Computação II

Departamento de Ciência da Computação – UFRJ

NumPy, SciPy e MatplotLib – Parte 1

Professora: *Fernanda Duarte Vilela Reis de Oliveira*

2019/01

1. Bibliotecas para Computação Científica

- Numpy – permite a criação e manipulação de vetores e matrizes com N dimensões. Trabalha com uma grande quantidade de dados de forma muito eficiente.
- Scipy – complemento para o numpy, possui bibliotecas mais completas para álgebra linear e processamento de dados;
- Matplotlib – permite a plotagem de gráficos e histogramas dados vetores de pontos.
- Não são instaladas automaticamente com o Python:

python –m pip install numpy

python –m pip install scipy

python –m pip install matplotlib

2. NumPy

- Representações de matrizes e vetores;
- Fornece um objeto array – vetor multidimensional de alta performance;
- Operações entre matrizes;
- Biblioteca com funções de álgebra linear.

2. NumPy

- Tipos de dados importantes fornecidos pelo numpy:

1. Array (ndarray): grid de elementos de um mesmo tipo indexado utilizando números inteiros separados por vírgulas.

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> a[0]
1
>>> a = np.array([[1,2],[3,4]])
>>> a[0,1]
2
```

2. Matrix (não é mais recomentado – será descontinuado): matriz de elementos de um mesmo tipo – necessariamente possui dimensão dois. Feito originalmente para operações de álgebra linear.

2. NumPy

- Tipos de dados importantes fornecidos pelo numpy:

3. Data type objects (dtype): objeto que descreve como os itens de um array devem ser interpretados. Utilizando um data type é possível criar arrays com diferentes tipos de dados. Dados são acessados através do nome dado na criação do dtype.

```
>>> import numpy as np
>>> dt = np.dtype([('nome',str,64),('idade',int),('peso',float)])
>>> dt
dtype([('nome', '<U64'), ('idade', '<i4'), ('peso', '<f8')])
>>> dados = np.array([('Ana',25,58.3),('Pedro',20,70)],dtype=dt)
>>> dados
array([('Ana', 25, 58.3), ('Pedro', 20, 70. )],
      dtype=[('nome', '<U64'), ('idade', '<i4'), ('peso', '<f8')])
>>> dados['nome']
array(['Ana', 'Pedro'], dtype='<U64')
>>> dados['idade']
array([25, 20])
```

2. NumPy

- Mais detalhes sobre o array:

```
>>> import numpy as np
>>> a = np.array([2,1,5])
>>> a
array([2, 1, 5])
>>> a.shape
(3,)
>>> a.ndim
1
>>> a.size
3
>>> type(a)
<type 'numpy.ndarray'>
```

2. NumPy

- Mais detalhes sobre o array:

Importando o módulo com np para simplificar a chamada das funções.

```
>>> import numpy as np
>>> a = np.array([2,1,5])
>>> a
array([2, 1, 5])
>>> a.shape
(3,)
>>> a.ndim
1
>>> a.size
3
>>> type(a)
<type 'numpy.ndarray'>
```

2. NumPy

- Mais detalhes sobre o array:

```
>>> import numpy as np
>>> a = np.array([2,1,5])
>>> a
array([2, 1, 5])
>>> a.shape
(3,)
>>> a.ndim
1
>>> a.size
3
>>> type(a)
<type 'numpy.ndarray'>
```

Criando um vetor com uma dimensão.

O acesso aos elementos de do array a é feito com colchetes:

```
>>> a[0]
2
>>> a[1]
1
>>> a[2]
5
```

2. NumPy

- Mais detalhes sobre o array:

```
>>> import numpy as np
>>> a = np.array([2,1,5])
>>> a
array([2, 1, 5])
>>> a.shape
(3,)
>>> a.ndim
1
>>> a.size
3
>>> type(a)
<type 'numpy.ndarray'>
```

array: recebe uma lista ou listas de listas.

Lista de elementos – cria uma instância de array de uma dimensão.

Listas de listas – cria uma instância de array de duas ou mais dimensões.

2. NumPy

- Mais detalhes sobre o array:

```
>>> import numpy as np
>>> a = np.array([2,1,5])
>>> a
array([2, 1, 5])
>>> a.shape
(3,)
>>> a.ndim
1
>>> a.size
3
>>> type(a)
<type 'numpy.ndarray'>
```

O formato do vetor (shape) indica o número de elementos por dimensão. Nesse caso, vetor possui uma dimensão (dada por a.ndim) e essa dimensão possui 3 elementos. Atenção! Essa dimensão é diferente da dimensão vista em álgebra!

2. NumPy

- Mais detalhes sobre o array:

```
>>> import numpy as np
>>> a = np.array([2,1,5])
>>> a
array([2, 1, 5])
>>> a.shape
(3,)
>>> a.ndim
1
>>> a.size
3
>>> type(a)
<type 'numpy.ndarray'>
```

A dimensão no numpy é definida pelo número de elementos retornado na tupla do atributo shape.

A dimensão do numpy pode ser pensada como a quantidade de números necessários para definir um ponto em um espaço de dimensão ndim. Ex.: para definir um ponto em uma reta é necessário um valor para x e outro para y – duas dimensões. Para definir um ponto no espaço, é necessário x, y e z – três dimensões.

Um array de uma dimensão não é definido como um vetor linha ou um vetor coluna, seria necessário duas dimensões para tal;

2. NumPy

- Mais detalhes sobre o array:

```
>>> import numpy as np
>>> a = np.array([2,1,5])
>>> a
array([2, 1, 5])
>>> a.shape
(3,)
>>> a.ndim
1
>>> a.size
3
>>> type(a)
<type 'numpy.ndarray'>
```

De forma simplificada, a dimensão é igual ao número de colchetes necessários para escrever um elemento na dimensão mais profunda.

Um array de uma dimensão possui um único colchete antes de chegar a um número. Tal array não é definido como um vetor linha ou um vetor coluna, seria necessário duas dimensões (duas profundidades de colchetes) para tal.

2. NumPy

- Mais detalhes sobre o array:

```
>>> import numpy as np
>>> a = np.array([2,1,5])
>>> a
array([2, 1, 5])
>>> a.shape
(3,)
>>> a.ndim
1
>>> a.size
3
>>> type(a)
<type 'numpy.ndarray'>
```

Número total de elementos.

2. NumPy

- Array com mais de uma dimensão – lista de listas:

```
>>> import numpy as np
>>> vetorLinha = np.array([[1,2,3]])
>>> vetorLinha.shape
(1, 3)
>>> vetorLinha.ndim
2
>>> vetorColuna = np.array([[1],[2],[3]])
>>> vetorColuna.shape
(3, 1)
>>> vetorColuna.ndim
2
```

2. NumPy

- Array com mais de uma dimensão – lista de listas:

```
>>> import numpy as np
>>> vetorLinha = np.array([[1,2,3]])
>>> vetorLinha.shape
(1, 3)
>>> vetorLinha.ndim
2
>>> vetorColuna = np.array([[1],[2],[3]])
>>> vetorColuna.shape
(3, 1)
>>> vetorColuna.ndim
2
```

Lista de listas.

Elementos em uma mesma lista estão na mesma linha.

2. NumPy

- Array com mais de uma dimensão – lista de listas:

```
>>> import numpy as np
>>> vetorLinha = np.array([[1,2,3]])
>>> vetorLinha.shape
(1, 3)
>>> vetorLinha.ndim
2
>>> vetorColuna = np.array([[1],[2],[3]])
>>> vetorColuna.shape
(3, 1)
>>> vetorColuna.ndim
2
```

Formato: matriz 1x3 – uma linha e três colunas.

Duas dimensões.

2. NumPy

- Array com mais de uma dimensão – lista de listas:

```
>>> import numpy as np
>>> vetorLinha = np.array([[1,2,3]])
>>> vetorLinha.shape
(1, 3)
>>> vetorLinha.ndim
2
>>> vetorColuna = np.array([[1],[2],[3]])
>>> vetorColuna.shape
(3, 1)
>>> vetorColuna.ndim
2
```

Lista de listas.

Cada nova lista está definindo uma nova coluna.

2. NumPy

- Array com mais de uma dimensão – lista de listas:

```
>>> import numpy as np
>>> vetorLinha = np.array([[1,2,3]])
>>> vetorLinha.shape
(1, 3)
>>> vetorLinha.ndim
2
>>> vetorColuna = np.array([[1],[2],[3]])
>>> vetorColuna.shape
(3, 1)
>>> vetorColuna.ndim
2
```

Formato: matriz 3x1 – três linha e uma coluna.

Duas dimensões.

2. NumPy

- Array com mais de uma dimensão – lista de listas:

```
>>> import numpy as np
>>> vetorLinha = np.array([[1,2,3]])
>>> vetorLinha.shape
(1, 3)
>>> vetorLinha.ndim
2
>>> vetorColuna = np.array([[1],[2],[3]])
>>> vetorColuna.shape
(3, 1)
>>> vetorColuna.ndim
2
```

Número de elementos da última dimensão é igual ao número de elementos da lista mais profunda.

2. NumPy

- Array com mais de uma dimensão – lista de listas:

```
>>> import numpy as np
>>> vetorLinha = np.array([[1,2,3]])
>>> vetorLinha.shape
(1, 3)
>>> vetorLinha.ndim
2
>>> vetorColuna = np.array([[1],[2],[3]])
>>> vetorColuna.shape
(3, 1)
>>> vetorColuna.ndim
2
```

Como o Python representa esses vetores:

```
>>> vetorLinha
array([[1, 2, 3]])
>>> vetorColuna
array([[1],
       [2],
       [3]])
```

2. NumPy

- Para criar mais dimensões: listas de listas de listas

```
>>> import numpy as np
>>> img=np.array([[[255,0,0],[0,255,0]],[[0,255,0],[0,0,255]]])
>>> img.shape
(2, 2, 3)
>>> img.ndim
3
>>> from matplotlib import pyplot
>>> pyplot.imshow(img)
<matplotlib.image.AxesImage object at 0x08CE9FB0>
>>> pyplot.show()
```

Vetor com três dimensões.

2. NumPy

- Para criar mais dimensões: listas de listas de listas

```
>>> import numpy as np
>>> img=np.array([[[255,0,0],[0,255,0]],[[0,255,0],[0,0,255]]])
>>> img.shape
(2, 2, 3)
>>> img.ndim
3
>>> from matplotlib import pyplot
>>> pyplot.imshow(img)
<matplotlib.image.AxesImage object at 0x08CE9FB0>
>>> pyplot.show()
```

Três elementos.

2. NumPy

- Para criar mais dimensões: listas de listas de listas

