# Aula 7 - Numpy

Karl Jan Clinckspoor

2 de julho de 2018

## Sumário



## 1 Introdução

Numpy é um pacote para operações numéricas de alta eficiência. Uma enorme parte dos pacotes científicos no Python são construídos com base no numpy, então é bom conhecer um pouco sobre como funciona. Ao invés de se utilizar listas com números e loops for para operar sobre essas listas, é possível utilizar *numpy arrays* para realizar o mesmo mais facilmente, e mais rapidamente. Porém, tudo tem seu lado negativo. Numpy arrays não podem conter dados de tipos diferentes, então todos os itens numa array devem, necessariamente, ser, por exemplo, floats.

## 2 Uso

É comum importar numpy e abreviá-lo como **np**.

```
In [2]: import numpy as np
```

Caso você deseje procurar algo no numpy pelo nome, é possível utilizar o comando **lookfor**.

```
In [53]: np.lookfor('singular value decomposition')

Search results for 'singular value decomposition'
-------------------------------------------------
numpy.linalg.svd
    Singular Value Decomposition.
```

```
numpy.polyfit
    Least squares polynomial fit.
numpy.ma.polyfit
    Least squares polynomial fit.
numpy.linalg.pinv
    Compute the (Moore-Penrose) pseudo-inverse of a matrix.
numpy.polynomial.Hermite.fit
    Least squares fit to data.
numpy.polynomial.Hermite._fit
    Least squares fit of Hermite series to data.
numpy.polynomial.HermiteE.fit
    Least squares fit to data.
numpy.polynomial.Laguerre.fit
    Least squares fit to data.
numpy.polynomial.Legendre.fit
    Least squares fit to data.
numpy.polynomial.Chebyshev.fit
    Least squares fit to data.
numpy.polynomial.HermiteE._fit
    Least squares fit of Hermite series to data.
numpy.polynomial.Laguerre._fit
    Least squares fit of Laguerre series to data.
numpy.polynomial.Legendre._fit
    Least squares fit of Legendre series to data.
numpy.polynomial.Chebyshev._fit
    Least squares fit of Chebyshev series to data.
numpy.polynomial.Polynomial.fit
    Least squares fit to data.
numpy.polynomial.Polynomial._fit
    Least-squares fit of a polynomial to data.
```

## 2.1 *Arrays* e *linspace*

Linspace cria um vetor numpy igualmente espaçado entre os pontos iniciais e finais (neste caso, ambos são inclusos). O número padrão de pontos nesse intervalo é 50.

```
In [16]: x = np.linspace(0, 10, 15)
         print(type(x))
         print(x)
```

```
<class 'numpy.ndarray'>
[ 0.          0.71428571  1.42857143  2.14285714  2.85714286  3.57142857
  4.28571429  5.          5.71428571  6.42857143  7.14285714  7.85714286
  8.57142857  9.28571429 10.        ]
```

Ao invés de ter que agir sobre todos os termos individualmente de uma array, podemos aplicar uma operação no array como se ela fosse um número qualquer, que a operação é automaticamente aplicada para todos os elementos.

```
In [22]: y1 = x * 2
         y2 = np.sin(x)
         y3 = x ** 3 - 10 * x + 5

         print(f'Curva senoidal:\n {y2}\n\nParábola de mínimo:\n{y3}')
```

```
Curva senoidal:
 [ 0.          0.6550779   0.98990308  0.84078711  0.2806294  -0.41672165
 -0.91034694 -0.95892427 -0.53870529  0.14487449  0.75762842  0.999995
  0.75348673  0.13861589 -0.54402111]

Parábola de mínimo:
[ 5.00000000e+00 -1.77842566e+00 -6.37026239e+00 -6.58892128e+00
 -2.47813411e-01  1.48396501e+01  4.08600583e+01  8.00000000e+01
  1.34446064e+02  2.06384840e+02  2.98002915e+02  4.11486880e+02
  5.49023324e+02  7.12798834e+02  9.05000000e+02]
```

## 2.2 Mínimos e máximos

Conseguimos observar, pelos números, que de fato as operações foram aplicadas com sucesso. A curva senoidal varia de -1 a 1 e que há aparentemente um valor mínimo na parábola. Para encontrar o valor de mínimo, e seu índice podemos utilizar outras funções, **min** e **argmin**. Encontrar o índice do valor é útil quando quisermos correlacionar esse valor de mínimo com outros pontos de outras funções, se todas possuírem o mesmo $x$. O máximo e índice do máximo podem ser encontradas pelas funções análogas.

```
In [24]: y3_min = y3.min()
         y3_argmin = y3.argmin()

         print(y3_min, y3[y3_argmin], y2[y3_argmin])
```

```
-6.588921282798832 -6.588921282798832 0.8407871057952504
```

Vemos por isso que os valores de mínimo encontrado por função e indexação são iguais, e que no ponto de mínimo da parábola, a função senoidal está perto de seu máximo (por coincidência).

