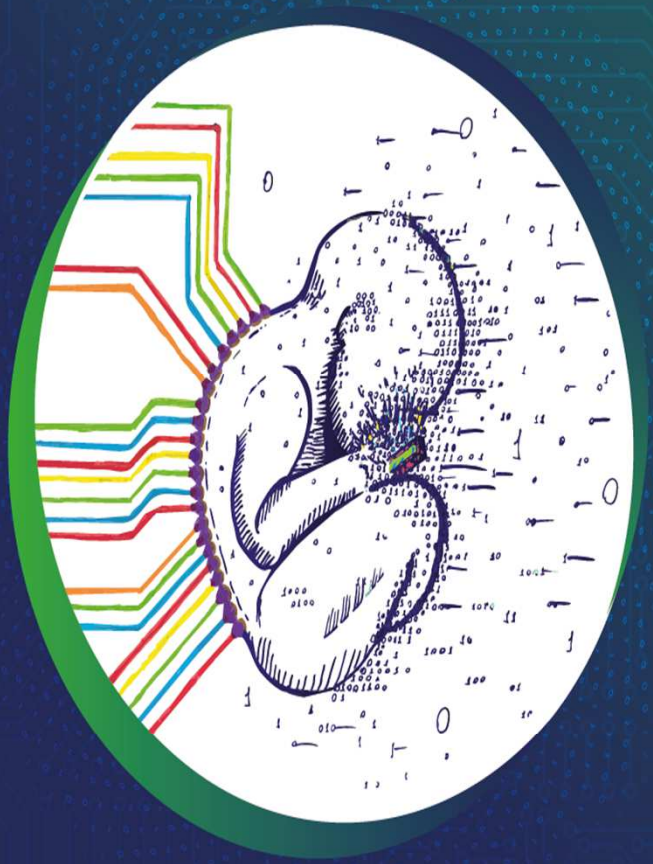
MÊS NACIONAL DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES
OUTUBRO | MCTI



17ª SEMANA NACIONAL

DE CIÊNCIA E TECNOLOGIA

INTELIGÊNCIA ARTIFICIAL:
A NOVA FRONTEIRA DA CIÊNCIA BRASILEIRA

#SNCTMCTI

EDIÇÃO 2020

INPE
UNIDADE DE PESQUISA DO MCTI

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES

PÁTRIA AMADA BRASIL
GOVERNO FEDERAL

17ª SEMANA NACIONAL DE CIÊNCIA E TECNOLOGIA
Inteligência Artificial: A Nova Fronteira da Ciência Brasileira

MÊS NACIONAL DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES
OUTUBRO | MCTI

Matemática com Python - Usando o Numpy, Matplotlib e Scipy

Lázaro Aparecido Pires de Camargo
Divisão de Pequenos Satélites - DPST
Instituto Nacional de Pesquisas Espaciais – INPE
São José dos Campos - SP

INPE
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES
PÁTRIA AMADA BRASIL GOVERNO FEDERAL

Contéudo:

- Uma breve introdução a linguagem Python
- Numpy e arrays
- Matplotlib e gráficos
- SciPy e funções básicas

Vamos usar o Winpython

http://winpython.github.io/

É uma distribuição Python, com diversos pacotes pré-instalados:

- Numpy
- Matplolib
- Scipy
- Pandas
- Cartopy,Astropy
- Keras, Pyserial
- Scikit_image, Scikit_learn

Também podemos usar o Repl.it (online)

Podemos também usar...

Github:

https://github.com/lazarocamargo/Matematica_com_Python

Podemos também usar...

Revisão – Linguagem Python

1) Variáveis em Python:

```
>>> uma_string = "oi gente"
>>> um_inteiro = 12
>>> um_float = 3.14
>>> booleano_verdadeiro = True
>>> booleano_falso = False
```

Revisão – Linguagem Python

2) Print para variaveis ou constantes:

```
>>> print("oi gente")
oi gente
>>> print(12)
12
>>> print((5+3)/2)
4.0
```

Revisão – Linguagem Python

Print com varios valores, separar com virgula:

```
>>> print("abc", 12, um_float)
abc 12 3.14
```

Comentários em Python:

```
>>> # comentarios iniciam com "#"
```

Linguagem Python

Python 3 Cheat Sheet

Latest version on :
https://perso.limsi.fr/pointal/python:memento

Base Types

integer, float, boolean, string, bytes

```
   int 783   0 -192   0b010 0o642 0xF3
                zero   binary octal hexa
float 9.23 0.0   -1.7e-6
                          ×10^-6
 bool True  False
  str "One\nTwo"        Multiline string:
        escaped new line  """X\tY\tZ
      'I\'m'              1\t2\t3"""
       escaped '           escaped tab
bytes b"toto\xfe\775"
          hexadecimal  octal
```

immutables

Container Types

- **ordered sequences**, fast index access, repeatable values

```
      list [1,5,9]    ["x",11,8.9]      ["mot"]      []
    tuple (1,5,9)     11,"y",7.4        ("mot",)     ()
Non modifiable values (immutables)   expression with only comas → tuple
      str bytes  (ordered sequences of chars / bytes)      ""  b""
```

- **key containers**, no *a priori* order, fast key access, each key is unique

```
dictionary   dict {"key":"value"}   dict(a=3,b=4,k="v")   {}
(key/value associations) {1:"one",3:"three",2:"two",3.14:"π"}
collection   set {"key1","key2"}    {1,9,3,0}            set()
```

keys=hashable values (base types, immutables…) frozenset immutable set *empty*

Identifiers

for variables, functions, modules, classes… names

a…zA…Z_ followed by a…zA…Z_0…9

- diacritics allowed but should be avoided
- language keywords forbidden
- lower/UPPER case discrimination

```
☺ a toto x7 y_max BigOne
☹ 8y and for
```

Variables assignment

=

assignment ⇔ **binding** of a *name* with a *value*

1) *evaluation of right side expression value*
2) *assignment in order with left side names*

```
x=1.2+8+sin(y)
a=b=c=0          assignment to same value
y,z,r=9.2,-7.6,0 multiple assignments
a,b=b,a          values swap
a,*b=seq  }  unpacking of sequence in
*a,b=seq  }  item and list
x+=3   increment ⇔ x=x+3      and *= /= %= …
x-=2   decrement ⇔ x=x-2
x=None « undefined » constant value
del x  remove name x
```

Conversions

type(*expression*)

```
int("15") → 15
int("3f",16) → 63        can specify integer number base in 2nd parameter
int(15.56) → 15          truncate decimal part
float("-11.24e8") → -1124000000.0
round(15.56,1)→15.6      rounding to 1 decimal (0 decimal → integer number)
bool(x)  False for null x, empty container x , None or False x ; True for other x
str(x)→ "…"   representation string of x for display (cf. formatting on the back)
chr(64)→'@'   ord('@')→64      code ↔ char
repr(x)→ "…"  literal representation string of x
bytes([72,9,64]) → b'H\t@'
list("abc") → ['a','b','c']
dict([(3,"three"),(1,"one")]) → {1:'one',3:'three'}
set(["one","two"]) → {'one','two'}
separator str and sequence of str → assembled str
   ':'.join(['toto','12','pswd']) → 'toto:12:pswd'
str splitted on whitespaces → list of str
   "words with  spaces".split() → ['words','with','spaces']
str splitted on separator str → list of str
   "1,4,8,2".split(",") → ['1','4','8','2']
sequence of one type → list of another type (via list comprehension)
   [int(x) for x in ('1','29','-3')] → [1,29,-3]
```