```
>>> import numpy as np
>>> img=np.array([[ 255,0,0 , 0,255,0 ],[[0,255,0],[0,0,255]]])
>>> img.shape
(2, 2, 3)
>>> img.ndim
3
>>> from matplotlib import pyplot
>>> pyplot.imshow(img)
<matplotlib.image.AxesImage object at 0x08CE9FB0>
>>> pyplot.show()
```

Dois elementos.

2. NumPy

- Para criar mais dimensões: listas de listas de listas

```
>>> import numpy as np
>>> img=np.array([ [[255,0,0],[0,255,0]] , [[0,255,0],[0,0,255]] ])
>>> img.shape
(2, 2, 3)
>>> img.ndim
3
>>> from matplotlib import pyplot
>>> pyplot.imshow(img)
<matplotlib.image.AxesImage object at 0x08CE9FB0>
>>> pyplot.show()
```

Dois elementos.

2. NumPy

- Para criar mais dimensões: listas de listas de listas

```
>>> import numpy as np
>>> img=np.array([[[255,0,0],[0,255,0]],[[0,255,0],[0,0,255]]])
>>> img.shape
(2, 2, 3)
>>> img.ndim
3
>>> from matplotlib import pyplot
>>> pyplot.imshow(img)
<matplotlib.image.AxesImage object at 0x08CE9FB0>
>>> pyplot.show()
```

Uma imagem é representada com um vetor de 3 dimensões. Cada matriz representa um canal de cor: RGB.

2. NumPy

- Para criar mais dimensões: listas de listas de listas

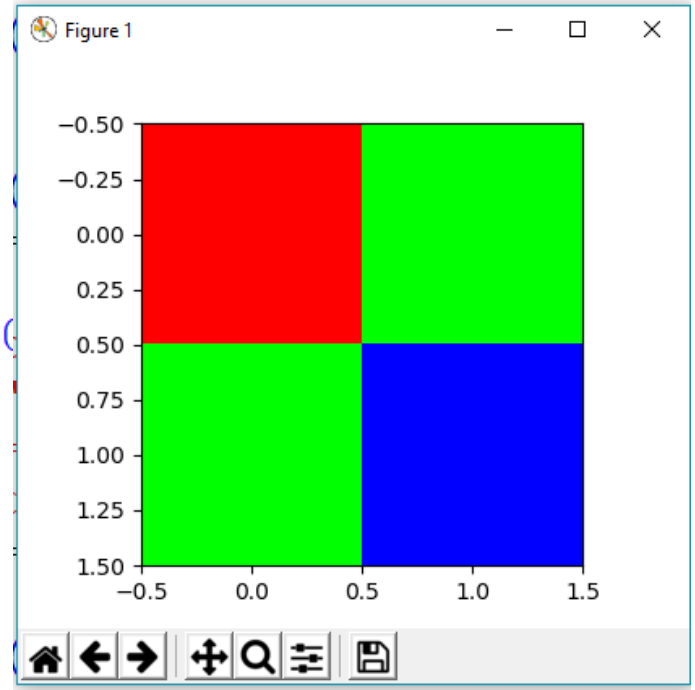
```
>>> import numpy as np
>>> img=np.array([[[255,0,0],[0,255,0]],[[0,255,0],[0,0,255]]])
>>> img.shape
(2, 2, 3)
>>> img.ndim
3
>>> from matplotlib import pyplot
>>> pyplot.imshow(img)
<matplotlib.image.AxesImage object at 0x08CE9FB0>
>>> pyplot.show()
```

$$\underset{R}{\begin{bmatrix}255 & 0\\ 0 & 0\end{bmatrix}}, \underset{G}{\begin{bmatrix}0 & 255\\ 255 & 0\end{bmatrix}}, \underset{B}{\begin{bmatrix}0 & 0\\ 0 & 255\end{bmatrix}}$$

2. NumPy

- Para criar mais dimensões: listas de listas de listas

```
>>> import numpy as np
>>> img=np.array([[[255,0,0],[0,255,0]],[[0,255,0],[0,0,255]]])
>>> img.shape
(2, 2, 3)
>>> img.ndim
3
>>> from matplotlib import pyplot
>>> pyplot.imshow(img)
<matplotlib.image.AxesImage object at 0
>>> pyplot.show()
```



2. NumPy

- Algumas operações:

```
>>> a = np.array([2,1,5])
>>> b = np.array([7,6,9])
>>> a+b
array([ 9,  7, 14])
>>> a-b
array([-5, -5, -4])
>>> a*b
array([14,  6, 45])
>>> a.dot(b)
65
>>> b/a
array([3, 6, 1])
>>> 3*a
array([ 6,  3, 15])
>>> a/2
array([1, 0, 2])
>>> c = np.arange(5)
>>> c
array([0, 1, 2, 3, 4])
```

Vetores a e b que possuem uma única dimensão.

2. NumPy

- Algumas operações:

```
>>> a = np.array([2,1,5])
>>> b = np.array([7,6,9])
>>> a+b
array([ 9,  7, 14])
>>> a-b
array([-5, -5, -4])
>>> a*b
array([14,  6, 45])
>>> a.dot(b)
65
>>> b/a
array([3, 6, 1])
>>> 3*a
array([ 6,  3, 15])
>>> a/2
array([1, 0, 2])
>>> c = np.arange(5)
>>> c
array([0, 1, 2, 3, 4])
```

Atenção: o * não faz um produto interno e sim uma multiplicação elemento a elemento.

2. NumPy

- Algumas operações:

```
>>> a = np.array([2,1,5])
>>> b = np.array([7,6,9])
>>> a+b
array([ 9,  7, 14])
>>> a-b
array([-5, -5, -4])
>>> a*b
array([14,  6, 45])
>>> a.dot(b)
65
>>> b/a
array([3, 6, 1])
>>> 3*a
array([ 6,  3, 15])
>>> a/2
array([1, 0, 2])
>>> c = np.arange(5)
>>> c
array([0, 1, 2, 3, 4])
```

Para fazer o produto interno (ou uma multiplicação de matrizes)é necessário usar a função dot. Como a e b são vetores de uma só dimensão, o numpy não considera necessário determinar um vetor linha e outro coluna, pois nesse caso seriam duas dimensões. Com só uma dimensão: a.dot(b) = b.dot(a)

2. NumPy

- Algumas operações:

```
>>> a = np.array([2,1,5])
>>> b = np.array([7,6,9])
>>> a+b
array([ 9,  7, 14])
>>> a-b
array([-5, -5, -4])
>>> a*b
array([14,  6, 45])
>>> a.dot(b)
65
>>> b/a
array([3, 6, 1])
>>> 3*a
array([ 6,  3, 15])
>>> a/2
array([1, 0, 2])
>>> c = np.arange(5)
>>> c
array([0, 1, 2, 3, 4])
```

Divisão elemento a elemento.

2. NumPy

- Algumas operações:

```
>>> a = np.array([2,1,5])
>>> b = np.array([7,6,9])
>>> a+b
array([ 9,  7, 14])
>>> a-b
array([-5, -5, -4])
>>> a*b
array([14,  6, 45])
>>> a.dot(b)
65
>>> b/a
array([3, 6, 1])
>>> 3*a
array([ 6,  3, 15])
>>> a/2
array([1, 0, 2])
>>> c = np.arange(5)
>>> c
array([0, 1, 2, 3, 4])
```

Multiplicando e dividindo cada elemento de 'a' por um número.

2. NumPy

- Algumas operações:

```
>>> a = np.array([2,1,5])
>>> b = np.array([7,6,9])
>>> a+b
array([ 9,  7, 14])
>>> a-b
array([-5, -5, -4])
>>> a*b
array([14,  6, 45])
>>> a.dot(b)
65
>>> b/a
array([3, 6, 1])
>>> 3*a
array([ 6,  3, 15])
>>> a/2
array([1, 0, 2])
>>> c = np.arange(5)
>>> c
array([0, 1, 2, 3, 4])
```

Equivalente à função range, mas cria uma variável do numpy.

2. NumPy

- Algumas operações:

```
>>> a = np.array([2,1,5])
>>> list(a)
[2, 1, 5]
>>> tuple(a)
(2, 1, 5)
>>> b = np.linspace(0,1.5,4)
>>> b
array([0. , 0.5, 1. , 1.5])
>>> c = np.random.rand(3,)
>>> c
array([0.33981563, 0.91879702, 0.12850224])
>>> a = np.array([[2,1,5]])
>>> a.shape
(1, 3)
>>> a.ndim
2
>>> a.size
3
```

Para transformar um array do numpy em lista ou tupla basta usar list e tuple.

2. NumPy

- Algumas operações:

```
>>> a = np.array([2,1,5])
>>> list(a)
[2, 1, 5]
>>> tuple(a)
(2, 1, 5)
>>> b = np.linspace(0,1.5,4)
>>> b
array([0. , 0.5, 1. , 1.5])
>>> c = np.random.rand(3,)
>>> c
array([0.33981563, 0.91879702, 0.12850224])
>>> a = np.array([[2,1,5]])
>>> a.shape
(1, 3)
>>> a.ndim
2
>>> a.size
3
```

Cria um array com 4 elementos igualmente espaçados cujo primeiro elemento será 0 e o último será 1.5

2. NumPy

- Algumas operações:

```
>>> a = np.array([2,1,5])
>>> list(a)
[2, 1, 5]
>>> tuple(a)
(2, 1, 5)
>>> b = np.linspace(0,1.5,4)
>>> b
array([0. , 0.5, 1. , 1.5])
>>> c = np.random.rand(3,)
>>> c
array([0.33981563, 0.91879702, 0.12850224])
>>> a = np.array([[2,1,5]])
>>> a.shape
(1, 3)
>>> a.ndim
2
>>> a.size
3
```

Cria um array aleatório com 3 elementos. A função recebe o formato da matriz aleatória que deve criar. Como foi passado (3,), estamos criando um array de uma dimensão e 3 elementos.

2. NumPy

- Algumas operações:

```
>>> a = np.array([2,1,5])
>>> list(a)
[2, 1, 5]
>>> tuple(a)
(2, 1, 5)
>>> b = np.linspace(0,1.5,4)
>>> b
array([0. , 0.5, 1. , 1.5])
>>> c = np.random.rand(3,)
>>> c
array([0.33981563, 0.91879702, 0.12850224])
>>> a = np.array([[2,1,5]])
>>> a.shape
(1, 3)
>>> a.ndim
2
>>> a.size
3
```

Estamos criando a, mas utilizamos uma lista de lista como argumento da função array. Por esse motivo, é criada uma matriz com uma linha e 3 colunas.

2. NumPy

- Algumas operações:

```
>>> a = np.array([2,1,5])
>>> list(a)
[2, 1, 5]
>>> tuple(a)
(2, 1, 5)
>>> b = np.linspace(0,1.5,4)
>>> b
array([0. , 0.5, 1. , 1.5])
>>> c = np.random.rand(3,)
>>> c
array([0.33981563, 0.91879702, 0.12850224])
>>> a = np.array([[2,1,5]])
>>> a.shape
(1, 3)
>>> a.ndim
2
>>> a.size
3
```

Compare com o resultado de shape que obtivemos antes. O primeiro número é o número de linhas da matriz e o segundo é o número de colunas – vetor linha.

2. NumPy

- Algumas operações:

```
>>> a = np.array([2,1,5])
>>> list(a)
[2, 1, 5]
>>> tuple(a)
(2, 1, 5)
>>> b = np.linspace(0,1.5,4)
>>> b
array([0. , 0.5, 1. , 1.5])
>>> c = np.random.rand(3,)
>>> c
array([0.33981563, 0.91879702, 0.12850224])
>>> a = np.array([[2,1,5]])
>>> a.shape
(1, 3)
>>> a.ndim
2
>>> a.size
3
```

Agora temos duas dimensões, mas o número de elementos continua o mesmo.

Como temos duas dimensões, agora temos que tomar mais cuidado com as operações.

2. NumPy

- Algumas operações:

```
>>> a = np.array([[2,1,5]])
>>> b = np.array([[7,6,9]])
>>> a+b
array([[ 9,  7, 14]])
>>> a-b
array([[-5, -5, -4]])
>>> a*b
array([[14,  6, 45]])
>>> b/a
array([[3, 6, 1]])
>>> a.dot(b)

Traceback (most recent call last):
  File "<pyshell#199>", line 1, in
<module>
    a.dot(b)
ValueError: shapes (1,3) and (1,3)
not aligned: 3 (dim 1) != 1 (dim 0)
```

Criando dois vetores linha.

2. NumPy

- Algumas operações:

```
>>> a = np.array([[2,1,5]])
>>> b = np.array([[7,6,9]])
>>> a+b
array([[ 9,  7, 14]])
>>> a-b
array([[-5, -5, -4]])
>>> a*b
array([[14,  6, 45]])
>>> b/a
array([[3, 6, 1]])
>>> a.dot(b)

Traceback (most recent call last):
  File "<pyshell#199>", line 1, in
<module>
    a.dot(b)
ValueError: shapes (1,3) and (1,3)
not aligned: 3 (dim 1) != 1 (dim 0)
```

As operações elemento a elemento continuam iguais – também funcionaria se uma das variáveis fosse um vetor com 3 elementos e outra fosse um vetor 3x1.

2. NumPy

- Algumas operações:

```
>>> a = np.array([[2,1,5]])
>>> b = np.array([[7,6,9]])
>>> a+b
array([[ 9,  7, 14]])
>>> a-b
array([[-5, -5, -4]])
>>> a*b
array([[14,  6, 45]])
>>> b/a
array([[3, 6, 1]])
>>> a.dot(b)

Traceback (most recent call last):
  File "<pyshell#199>", line 1, in
<module>
    a.dot(b)
ValueError: shapes (1,3) and (1,3)
not aligned: 3 (dim 1) != 1 (dim 0)
```

A operação de produto interno não funciona, pois o produto interno requer que o número de colunas da primeira matriz seja igual ao número de linhas da segunda matriz.

Para que funcione, precisamos de um vetor coluna.

2. NumPy

- Algumas operações:

```
>>> a = np.array([[2,1,5]])
>>> a
array([[2, 1, 5]])
>>> b = np.array([[7],[6],[9]])
>>> b
array([[7],
       [6],
       [9]])
>>> a+b
array([[ 9,  8, 12],
       [ 8,  7, 11],
       [11, 10, 14]])
>>> a-b
array([[-5, -6, -2],
       [-4, -5, -1],
       [-7, -8, -4]])
>>> a*b
array([[14,  7, 35],
       [12,  6, 30],
       [18,  9, 45]])
```

Criando um vetor coluna.

```
>>> b/a
array([[3, 7, 1],
       [3, 6, 1],
       [4, 9, 1]])
```

2. NumPy

- Algumas operações:

```
>>> a = np.array([[2,1,5]])
>>> a
array([[2, 1, 5]])
>>> b = np.array([[7],[6],[9]])
>>> b
array([[7],
       [6],
       [9]])
>>> a+b
array([[ 9,  8, 12],
       [ 8,  7, 11],
       [11, 10, 14]])
>>> a-b
array([[-5, -6, -2],
       [-4, -5, -1],
       [-7, -8, -4]])
>>> a*b
array([[14,  7, 35],
       [12,  6, 30],
       [18,  9, 45]])
```

```
>>> b/a
array([[3, 7, 1],
       [3, 6, 1],
       [4, 9, 1]])
```

As operações de soma, subtração, multiplicação e divisão criam matrizes.

2. NumPy

- Algumas operações:

```
>>> a = np.array([[2,1,5]])
>>> a
array([[2, 1, 5]])
>>> b = np.array([[7],[6],[9]])
>>> b
array([[7],
       [6],
       [9]])
>>> a+b
array([[ 9,  8, 12],
       [ 8,  7, 11],
       [11, 10, 14]])
>>> a-b
array([[-5, -6, -2],
       [-4, -5, -1],
       [-7, -8, -4]])
>>> a*b
array([[14,  7, 35],
       [12,  6, 30],
       [18,  9, 45]])
```

```
>>> b/a
array([[3, 7, 1],
       [3, 6, 1],
       [4, 9, 1]])
```

As operações de soma, subtração, multiplicação e divisão criam matrizes.

2. NumPy

- Algumas operações:

```
>>> a = np.array([[2,1,5]])
>>> a
array([[2, 1, 5]])
>>> b = np.array([[7],[6],[9]])
>>> b
array([[7],
       [6],
       [9]])
>>> a+b
array([[ 9,  8, 12],
       [ 8,  7, 11],
       [11, 10, 14]])
>>> a-b
array([[-5, -6, -2],
       [-4, -5, -1],
       [-7, -8, -4]])
>>> a*b
array([[14,  7, 35],
       [12,  6, 30],
       [18,  9, 45]])
```

```
>>> b/a
array([[3, 7, 1],
       [3, 6, 1],
       [4, 9, 1]])
```

As operações de soma, subtração, multiplicação e divisão criam matrizes.

2. NumPy

- Algumas operações:

```
>>> a = np.array([[2,1,5]])
>>> a
array([[2, 1, 5]])
>>> b = np.array([[7],[6],[9]])
>>> b
array([[7],
       [6],
       [9]])
>>> a+b
array([[ 9,  8, 12],
       [ 8,  7, 11],
       [11, 10, 14]])
>>> a-b
array([[-5, -6, -2],
       [-4, -5, -1],
       [-7, -8, -4]])
>>> a*b
array([[14,  7, 35],
       [12,  6, 30],
       [18,  9, 45]])
```

```
>>> b/a
array([[3, 7, 1],
       [3, 6, 1],
       [4, 9, 1]])
```

As operações de soma, subtração, multiplicação e divisão criam matrizes.

2. NumPy

- Algumas operações:

```
>>> a = np.array([[2,1,5]])
>>> a
array([[2, 1, 5]])
>>> b = np.array([[7],[6],[9]])
>>> b
array([[7],
       [6],
       [9]])
>>> a+b
array([[ 9,  8, 12],
       [ 8,  7, 11],
       [11, 10, 14]])
>>> a-b
array([[-5, -6, -2],
       [-4, -5, -1],
       [-7, -8, -4]])
>>> a*b
array([[14,  7, 35],
       [12,  6, 30],
       [18,  9, 45]])
```

```
>>> b/a
array([[3, 7, 1],
       [3, 6, 1],
       [4, 9, 1]])
```

As operações de soma, subtração, multiplicação e divisão criam matrizes.

2. NumPy

- Algumas operações:

```
>>> a = np.array([[2,1,5]])
>>> a
array([[2, 1, 5]])
>>> b = np.array([[7],[6],[9]])
>>> b
array([[7],
       [6],
       [9]])
>>> a+b
array([[ 9,  8, 12],
       [ 8,  7, 11],
       [11, 10, 14]])
>>> a-b
array([[-5, -6, -2],
       [-4, -5, -1],
       [-7, -8, -4]])
>>> a*b
array([[14,  7, 35],
       [12,  6, 30],
       [18,  9, 45]])
```

```
>>> b/a
array([[3, 7, 1],
       [3, 6, 1],
       [4, 9, 1]])
```

As operações de soma, subtração, multiplicação e divisão criam matrizes.

2. NumPy

- Algumas operações:

```
>>> a = np.array([[2,1,5]])
>>> b = np.array([[7],[6],[9]])
>>> a.dot(b)
array([[65]])
>>> b.dot(a)
array([[14,  7, 35],
       [12,  6, 30],
       [18,  9, 45]])
>>> a[0,0]
2
>>> a[0,1]
1
>>> a[0,:]
array([2, 1, 5])
>>> b[0,0]
7
>>> b[1,0]
6
>>> b[:,0]
array([7, 6, 9])
```

Produto interno.

2. NumPy

- Algumas operações:

```
>>> a = np.array([[2,1,5]])
>>> b = np.array([[7],[6],[9]])
>>> a.dot(b)
array([[65]])
>>> b.dot(a)
array([[14,  7, 35],
       [12,  6, 30],
       [18,  9, 45]])
>>> a[0,0]
2
>>> a[0,1]
1
```

Acessando o elemento da posição (0,0) e o elemento da posição (0,1).

