =11
my_array
```

Out[29]:

```
array([[11,  1,  4,  9],
       [16, 25, 36, 49]], dtype=int32)
```

In [33]:

```
my_array.min(axis=1)
```

Out[33]:

```
array([ 1, 16], dtype=int32)
```

In [35]:

```
my_array.argmin()
```

Out[35]:

```
1
```

In [40]:

```
my_array.sum()
```

Out[40]:

```
151
```

In [39]:

```
my_array.cumsum()
```

Out[39]:

```
array([ 11,  12,  16,  25,  41,  66, 102, 151], dtype=int32)
```

8.6 Índices e fatias de arrays (Indexing and slicing)

Guidelines

https://scipy-lectures.org/intro/numpy/array_object.html#indexing-and-slicing (https://scipy-lectures.org/intro/numpy/array_object.html#indexing-and-slicing)

Sintaxe	Descrição

Sintaxe	Descrição
[i]	para 1 dimensão funciona como nas listas
[:i]	exibe um array que vai do índice 0 até o anterior ao i
[i:]	exibe um array que vai do índice posterior ao i até o último índice
[::i]	exibe todos os elementos variando de i em i
[::-i]	exibe a sequência de elementos de trás pra frente variando de i em i
[a:b:c]	início, fim, step
[i,j]	para exibir um elemento de uma matriz
[i,a:b]	retorna uma fatia da linha i com elementos das colunas de a até b
[n,i,j]	outra dimensão, linha, coluna

Quando criamos um subarray o numpy pode compartilhar memória entre arrays (como mostrado em aulas anteriores com a propriedade .base). Para descobrir se um array compartilha memória com outro use: **np.may_share_memory(a, c)**

In [46]:

```
import numpy as np

a = np.arange(10)**2
a
```

Out[46]:

```
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81], dtype=int32)
```

In [54]:

```
a[::-2]
```

Out[54]:

```
array([81, 49, 25,  9,  1], dtype=int32)
```

In [56]:

```
a[2:8:2]
```

Out[56]:

```
array([ 4, 16, 36], dtype=int32)
```

In [57]:

```
b = np.arange(20).reshape(4,5)
b
```

Out[57]:

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

In [61]:

```
b[1,1]
```

Out[61]:

```
6
```

In [64]:

```
b[2:,3:]
```

Out[64]:

```
array([[13, 14],
       [18, 19]])
```

In [66]:

```
c = np.arange(40).reshape(2,4,5)
c[1,1,1]
```

Out[66]:

```
26
```

In [67]:

```
b
```

Out[67]:

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

In [69]:

```
d = b[2:,3:]
d
```

Out[69]:

```
array([[13, 14],
       [18, 19]])
```

In [73]:

```
d.base is b.base
```

Out[73]:

```
True
```

In [74]:

```
np.may_share_memory(b, d)
```

Out[74]:

```
True
```

In [75]:

```
e = b[2:,3:].copy()
e
```

Out[75]:

```
array([[13, 14],
       [18, 19]])
```

In [76]:

```
e is d
```

Out[76]:

```
False
```

In [77]:

```
np.may_share_memory(e, d)
```

Out[77]:

```
False
```

8.7 Operações com vetores e matrizes no Numpy

- https://numpy.org/doc/stable/reference/routines.linalg.html (https://numpy.org/doc/stable/reference/routines.linalg.html)
- https://numpy.org/doc/stable/reference/generated/numpy.cross.html (https://numpy.org/doc/stable/reference/generated/numpy.cross.html)

Método	Descrição
.dot(a,b) ou @	retorna o produto escalar entre os vetores **a** e **b**
.cross(a,b)	retorna o produto vetorial entre os vetores **a** e **b**
.inner(a,b) e .outer(a,b)	retorna o produto interno e externo entre os vetores **a** e **b**
.linalg.norm(a)	retorna a norma do vetor **a**
.matmul(a,b)	produto entre as matrizes **a** e **b**
.linalg.det(A)	retorna o determinante da matriz **A**
.linalg.inv(A)	retorna a matriz inversa de **A**

Vetores

In [1]:

```
import numpy as np
```

In [13]:

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


a@a
np.dot(a,a)
```

Out[13]:

```
14
```

In [30]:

```
np.inner(a,a)
```

Out[30]:

```
14
```

In [31]:

```
np.outer(a,a)
```

Out[31]:

```
array([[1, 2, 3],
       [2, 4, 6],
       [3, 6, 9]])
```

In [50]:

```
np.linalg.norm(a)
#np.sqrt(1**2+2**2+3**2)
```

Out[50]:

```
3.7416573867739413
```

In [35]:

```
# produto vetorial
```

In [37]:

```
v1 = np.array([1,0,0])
v2 = np.array([0,1,0])

np.cross(v1,v2)
```

Out[37]:

```
array([0, 0, 1])
```

In [33]:

```
import sympy as sp

a_s = sp.Matrix([1,2,3])


a_s.T*a_s
```

Out[33]:

$$\begin{bmatrix} 14 \end{bmatrix}$$

In [34]:

```
a_s*a_s.T
```

Out[34]:

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}$$

Matrizes

In [54]:

```
#multiplicação numpy

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

np.matmul(A,A)

A@A # (cuidado com o A*A)
```

Out[54]:

```
array([[ 9, 10],
       [20, 24]])
```

In [55]:

```
A_s = sp.Matrix([[1,2],[4,4]])

A_s*A_s
```

Out[55]:

$$\begin{bmatrix} 9 & 10 \\ 20 & 24 \end{bmatrix}$$

In [56]:

```
np.linalg.det(A)
```

Out[56]:

```
-4.0
```

In [57]:

```
np.linalg.inv(A)
```

Out[57]:

```
array([[-1.  ,  0.5 ],
       [ 1.  , -0.25]])
```

In [58]:

```
np.linalg.inv(A)@A
```

Out[58]:

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

8.8 Autovalor e autovetor

- https://numpy.org/doc/stable/reference/routines.linalg.html (https://numpy.org/doc/stable/reference/routines.linalg.html)

Método	Descrição
linalg.eig(A)	retorna os autovalores e autovetores da matrix **A**

Sympy

In [80]:

```
import sympy as sp

lamb = sp.symbols('lambda')
A_s = sp.Matrix([[5.,3,-1],
                 [3,4,0],
                 [-1,0,2]])

eq_carac = sp.simplify(((A - lamb*sp.eye(3))))
eq_carac.det()
```

Out[80]:

$$10\lambda + (2 - \lambda)(4 - \lambda)(5 - \lambda) - 22$$

In [79]:

```
sp.solve(eq_carac.det())
```

Out[79]:

```
[1, 5 - sqrt(7), sqrt(7) + 5]
```

Numpy

In [59]:

```
import numpy as np

A = np.array([[5,3,-1],
              [3,4,0],
              [-1,0,2]])
```

In [64]:

```
l,v = np.linalg.eig(A)
l
```

Out[64]:

```
array([7.64575131, 1.        , 2.35424869])
```

8.9 Sistemas lineares

- https://numpy.org/doc/stable/reference/routines.linalg.html (https://numpy.org/doc/stable/reference/routines.linalg.html)

Método	Descrição
.linalg.solve(A, B)	resolve um sistema do tipo AX= B

In [81]:

```
import numpy as np


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

B = np.array([6,9,11])
```

In [82]:

```
np.linalg.solve(A,B)
```

Out[82]:

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

In [ ]:

```

```