Revisão – Linguagem Python

Listas

São conjuntos ordenados de valores, são criadas com colchetes [] :

```
>>> uma_lista = [1, 2, 3, 'abc', 'def']
>>> print(uma_lista)
[1, 2, 3, 'abc', 'def']
```

Listas de listas:

```
>>> outra_lista = [uma_lista, 'abc', uma_lista,[1, 2, 3]]
>>> print outra_lista
[[1, 2, 3, 'abc', 'def'], 'abc', [1, 2, 3, 'abc', 'def'], [1, 2, 3]]
```

Revisão – Linguagem Python

Acessando um elemento de uma lista pelo indice (indices iniciam em zero):

```
>>> elem = uma_lista[2]
>>> print(elem)
3
>>> elem2 = outra_lista[3][1]
>>> print(elem2)
```

Alguns métodos do tipo Lista

L.append(elemento) - Insere o elemento no fim da lista.
L.append([posição]) - Retira o último elemento da lista.
L.sort() - Ordena a lista
L.remove(elemento) - remove a primeira ocorrência do elemento na lista
L.count(elemento) - Conta o número de aparições do elemento na lista.
L.insert(posição,elemento) - Insere na posição informada o elemento.
L.reverse() - Inverte a lista.

Revisão – Linguagem Python

Fatiando uma lista:

```
>>> lista_2 = uma_lista[2:4] #retorna uma lista com os itens 2 e 3
>>> print lista_2
[3, 'abc']
```

Revisão – Linguagem Python

Adicionando e removendo itens de uma lista:

```
>>> uma_lista.append('ghi')
>>> uma_lista.remove('abc')
>>> print uma_lista
[1, 2, 3, 'def', 'ghi']
```

Revisão – Linguagem Python

Tuplas

São similares a listas, mas imutáveis.

```
>>> uma_tupla = (1, 2, 3, 'abc', 'def')
>>> print(uma_tupla)
(1, 2, 3, 'abc', 'def')

>>> outra_tupla = 1, 2, 3, 'abc', 'def'
>>> print(outra_tupla)
(1, 2, 3, 'abc', 'def')
```

Revisão – Linguagem Python

Blocos e indentação

Blocos de código são delimitados utilizando indentação.

if / elif / else (salve como modulo_if.py e execute):

```
a = 3
b = 1

if a == 3:
    print('O valor e')
    print('a = 3')

if a == 'teste':
    print('O valor e')
    print('a = "teste"')
    modo_teste = True
```

Revisão – Linguagem Python

```
else:
    print('a != "teste"')
    modo_teste = False

if a == 1 or a == 2:
    pass #nao faz nada

elif a ==3 and b > 1:
    pass

elif a == 3 and not b > 1:
    pass

else:
    pass
```

Revisão – Linguagem Python

while loops (salve como modulo_while.py):

```
a = 1
while a < 10:
    print(a)
    a += 1    # a = a + 1
```

Revisão – Linguagem Python

for loops (salve como modulo_for_1.py):

```
for a in range(10):
    print(a)
```

Revisão – Linguagem Python

Agora salve como modulo_for_2.py, e execute:

```
minha_lista = [2, 4, 8, 16, 32]
for a in minha_lista:
    print(a)
```

Revisão – Linguagem Python

Funções

Uma função é definida com a palavra reservada "def":

(salve como modulo_funcao_1.py) e execute:

```
def minha_funcao(arg1, arg2, arg3='valor default'):
    print('arg1 = ', arg1)
    print('arg2 = ', arg2)
    print('arg3 = ', arg3)
```

Revisão – Linguagem Python

Uma função pode retornar um valor:

```
>>> def fun2():
...     print('fun2')
...     return 'retorna qualquer valor'
...
>>> print('fun2() = %s' % fun2())
fun2
fun2() = retorna qualquer valor
```

Revisão – Linguagem Python

```
>>> def dados_pessoais():
...     return ('Maria', 25)
...
>>> print('nome = %s e idade = %s' % dados_pessoais())
nome = Maria e idade = 25
```

Revisão – Linguagem Python

Arquivos

Crie um módulo com o nome "arquivo1.py", execute, e verifique se foi criado o arquivo meu_arquivo.txt:

```
uma_string = 'oi amigos'

fw = open('meu_arquivo.txt','w')    # abre o arquivo para escrita


fw.write('uma linha de texto\n')    # escreve no arquivo
fw.write(uma_string)

fw.close()                          # fecha o arquivo
```

Revisão – Linguagem Python

Para ler o arquivo todo:

Crie um módulo com o nome "le_arquivo2.py" e execute:

```
f = open("meu_arquivo.txt")      # abre o arquivo para leitura

print(f.read())                  # mostra o arquivo todo

f.close()                        # fecha arquivo
```

Revisão – Linguagem Python

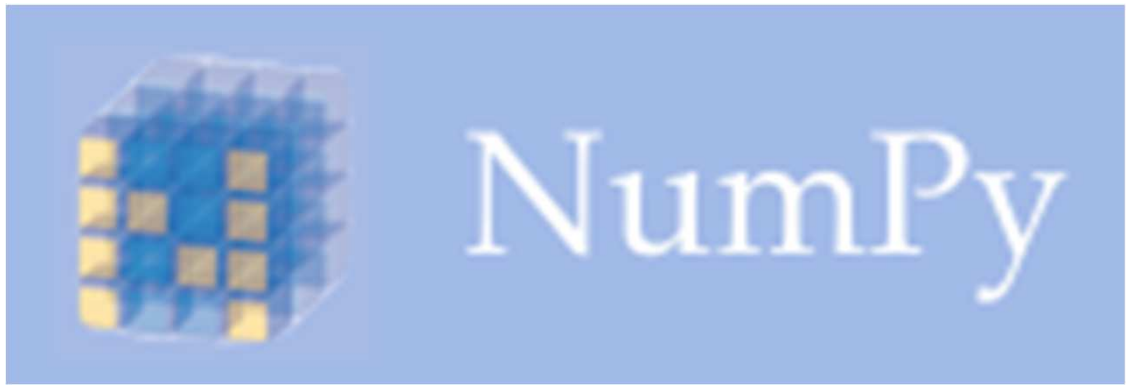
Escrever uma lista em um arquivo:

```
>>> lista = range(10)              # lista [0,..., 9]

>>> f = open('minha_lista.txt','w')
>>> for item in lista:
...     f.write("%s\n" % item)
...
>>> f.close()
```

