XVIII Escola de Verão
IAG/USP

Numerical Python → NumPy

Marcelo Bianchi, Victor Sacek, Leonardo Uieda

Jan/2016

O que é

- É um módulo parte de um projeto maior, chamado de scipy que oferece tipos ***vetorizados*** de ***n dimensões*** e um conjunto básico de funções para:
  - álgebra linear
  - transformada de Fourrier
  - cálculo de número aleatórios
  - estatística básica
  - e etc.

Vetorização

```
a = [1,2,3]
```

```
a + 1
```

```
a / 2
```

```
a * 2
```

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

- A única operação que funciona não executa o esperado !

Vetorização

- Vetorização é a capacidade de operar em vetores ou mesmo em matrizes sem o uso de funções auxiliares ou laços !

```
import numpy as np

numeros = np.array([1,2,3])
print("     numeros = ",numeros)
print("2 x  numeros = ",numeros * 2)
print("numeros  + 1 = ",numeros + 1)
print("numeros // 2 = ",numeros // 2)
```

```
     numeros =  [1 2 3]
2 x  numeros =  [2 4 6]
numeros  + 1 =  [2 3 4]
numeros // 2 =  [0 1 1]
```

n dimensional

```
ndim = np.zeros((2,))
print(ndim)
```

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

```
ndim = np.zeros((2,2))
print(ndim)
```

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

```
ndim = np.zeros((2,2,2))
print(ndim)
```

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

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

“Shape”, é o nome utilizado pelo NumPy para a forma ou dimensão de um array.

Importando o numpy

- O modo “tradicional” de importar o numpy é utilizando:

```
import numpy as np
```

- O principal* tipo disponibilizado pelo Numpy é a classe “ndarray” !

```
ndarray = np.ndarray((1,))
print(type(ndarray),ndarray)
```

```
<type 'numpy.ndarray'> [ 0.]
```

* O Numpy também oferece um tipo para representar matrizes que operam então como matrizes e não como vetores n dimensionais

Criando instâncias do *ndarray*

- np.array(**VALORES**)
  - Retorna um *ndarray* preenchido com os valores indicados
- np.zeros(**SHAPE**)
  - Retorna um *ndarray* com zeros, no *shape* indicado
- np.ones(**SHAPE**)
  - Retorna um *ndarray* com “1's”, no *shape* indicado
- np.empty(**SHAPE**)
  - Retorna um *ndarray* com valores aleatórios, no *shape* indicado
- np.linspace(**XMIN, XMAX, N**)
  - Retorna um *ndarray*, com uma sequência linear de *N* pontos entre os intervalos dados que podem ou não excluir o valor de *XMAX*
- np.arange(**XMIN, XMAX, INC**)
  - Retorna um *ndarray* com uma sequência linear de números gerados no intervalo (excluindo *XMAX*) de incremento *INC*

Lendo um *ndarray* de um arquivo

```
import numpy as np
```

```
%cat lala
```

```
1 2
3 4
5 6
```

Um arquivo com duas colunas

```
np.loadtxt("lala", unpack=False)
```

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

Load **sem** desenpacotamento

```
np.loadtxt("lala", unpack=True)
```

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

Load **com** desenpacotamento

Visualizando os ndarray

- Para visualizar um ndarray (os seus valores) basta imprimi-lo, utilizando o comando print() !

```
ndim = np.zeros((2,2))
print(ndim)
```

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

Indexação & Slicing

- O acesso a elementos do ndarray é feito utilizando os […]'s !
- [T]
  - Obtém o elemento na posição T, posição da lista inicia em 0
- [S:E]
  - Faz um slice de S até E (sem incluir o E)
- [S:E:D]
  - Faz um slice de S até E de D em D
- O mesmo vale quando estamos lidando em múltiplas dimensões que devem ser separadas por “,”
- Os índices dados podem ser negativos, neste caso, a contagem das posições se da do fim para o início !