```
>>> a[0,:]
array([2, 1, 5])
>>> b[0,0]
7
>>> b[1,0]
6
>>> b[:,0]
array([7, 6, 9])
```

Acessando o elemento da posição (0,0) e o elemento da posição (1,0).

2. NumPy

- Algumas operações:

```
>>> a = np.array([[2,1,5]])
>>> b = np.array([[7],[6],[9]])
>>> a.dot(b)
array([[65]])
>>> b.dot(a)
array([[14,  7, 35],
       [12,  6, 30],
       [18,  9, 45]])
>>> a[0,0]
2
>>> a[0,1]
1
>>> a[0,:]
array([2, 1, 5])
>>> b[0,0]
7
>>> b[1,0]
6
>>> b[:,0]
array([7, 6, 9])
```

Acessando todos os elementos da linha 0.

Acessando todos os elementos da coluna 0.

2. NumPy

- Outras formas de criar um vetor:

```
>>> a = np.zeros((2,2))
>>> a
array([[0., 0.],
       [0., 0.]])
>>> b = np.ones((2,2))
>>> b
array([[1., 1.],
       [1., 1.]])
>>> c = np.full((2,2),5)
>>> c
array([[5, 5],
       [5, 5]])
>>> d = np.eye(2)
>>> d
array([[1., 0.],
       [0., 1.]])
>>> e = np.random.random((2,2))
>>> e
array([[0.52578892, 0.24610589],
       [0.18222576, 0.61568408]])
```

2. NumPy

- Outras formas de criar um vetor:

```
>>> a = np.zeros((2,2))
>>> a
array([[0., 0.],
       [0., 0.]])
>>> b = np.ones((2,2))
>>> b
array([[1., 1.],
       [1., 1.]])
>>> c = np.full((2,2),5)
>>> c
array([[5, 5],
       [5, 5]])
>>> d = np.eye(2)
>>> d
array([[1., 0.],
       [0., 1.]])
>>> e = np.random.random((2,2))
>>> e
array([[0.52578892, 0.24610589],
       [0.18222576, 0.61568408]])
```

Array de zeros.

2. NumPy

- Outras formas de criar um vetor:

```
>>> a = np.zeros((2,2))
>>> a
array([[0., 0.],
       [0., 0.]])
>>> b = np.ones((2,2))
>>> b
array([[1., 1.],
       [1., 1.]])
>>> c = np.full((2,2),5)
>>> c
array([[5, 5],
       [5, 5]])
>>> d = np.eye(2)
>>> d
array([[1., 0.],
       [0., 1.]])
>>> e = np.random.random((2,2))
>>> e
array([[0.52578892, 0.24610589],
       [0.18222576, 0.61568408]])
```

Array de uns.

2. NumPy

- Outras formas de criar um vetor:

```
>>> a = np.zeros((2,2))
>>> a
array([[0., 0.],
       [0., 0.]])
>>> b = np.ones((2,2))
>>> b
array([[1., 1.],
       [1., 1.]])
>>> c = np.full((2,2),5)
>>> c
array([[5, 5],
       [5, 5]])
>>> d = np.eye(2)
>>> d
array([[1., 0.],
       [0., 1.]])
>>> e = np.random.random((2,2))
>>> e
array([[0.52578892, 0.24610589],
       [0.18222576, 0.61568408]])
```

Array de elementos constante com valor selecionado pelo programador.

2. NumPy

- Outras formas de criar um vetor:

```
>>> a = np.zeros((2,2))
>>> a
array([[0., 0.],
       [0., 0.]])
>>> b = np.ones((2,2))
>>> b
array([[1., 1.],
       [1., 1.]])
>>> c = np.full((2,2),5)
>>> c
array([[5, 5],
       [5, 5]])
>>> d = np.eye(2)
>>> d
array([[1., 0.],
       [0., 1.]])
>>> e = np.random.random((2,2))
>>> e
array([[0.52578892, 0.24610589],
       [0.18222576, 0.61568408]])
```

Matriz identidade do tipo array.

2. NumPy

- Outras formas de criar um vetor:

```
>>> a = np.zeros((2,2))
>>> a
array([[0., 0.],
       [0., 0.]])
>>> b = np.ones((2,2))
>>> b
array([[1., 1.],
       [1., 1.]])
>>> c = np.full((2,2),5)
>>> c
array([[5, 5],
       [5, 5]])
>>> d = np.eye(2)
>>> d
array([[1., 0.],
       [0., 1.]])
>>> e = np.random.random((2,2))
>>> e
array([[0.52578892, 0.24610589],
       [0.18222576, 0.61568408]])
```

Array com elementos aleatórios.

2. NumPy

- Operações de álgebra linear:

```
>>> a = np.array([[1,2,1],[3,-9,15],[2,-1,3]])
>>> a.transpose()
array([[ 1,  3,  2],
       [ 2, -9, -1],
       [ 1, 15,  3]])
>>> np.linalg.inv(a)
array([[-0.26666667, -0.15555556,  0.86666667],
       [ 0.46666667,  0.02222222, -0.26666667],
       [ 0.33333333,  0.11111111, -0.33333333]])
```

2. NumPy

- Operações de álgebra linear:

  Obs.: resolução de sistemas lineares onde o número de equações é igual ao número de incógnitas.

  Considere o seguinte sistema linear:

$$A \cdot x = b$$

  Se A possui inversa:

$$A^{-1} \cdot A \cdot x = A^{-1} \cdot b \;\rightarrow\; x = A^{-1} \cdot b$$

2. NumPy

- Resolvendo o sistema:

```
>>> a = np.array([[1,2,1],[3,-9,15],[2,-1,3]])
>>> b = np.array([12,3,10])
>>> x = np.linalg.inv(a).dot(b)
>>> x
array([5., 3., 1.])
```

- Numpy fornece uma função para resolução de sistemas lineares:

```
>>> a = np.array([[1,2,1],[3,-9,15],[2,-1,3]])
>>> b = np.array([12,3,10])
>>> x = np.linalg.solve(a,b)
>>> x
array([5., 3., 1.])
```

2. NumPy

- Operações de álgebra linear:

  Obs2.: resolução de sistemas com mais equações que incógnitas – método dos mínimos quadrados.

  Considere o seguinte sistema linear onde A não é quadrada:

$$A \cdot x = b$$

  Se $A^T A$ possui inversa:

$$A^T \cdot A \cdot x = A^T \cdot b$$

$$x = (A^T \cdot A)^{-1} \cdot A^T \cdot b$$

2. NumPy

- Vamos resolver o seguinte sistema pelo método dos mínimos quadrados:
  - Os seguintes pontos foram medidos:
  - (0,2),(3,5),(5,6),(8,9),(10,11)
  - Desejamos encontrar a melhor reta para se adequar a esses pontos. Como o sistema é construído?

2. NumPy

- Vamos resolver o seguinte sistema pelo método dos mínimos quadrados:
  - Os seguintes pontos foram medidos:
  - (0,2),(3,5),(5,6),(8,9),(10,11)
  - Desejamos encontrar a melhor reta para se adequar a esses pontos. Como o sistema é construído?
  - O seguinte sistema de equações define tal problema:

$$\begin{bmatrix} 0 & 1 \\ 3 & 1 \\ 5 & 1 \\ 8 & 1 \\ 10 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 5 \\ 9 \\ 11 \end{bmatrix}$$

2. NumPy

- Resolvendo um sistema pelo método dos mínimos quadrados:

$$\begin{bmatrix} 0 & 1 \\ 3 & 1 \\ 5 & 1 \\ 8 & 1 \\ 10 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 5 \\ 9 \\ 11 \end{bmatrix}$$

```
>>> a = np.array([[0,1],[3,1],[5,1],[8,1],[10,1]])
>>> b = np.array([2,5,6,9,11])
>>> at = a.transpose()
>>> x = (np.linalg.inv(at.dot(a)).dot(at)).dot(b)
>>> x
array([0.88216561, 2.01273885])
```

2. NumPy

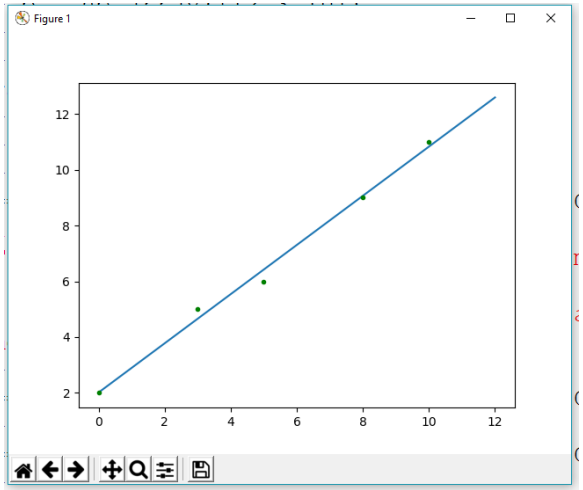
- Resolvendo um sistema pelo método dos mínimos quadrados:

```
import numpy as np
from matplotlib import pyplot
def minimosQuadrados(pontos):
    listaA = []
    listaB = []
    for ponto in pontos:
        listaA.append([ponto[0],1])
        listaB.append([ponto[1]])
    A = np.array(listaA)
    b = np.array(listaB)
    AT = A.transpose()
    return (np.linalg.inv(AT.dot(A)).dot(AT)).dot(b)
pontos = [(0,2),(3,5),(5,6),(8,9),(10,11)]
a,b = minimosQuadrados(pontos)
x = np.linspace(0,12,4)
y = a*x + b
pyplot.plot(x,y)
pyplot.plot(np.array(pontos)[:,0],np.array(pontos)[:,1],'g.')
pyplot.show()
pyplot.close()
```

2. NumPy

- Resolvendo um sistema pelo método dos mínimos quadrados:

```
import numpy as np
from matplotlib import pyplot
def minimosQuadrados(pontos):
    listaA = []
    listaB = []
    for ponto in pontos:
        listaA.append([ponto[0],1])
        listaB.append([ponto[1]])
    A = np.array(listaA)
    b = np.array(listaB)
    AT = A.transpose()
    return (np.linalg.inv(AT.dot(A)).dot(AT)).dot(b)
pontos = [(0,2),(3,5),(5,6),(8,9),(10,11)]
a,b = minimosQuadrados(pontos)
x = np.linspace(0,12,4)
y = a*x + b
pyplot.plot(x,y)
pyplot.plot(np.array(pontos)[:,0],np.array(pontos)[:,1],'g.')
pyplot.show()
pyplot.close()
```



2. NumPy

- Numpy:

  Acessando os elementos com for:

```
>>> for linha in a:
        type(linha)
```

Linha a linha

```
>>> for lin in range(len(a[:,0])):
        for col in range(len(a[0,:])):
            print a[lin,col]
```

Elemento a elemento

```
>>> for elemento in a.flat:
        print elemento
```

Elemento a elemento

  Algumas outras funções interessantes: copy, identity, logspace, reshape, resize, max, min, sum, std, var, linalg.eig ...

  O numpy também permite criar uma variável do tipo matrix. Esse tipo de variável necessariamente tem 2 dimensões e tem como objetivo facilitar cálculos de álgebra linear, mas as diferenças são pequenas.