## 2.3 *logspace* e *log10*

Cria um array com os números espaçados igualmente na escala logarítmica (base 10). Útil para equações que são geralmente visualizadas na escala logarítmica. Os argumentos são valores das potências de 10 dos limites.

A função *log10* é utilizada para aplicar o logaritmo de base 10 num array.

```
In [28]: x2 = np.logspace(-1, 1)

         # Equivalente a utilizar logspace
         x3 = np.linspace(-1, 1)
```

```
        x4 = 10 ** x3

        print(x2 - x4)  # Se forem iguais, tudo será zero.
```

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

## 2.4 Criação de arrays a partir de listas, dimensões maiores

É possível criar arrays fornecendo uma lista, ou outro iterável compatível, para a função *array*. Se for fornecida uma lista com listas, será criada uma array bidimensional.

```
In [30]: lista1 = [1, 2, 3, 4, 5]
         array1 = np.array(lista1)

         lista2d = [[1, 2, 3, 4], [5, 6, 7, 8]]
         array2d = np.array(lista2d)

         print(f'{array1}\n\n{array2d}')
```

```
[1 2 3 4 5]

[[1 2 3 4]
 [5 6 7 8]]
```

Para acessar um elemento de uma matriz 2d, é possível utilizar a mesma notação que uma lista aninhada, mas também é possível utilizar uma notação mais próxima à matemática, com (**linha**, **coluna**), lembrando que a indexação começa do zero.

```
In [32]: item1 = array2d[0][1]
         item2 = array2d[0, 1]
         print(f'São iguais: {item1 == item2}')
```

```
São iguais: True
```

*Slicing* funciona de uma maneira semelhante à vista anteriormente, onde os slices serão, então, separados por vírgulas. É normal achar que a notação é confusa, não se preocupe.

```
In [33]: meio = array2d[0:2, 1:3]
         meio
```

```
Out[33]: array([[2, 3],
                [6, 7]])
```

## 2.5 *random*

Random é um pacote dentro do numpy. O papel das funções random é providenciar uma maneira de se escolher um valor de uma lista aleatoriamente, ou de gerar valores aleatórios. Rode a seguinte cela várias vezes para ver os valores mudando.

```
In [45]: r1 = np.random.random()  # Cria um número aleatório entre 0 e 1
         r2 = np.random.choice(array1)  # Escolhe um valor aleatório da lista
         r3 = np.random.randint(5)  # Escolhe um número inteiro entre 0 e 5

         print(f'{r1:.2f}; {r2:.2f}; {r3:.2f}')
```

```
0.29; 5.00; 4.00
```

## 2.6 Constantes

O numpy vem com as constantes mais famosas já inclusas, então não é necessário escrevê-las sempre.

```
In [46]: print(f'Pi:{np.pi}, e:{np.e}')
```

```
Pi:3.141592653589793, e:2.718281828459045
```

## 2.7 Multiplicação de arrays

Numpy pode também ser utilizado para álgebra linear. Veja os seguintes exemplos de multiplicação de matrizes.

```
In [108]: c = np.array([[1, 2], [6, 7]])
          d = np.array([[3, 1], [2, 1]])

          print('c:\n', c,'\n\nd:\n', d, '\n')

           # Multiplicação de termo a termo
          print('Mult. termo a termo\n', c * d, '\n')

          # Multiplicação de matriz caso as dimensões sejam maiores. Senão faz multiplicação esc
          print('Mult. matriz c . d\n', np.dot(c, d), '\n')
          print('Mult. matriz c . d\n', c @ d, '\n')  # Exclusivo no Python 3.5+

          # Para multiplicar matricialmente dois vetores, é necessário transformá-los em matrize
          print('Mult. matriz de dois vetores')
          a = np.array([1, 2, 3, 4])
          b = np.array([2, 2, 2, 2])
          print(np.dot(a[:,None], b[None,:]))
```

```
c:
 [[1 2]
 [6 7]]

d:
 [[3 1]
 [2 1]]

Mult. termo a termo
 [[ 3  2]
 [12  7]]

Mult. matriz c . d
 [[ 7  3]
 [32 13]]

Mult. matriz c . d
 [[ 7  3]
 [32 13]]

Mult. matriz de dois vetores
[[2 2 2 2]
 [4 4 4 4]
 [6 6 6 6]
 [8 8 8 8]]
```

Numpy possui mais uma enorme quantidade de funções e pacotes para as mais diversas funções. Ele é capaz também de adaptar código de C/C++/Fortran para Python, para aumentar a eficiência de suas funções. Veja o site deles para mais informações, aqui. Veja também as diferenças entre Numpy e Matlab aqui.