Indexação avançada

- Dentro dos [...]'s podemos utilizar listas para selecionar os elementos

```
import numpy as np
```

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

```
nd[range(1,5)]
```

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

Indexação avançada

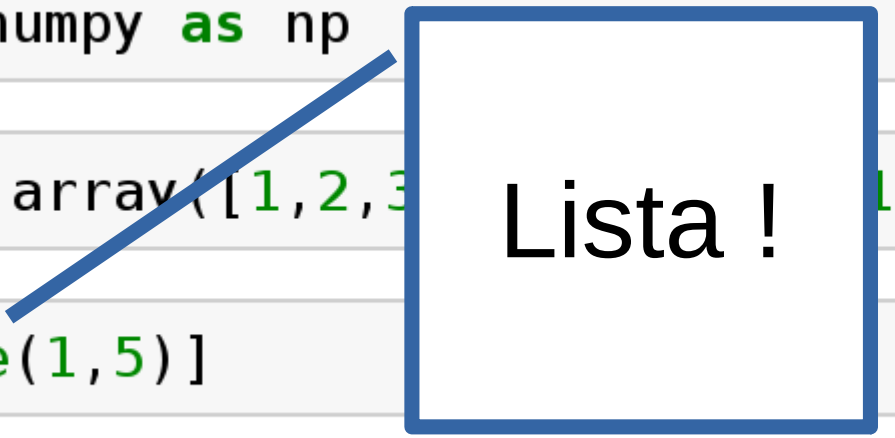
- Dentro dos [...]'s podemos utilizar listas para selecionar os elementos

```
import numpy as np
```

```
nd = np.array([1,2,3                    10])
```

```
nd[range(1,5)]
```

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



Moldando um ndarray

- O elemento *ndarray* pode mudar de forma através do método *reshape(SHAPE)*;
- Use a propriedade *shape* para descobrir o shape atual;

```
a = np.arange(1,5)
print("a[%s]=\n%s"%(a.shape, str(a)))
```

```
a[(4,)]=
[1 2 3 4]
```

```
a = a.reshape(2,2)
print("a[%s]=\n%s"%(a.shape, str(a)))
```

```
a[(2, 2)]=
[[1 2]
 [3 4]]
```

```
a = a.reshape(4,1)
print("a[%s]=\n%s"%(a.shape, str(a)))
```

```
a[(4, 1)]=
[[1]
 [2]
 [3]
 [4]]
```

Operações Básicas

- De modo geral o ndarray é multiplicado elemento a elemento para listas de mesmo shape !
- Em casos onde os shapes diferem ele busca estender o elemento de menor shape no maior, o caso mais simples é:

```
import numpy as np
```

```
np.arange(0,10) * 2
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

O dois é multiplicado por cada elemento

Funções Universais

- No *NumPy* também temos diversas funções básicas para serem utilizadas de forma a otimizar o código !
- Elas implementam as funções básicas como sin, cos, log, exp, sqrt, … ,etc
- E operam em geral elemento a elemento de forma otimizada em vetores ndarray !