Revisão – Linguagem Python

Lendo um arquivo e armazenando em uma lista:

```
>>> lista1=[]

>> f = open('minha_lista.txt','r')
>>> for val in f.read().split():
...     lista1.append(int(val))

>>> f.close()
>>> lista1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```



É um pacote para a linguagem Python, que permite trabalhar com vetores e matrizes, com múltiplas dimensões.

https://numpy.org/

1995 – Numeric
2006 - Numpy

Sub-pacotes disponíveis em Numpy:

core: define arranjos multidimensionais e funções paracomputação numérica

random: geradores de números aleatórios e funções relacionadas

linalg: ferramentas para algebra linear

fft: rotinas para transformada de Fourier

Python simples: multiplicando duas arrays de tamanho 10.000.000 (python_lista.py)

```
import time

t = 10000000

inicio =  time.time()
a, b = range(t), range(t)
c = []

for i in a:
    c.append(a[i] * b[i])

tempo = time.time() - inicio
print('Duração: %s' %tempo)
```

Usando numpy: multiplicando duas arrays de tamanho 10.000.000 (array1.py)

```
import numpy as np
import time

t = 10000000

inicio = time.time()

a = np.arange(t)
b = np.arange(t)
c = a * b

tempo = time.time() - inicio
print('Duração: %s' %tempo)
```

Documentação de referência

http://docs.scipy.org

Ajuda interativa:

```
help(np.array)
```

Procura por algo específico:

```
np.lookfor('fft')
```

Criando uma array simples

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

>>> a.ndim
1
>>> a.shape
(4,)
>>> len(a)
4
```

Criando uma array bidimensional:

```
>>> b = np.array([[0, 1, 2],[3, 4, 5]])
>>> b
array([[0, 1, 2],
       [3, 4, 5]])
>>> b.ndim
2
>>> b.shape
(2, 3)
>>> len(b)
2
```

Criando array com valores com espaços iguais

```
>>> a = np.arange(10)   # 0..n-1
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])


>>> b = np.arange(1, 9, 2)  #inicio, fim, passo
>>> b
array([1, 3, 5, 7])
```

Criando por número de pontos

```
>>> c = np.linspace(0, 1, 5)     #inicio,fim,npt
>>> c
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ])
```

```
>>> d = np.linspace(0, 1, 5, endpoint=False)
>>> d
array([ 0. ,  0.2,  0.4,  0.6,  0.8])
```

Matrizes comuns:

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


>>> b = np.zeros((2,2))
>>> b
array([[ 0.,  0.],
       [ 0.,  0.]])
```

Matriz identidade e diagonal

```
>>> np.eye(3)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

```
>>> np.diag(np.array([1, 2, 3, 4]))
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

Números aleatórios

```
>>> a = np.random.rand(4)   # uniforme [0, 1]
>>> a

>>> a = np.random.randn(4) # gaussiana
>>> a


>>> np.random.seed(1234)    # def. inicio
```

Tipos básicos de dados

```
>>> a = np.array([1, 2, 3])
>>> a.dtype
dtype('int32')
```

```
>>> b = np.array([1., 2., 3.])
>>> b.dtype
dtype('float64')
```

Pode-se especificar o tipo de dado

```
>>> c = np.array([1,2,3], dtype=float)

>>> c
array([ 1.,  2.,  3.])

>>> c.dtype
```

Também existem outros tipos:

Complexo

```
>>> d = np.array([1+2j, 3+4j])
>>> d.dtype
```

Booleano

```
>>> e = np.array([True, False])
>>> e.dtype
```

Strings

```
>>> f = np.array(['oi', 'ola'])
>>> f.dtype
```

Indexando e cortando

Os itens de uma array (matriz) podem ser acessados e atribuidos da mesma forma que outras sequências do Python, como listas por exemplo:

```
>>> a = np.arange(10)
>>> a

>>> a[0]

>>> a[2]

>>> a[-1]
```

Importante: os índices começam em zero no numpy.

Para arrays multidimensionais

```
>>> a = np.diag(np.arange(3))

>>> a

>>> a[1, 1]

>>> a[2, 1]

>>> a[2, 1] = 10 # 3a linha, 2a coluna

>>> a

>>> a[1]
```

Cortando arrays

Arrays, como qualquer outra sequência em Python, podem ser cortadas:

```
>>> a = np.arange(10)

>>> a

>>> a[2:9:3]    # [inicio:fim:incremento]
```

Cortando arrays

Como em qualquer sequências do Python, o último índice não é incluido:

```
>>> a[:4]
```

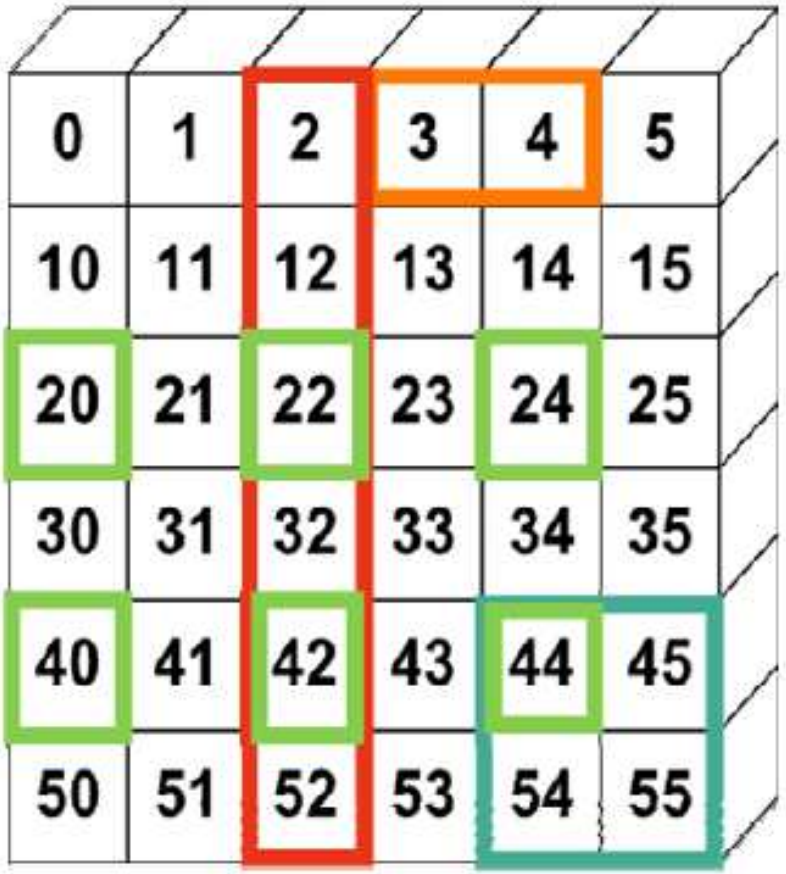
Cortando arrays

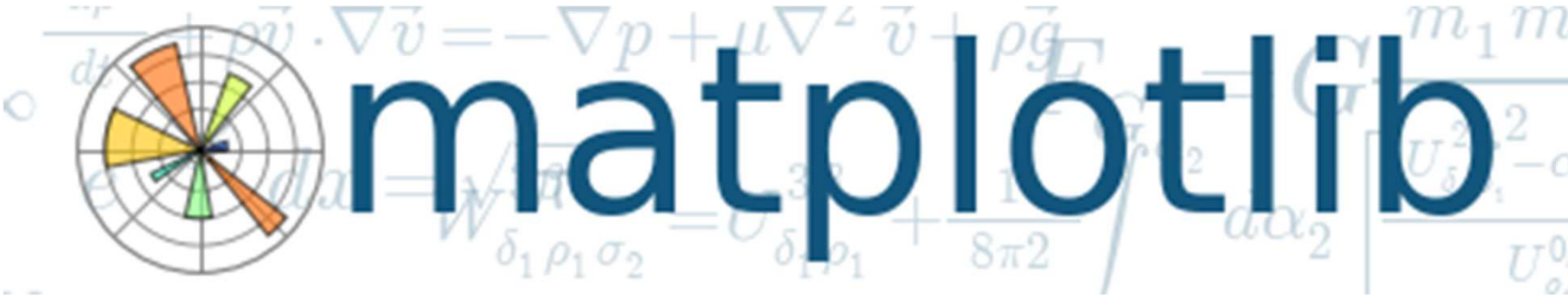
Nenhum dos três componentes para o corte são necessário: por padrão, o início é 0, o final é o último e o incremento é 1:

```
>>> a[1:3]

>>> a[::2]

>>> a[3:]
```

Resumo de indexação e corte de arrays:

```
>>> a[0,3:5]
array([3,4])


>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



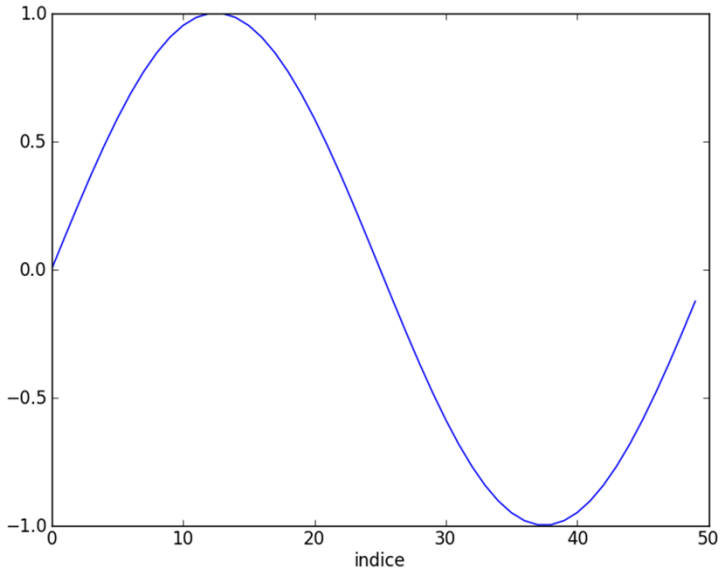


É um pacote para Python, para plotagem de gráficos.

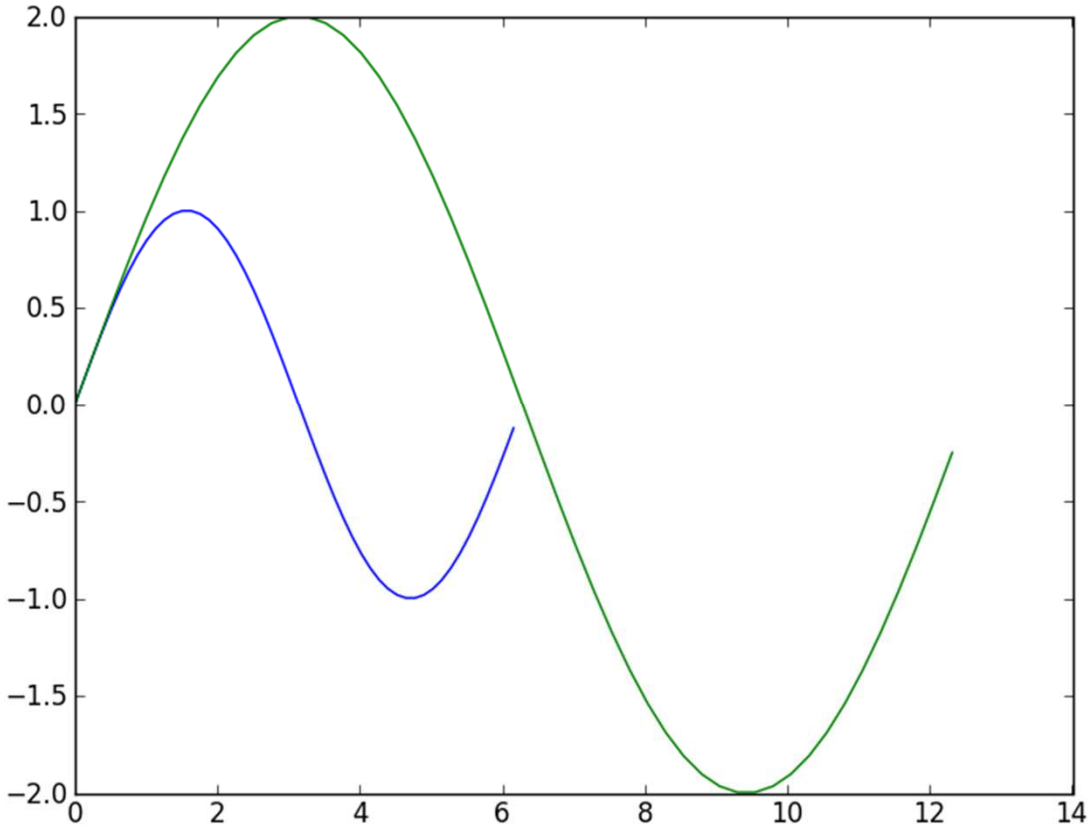
Início: 2003

Plotagem simples:

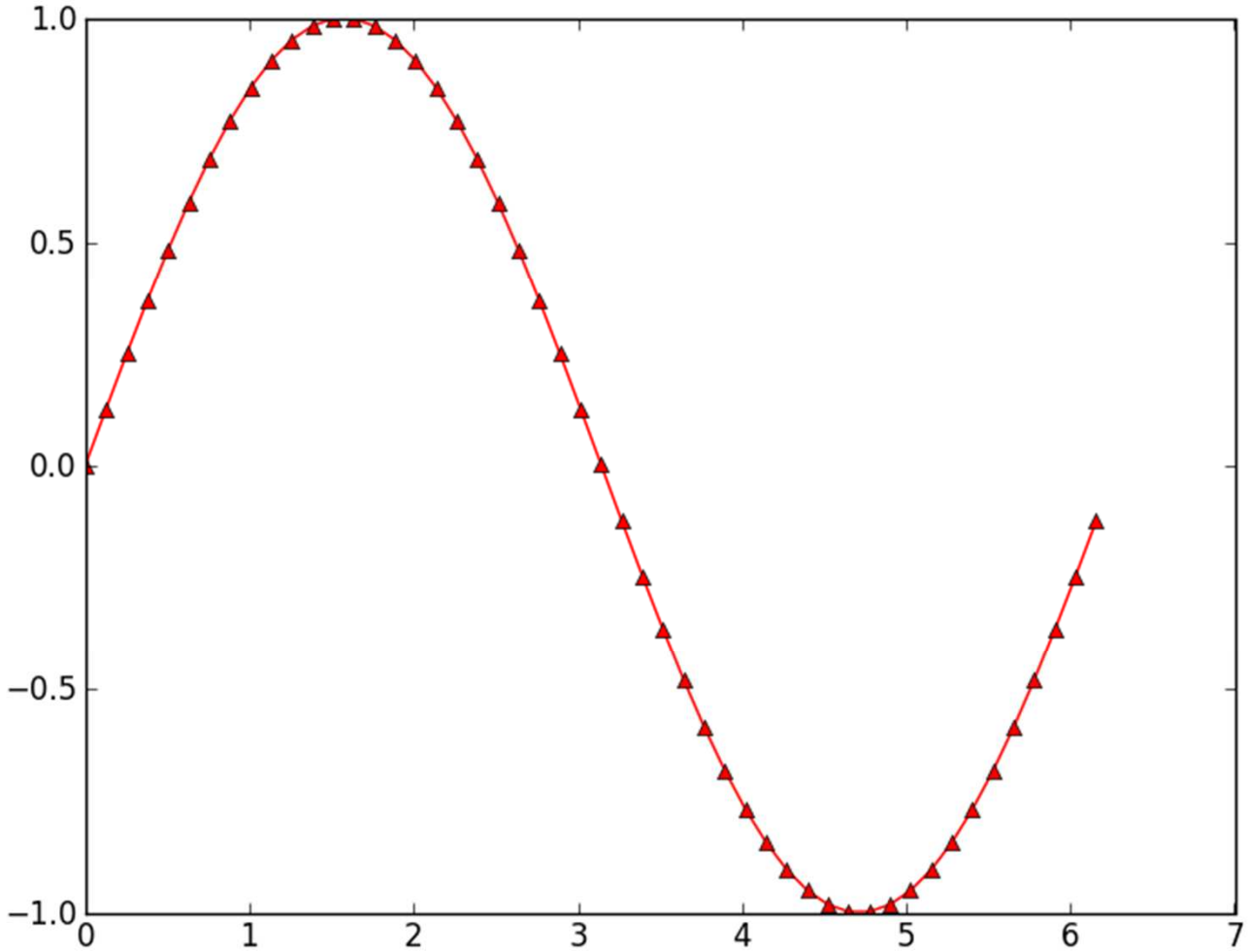
```
>>> import numpy as np
>>> import matplotlib.pyplot as plt


>>> x = (np.arange(50))*2*(np.pi)/50

>>> y = np.sin(x)

>>> plt.plot(y)

>>> plt.xlabel('indice')

>>> plt.show()
```



```
>>> plt.plot(x,y,x*2,y*2)

>>> plt.show()
```

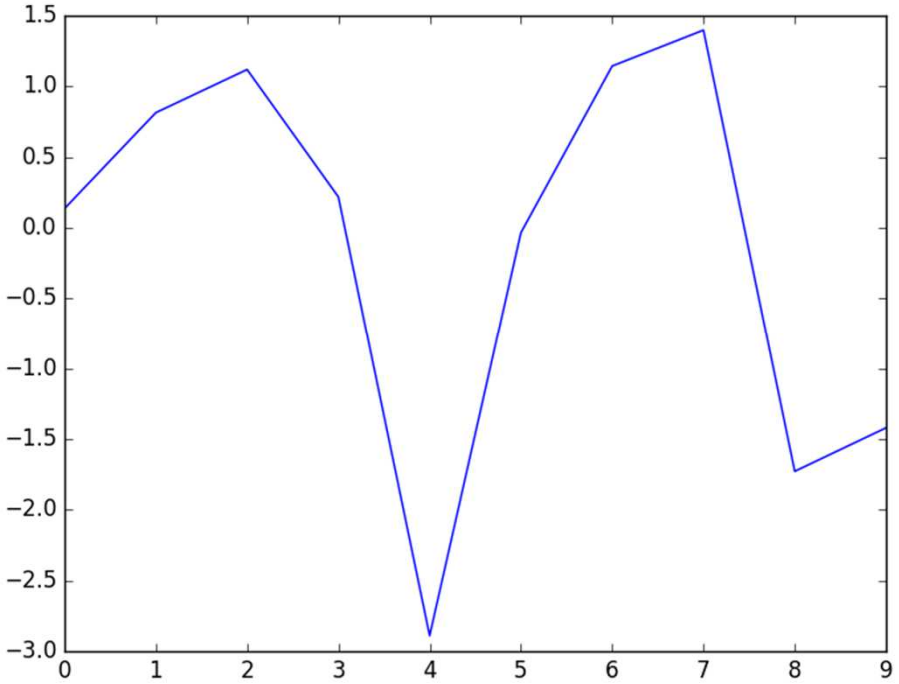


```
>>> plt.plot(x,y,'r-^')

>>> plt.show()
```



Uma plotagem simples

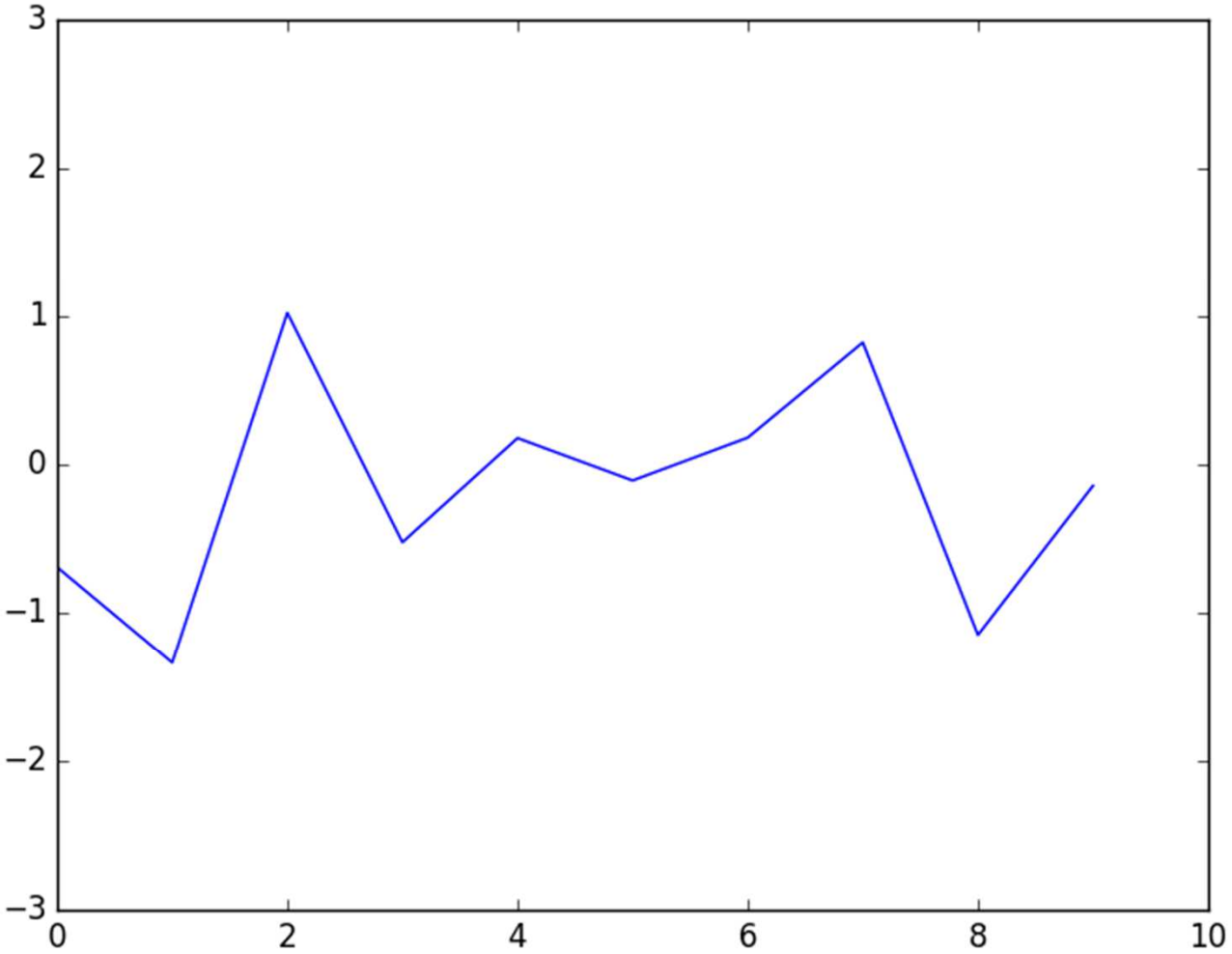
```
>>> x = np.random.randn(10)   # gera 10 valores com
                              # dist. gaussiana

>>> plt.plot(x)

>>> plt.show()
```