```
np.cos( np.linspace(-np.pi,np.pi,5) )
```

```
array([ -1.00000000e+00,   6.12323400e-17,   1.00000000e+00,
         6.12323400e-17,  -1.00000000e+00])
```

np.linalg

Linear algebra (numpy.linalg)

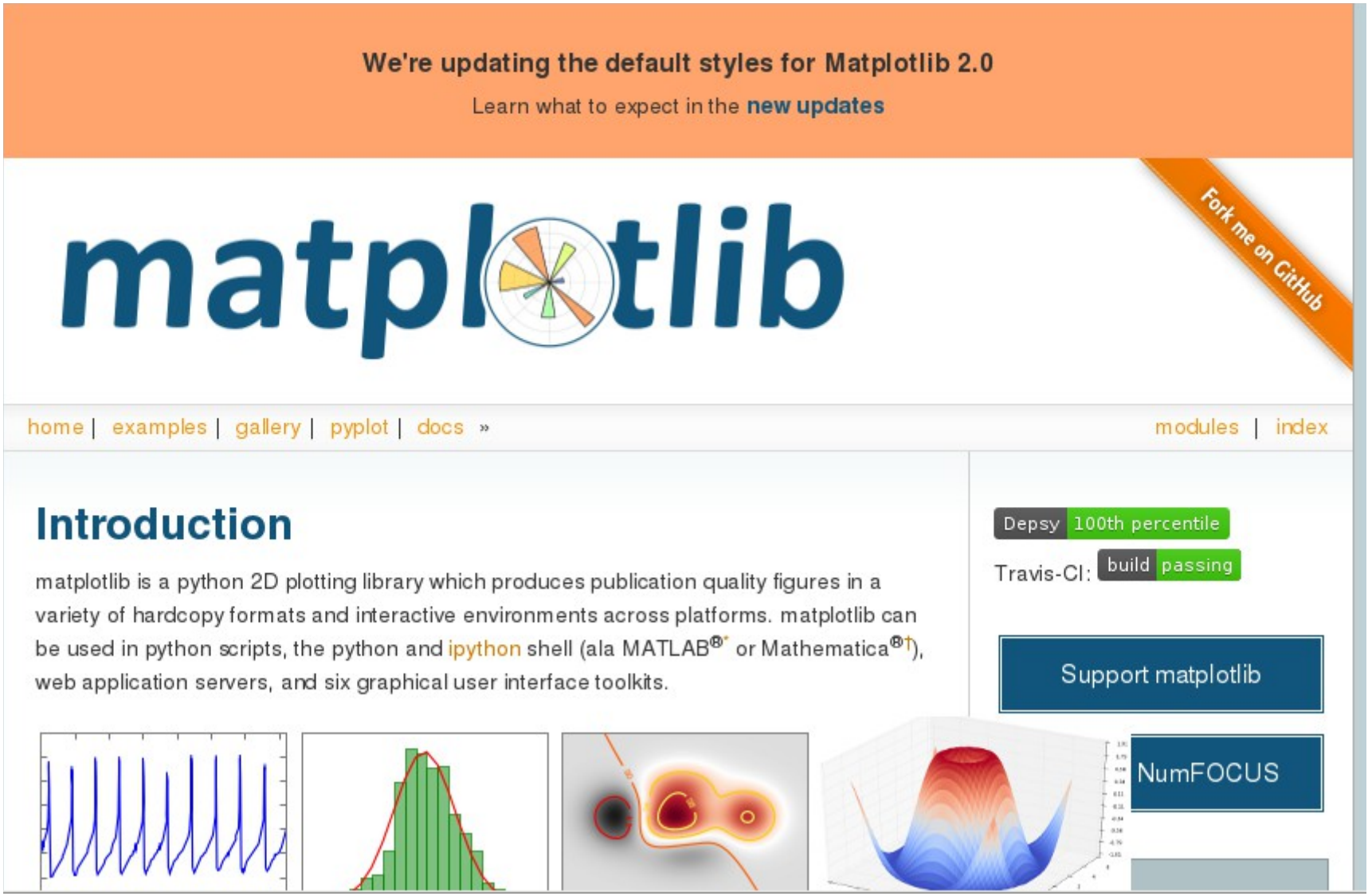
Matrix and vector products

dot(a, b[, out])	Dot product of two arrays.
vdot(a, b)	Return the dot product of two vectors.
inner(a, b)	Inner product of two arrays.
outer(a, b[, out])	Compute the outer product of two vectors.
matmul(a, b[, out])	Matrix product of two arrays.
tensordot(a, b[, axes])	Compute tensor dot product along specified axes for arrays >= 1-D.
einsum(subscripts, *operands[, out, dtype, ...])	Evaluates the Einstein summation convention on the operands.
linalg.matrix_power(M, n)	Raise a square matrix to the (integer) power *n*.
kron(a, b)	Kronecker product of two arrays.

Table Of Contents