Uma plotagem simples - axis

```
>>> x = np.random.randn(10)

>>> plt.plot(x)

>>> plt.show(x)

>>> plt.axis([0, 10, -3, 3]) # limites do gráfico

>>> plt.plot(x)

>>> plt.show(x)
```

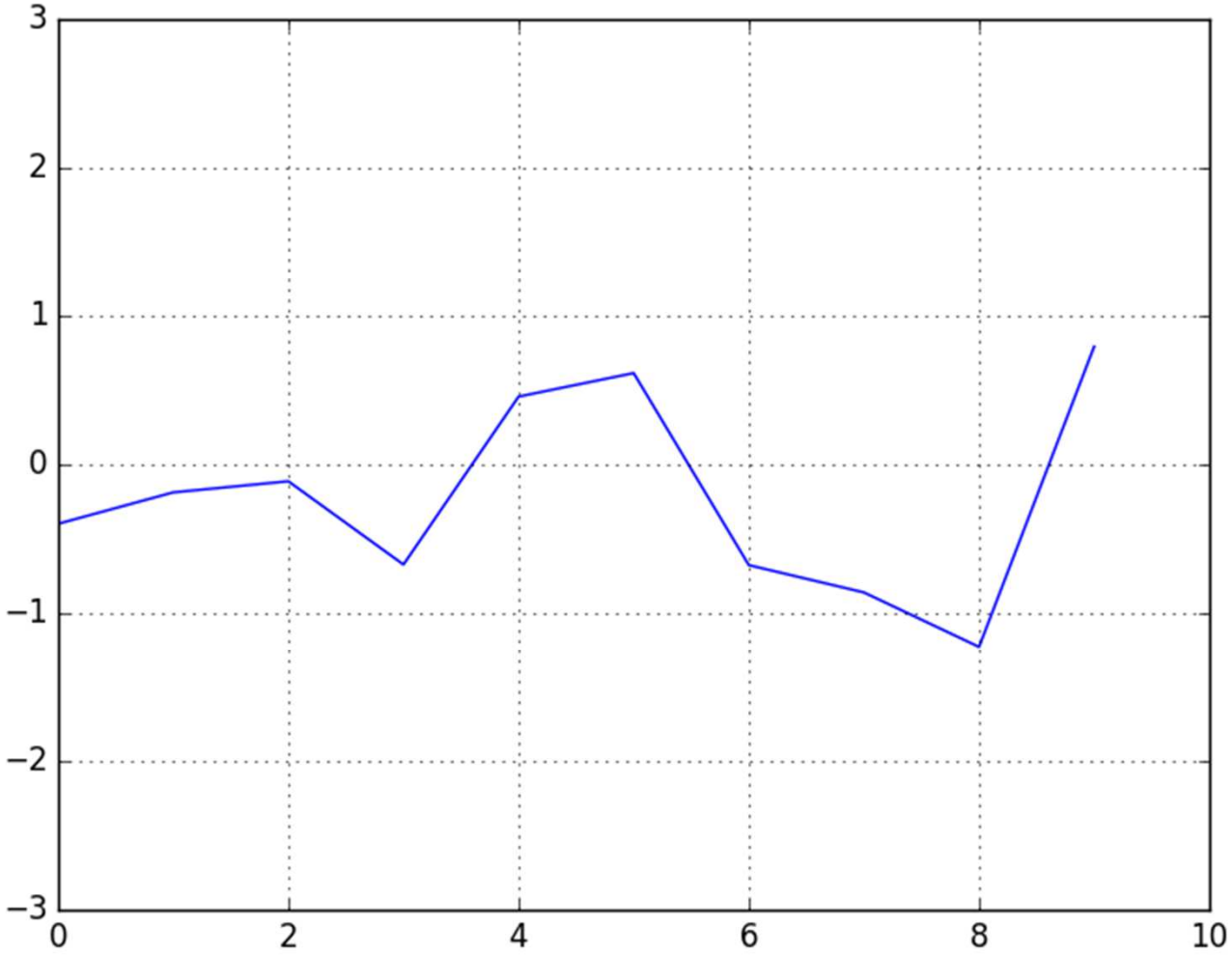
Uma plotagem simples - axis



Uma plotagem simples – grid

```
>>> x = np.random.randn(10)

>>> plt.axis([0, 10, -3, 3])

>>> plt.grid() # coloca grid

>>> plt.plot(x)

>>> plt.show()
```

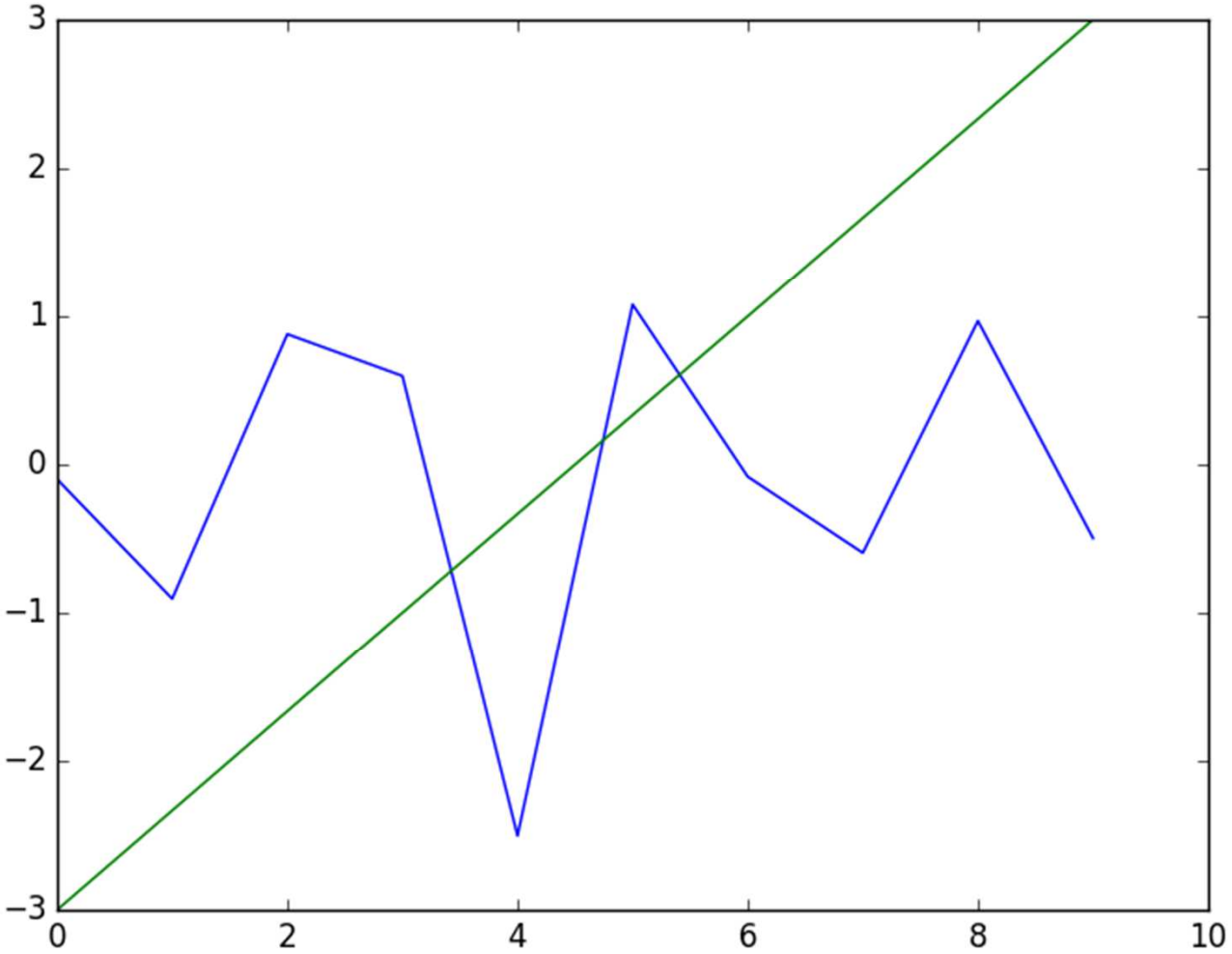
Uma plotagem simples - grid



Uma plotagem simples – adicionando outra plotagem

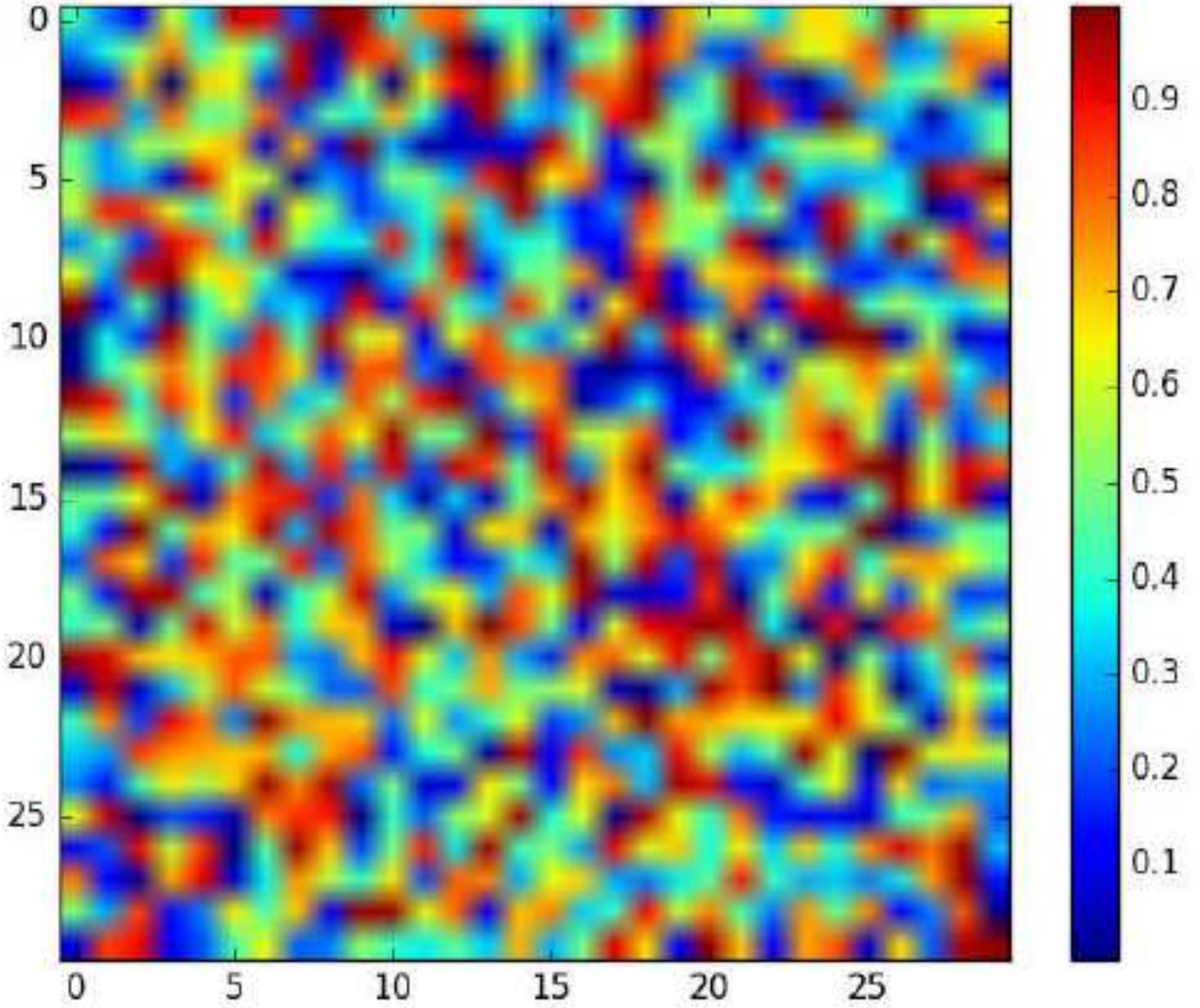
```
>>> x = np.random.randn(10)

>>> plt.axis([0, 10, -3, 3])

>>> plt.plot(x)

>>> y = np.linspace(-3,3,10)

>>> plt.plot(y)

>>> plt.show()
```