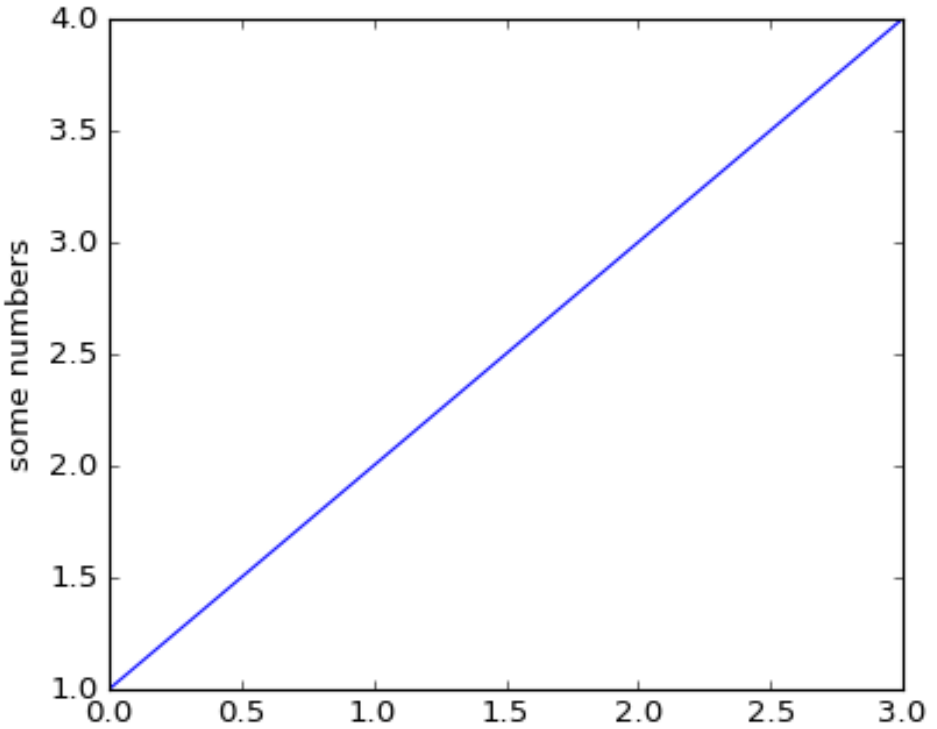
- Linear algebra (**numpy.linalg**)
  - Matrix and vector products
  - Decomposition
  - Matrix eigenvalues
  - Norms and other numbers
  - Solving equations and inverting matrices
  - Exceptions
  - Linear algebra on several matrices at once

Matplotlib


We're updating the default styles for Matplotlib 2.0
Learn what to expect in the new updates
matplotlib
Fork me on GitHub
home | examples | gallery | pyplot | docs »
modules | index
Introduction
matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala MATLAB® or Mathematica®†), web application servers, and six graphical user interface toolkits.
Depsy 100th percentile
Travis-CI: build passing
Support matplotlib
NumFOCUS

Matplotlib

- É o principal módulo do Python capaz de gerar gráficos
- Ele é carregado dentro do IPython notebook da seguinte forma:




```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

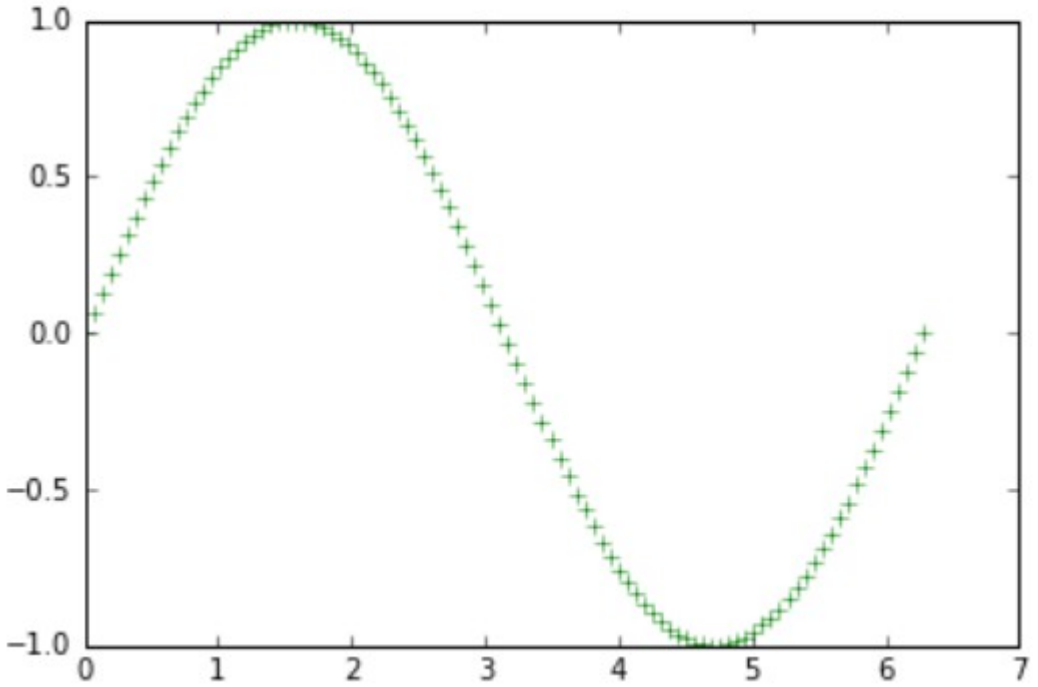
2D Plot

- plt.plot(X, Y, OPT)
- semelhante a sintaxe utilizada pelo Matlab
- pode passar múltiplos valores X, Y, OPT

```
x = np.linspace(0, 2 * np.pi, 100)
```

```
y = np.sin(x)
```

```
plt.plot(x,y, "g+")
plt.show()
```

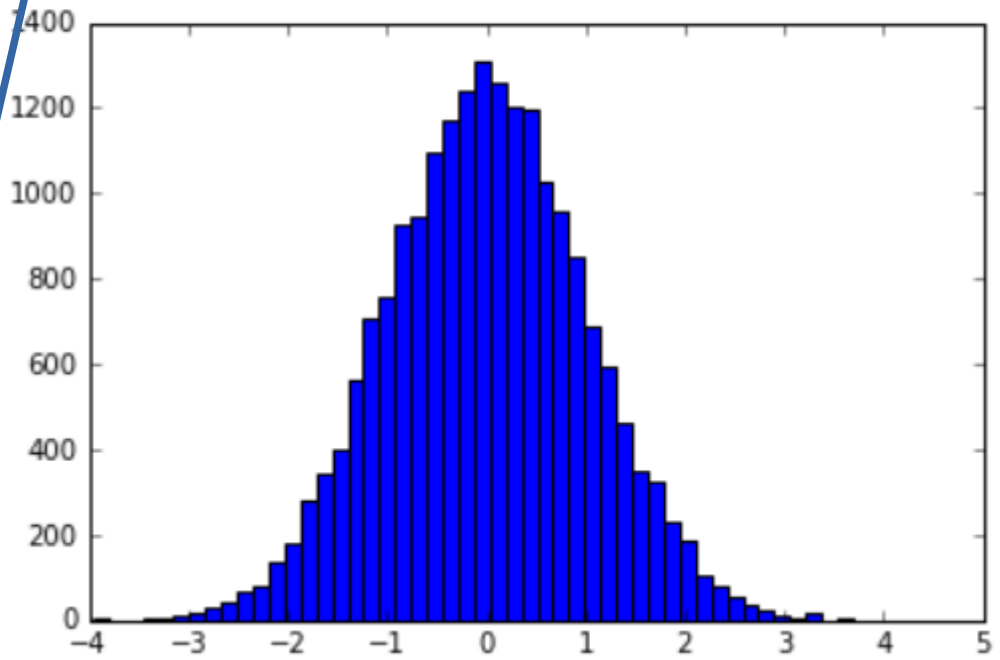


Histogramas

- Calcula e plota o histograma !
- Retorna os bins calculados

```
x = np.random.randn(20000)
```

```
b,xb,p = plt.hist(x, bins=50)
plt.show()
```



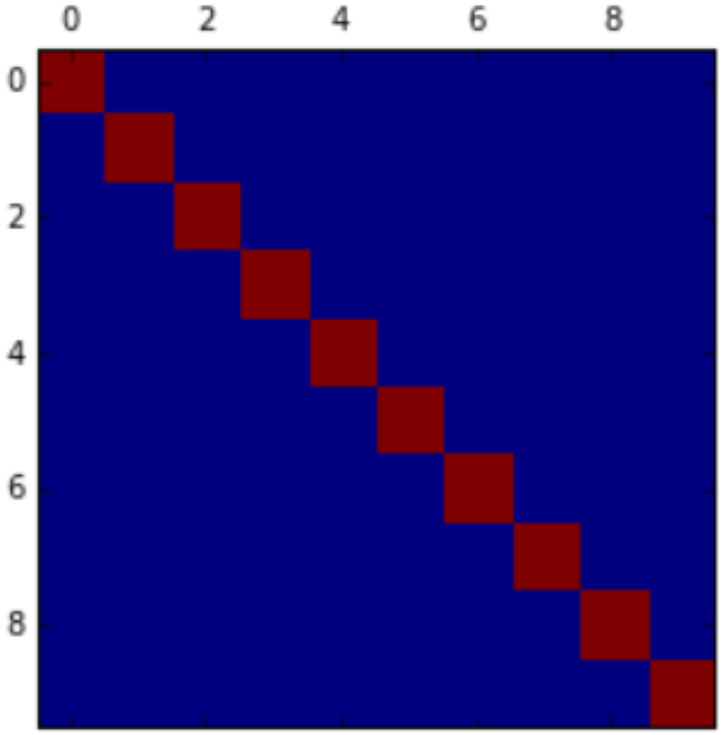
Onde:
b são os bins
xb a posição do bin

Matrizes

- Um vetor 2-dimensional (ou uma matriz) pode ser visualizado como uma imagem !

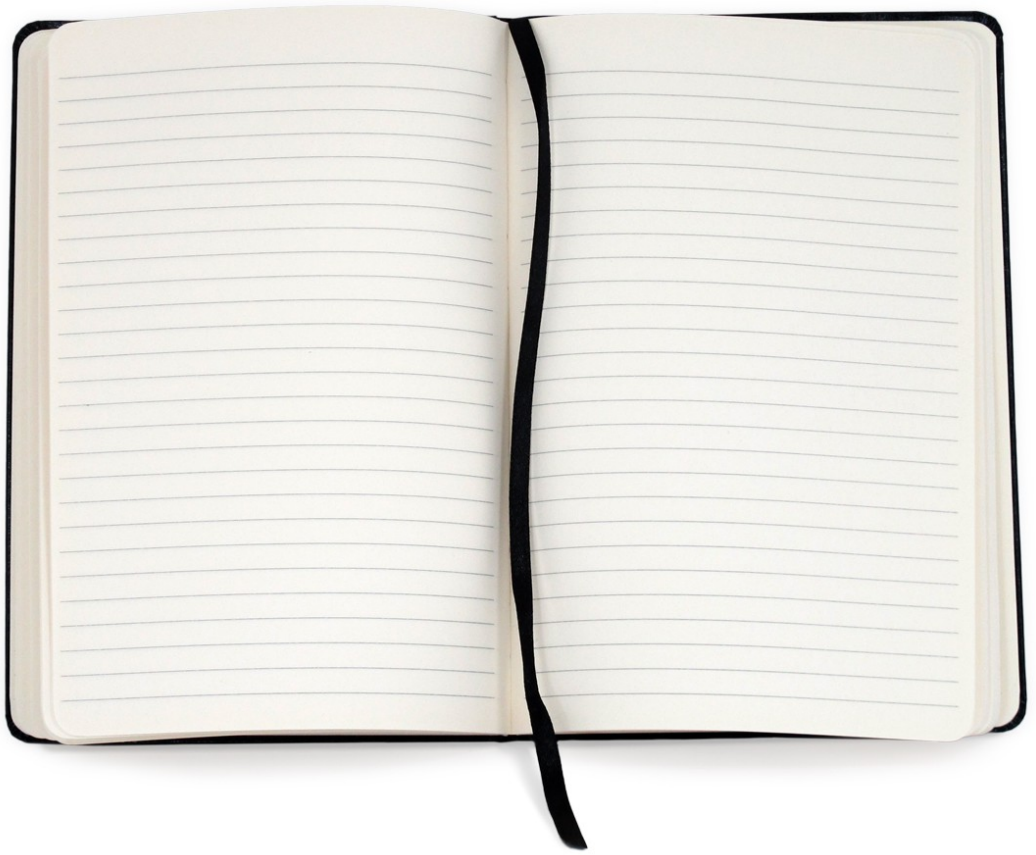
```
m = np.zeros((10,10))
m[range(0,10),range(0,10)] = 10
plt.matshow(m)
plt.plot()
```

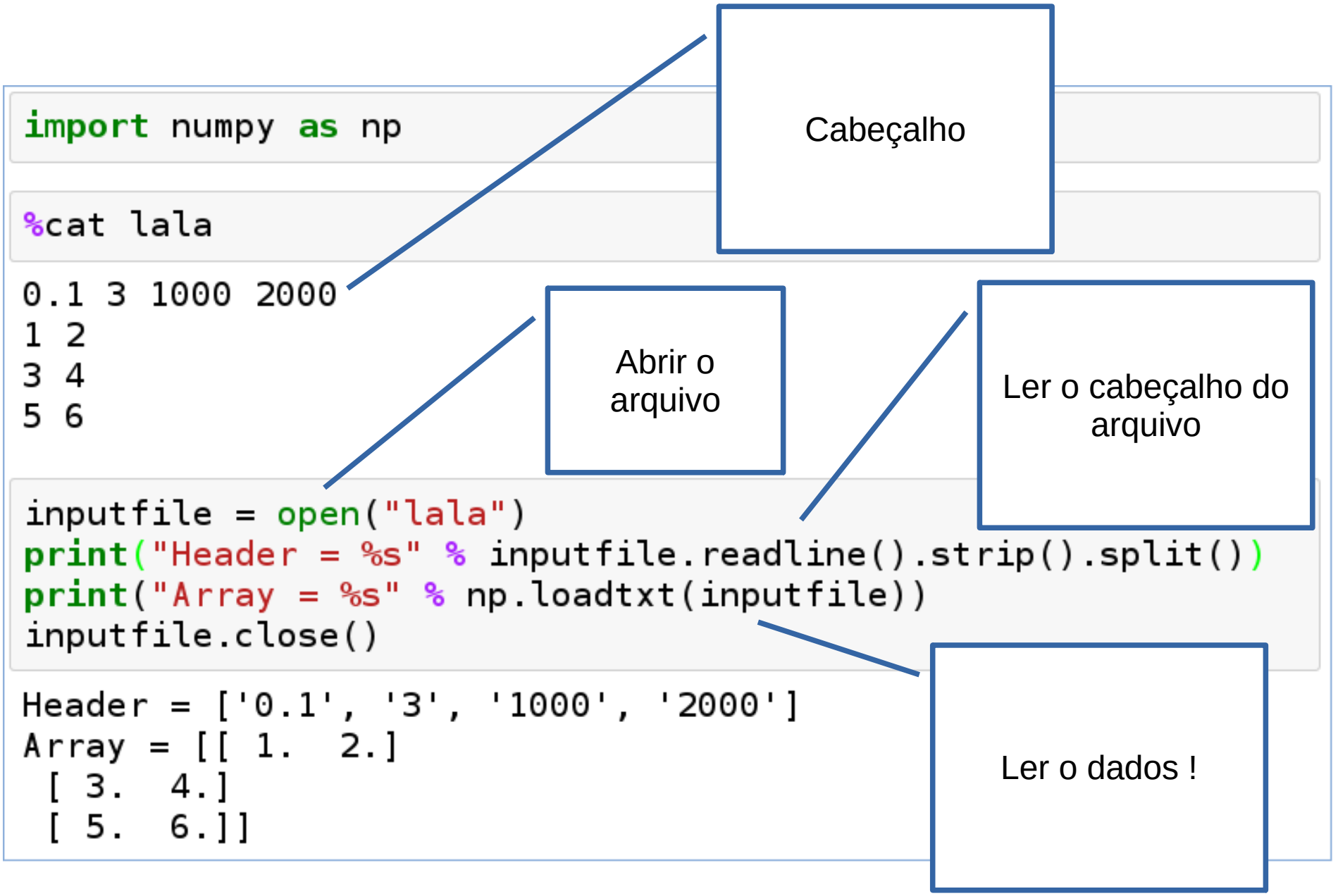
```
[]
```



Prática

- Trabalhe no notebook, 01-NumpyBasico




import numpy as np
%cat lala
0.1 3 1000 2000
1 2
3 4
5 6
inputfile = open("lala")
print("Header = %s" % inputfile.readline().strip().split())
print("Array = %s" % np.loadtxt(inputfile))
inputfile.close()
Header = ['0.1', '3', '1000', '2000']
Array = [[1. 2.]
 [3. 4.]
 [5. 6.]]
Cabeçalho
Abrir o arquivo
Ler o cabeçalho do arquivo
Ler o dados !