Uma plotagem simples – adicionando outra plotagem



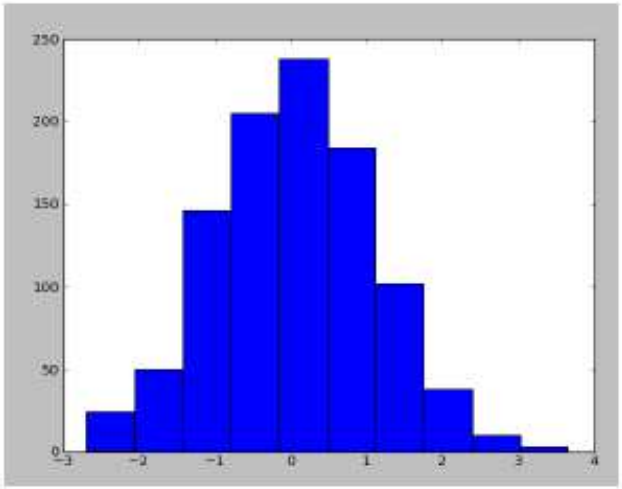
Plotagem 2-D

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt

>>> imagem = np.random.rand(30, 30)

>>> plt.imshow(imagem, cmap=plt.cm.jet)

>>> plt.colorbar()

>>> plt.show()
```


0
5
10
15
20
25
0
5
10
15
20
25
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

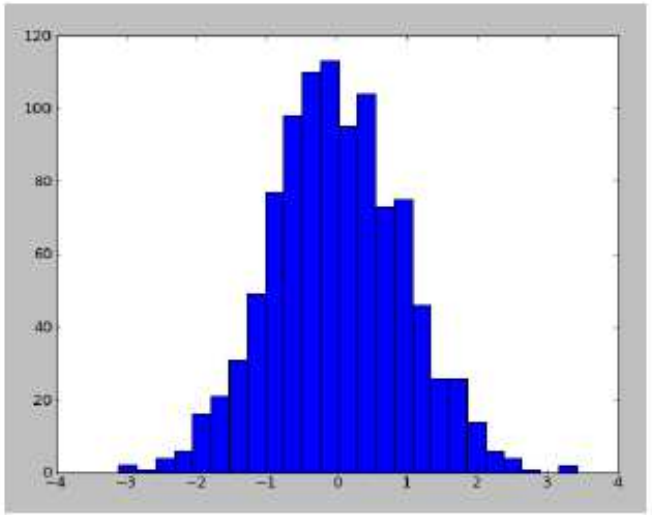
Histogramas

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> y = np.random.randn(1000)
>>> plt.hist(y)
(array([ 24,  50, 146, 205, 238, 184, 102,  38,  10,   3]),
 array([-2.69043671, -2.05660623, -1.42277575, -0.78894526, -0.15511478,
        0.4787157 ,  1.11254618,  1.74637667,  2.38020715,  3.01403763,
        3.64786811]),
 <a list of 10 Patch objects>)
>>> plt.show()
```



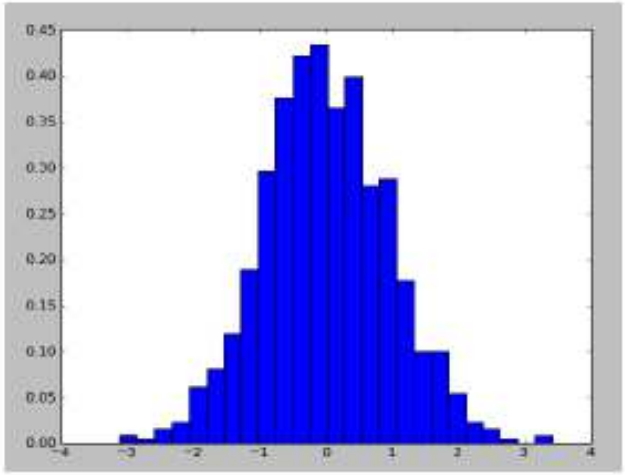
Histogramas

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> y = np.random.randn(1000)
>>> plt.hist(y, 25)
(array([  2,   1,   4,   6,  16,  21,  31,  49,  77,  98, 110, 113,  95,
        104,  73,  75,  46,  26,  26,  14,   6,   4,   1,   0,   2]),
 array([-3.10254339, -2.84183405, -2.58112471, -2.32041536, -2.05970602,
        -1.79899667, -1.53828733, -1.27757798, -1.01686864, -0.7561593 ,
        -0.49544995, -0.23474061,  0.02596874,  0.28667808,  0.54738743,
         0.80809677,  1.06880611,  1.32951546,  1.5902248 ,  1.85093415,
         2.11164349,  2.37235284,  2.63306218,  2.89377152,  3.15448087,
         3.41519021]),
 <a list of 25 Patch objects>)
>>> plt.show()
```



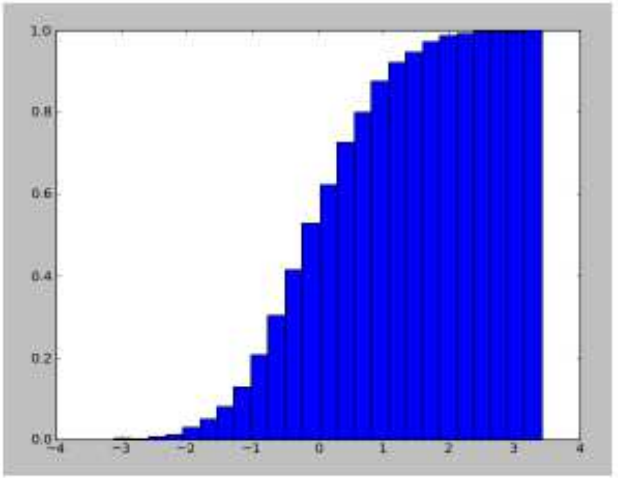
Normalizando o histograma (a area total do histograma é 1)

```
>>> plt.hist(y, 25, normed=True)
>>> plt.show()
```



Plotando o histograma de forma cumulativa:

```
>>> plt.hist(y, 25, normed=True, cumulative=True)
>>> plt.show()
```



É uma biblioteca para Python, que trabalha com arrays Numpy, e fornece rotinas para computação cientifica como integração numérica e otimização.

Inicio: 2001

Scipy

É um pacote Python com várias ferramentas dedicadas a problemas em computação científica.

Possui sub-módulos para:

*Interpolação
*Otimização
*Processamento de imagens
*Estatísticas
*Funções especiais

Scipy – alguns pacotes

scipy.cluster	Quantização de vetores (Kmean)
scipy.constants	Constantes matemáticas e físicas
scipy.fftpack	Tranformada de Fourier
scipy.integrate	Rotinas de Integração
scipy.interpolate	Interpolação
scipy.io	Entrada e saída de dados
scipy.linalg	Rotinas de algebra linear
scipy.ndimage	pacote para imagens n-dimensionais
scipy.odr	Regrassão ortogonal de distância
scipy.optimize	Otimização
scipy.signal	Processamento de sinal
scipy.sparse	Matrizes esparsas
scipy.spatial	Estruturas de dados espacial e algoritmos
scipy.special	Qualquer função matemática especial
scipy.stats	Estatística

SciPy - Integração

$$\int_0^4 x^2 \, dx$$

```
>>> import scipy.integrate as spint

>>> def f(x):
        return x**2

>>> spint.quad(f, 0, 4)
(21.333333333333336, 2.3684757858670001e-13)
```

SciPy - Integração

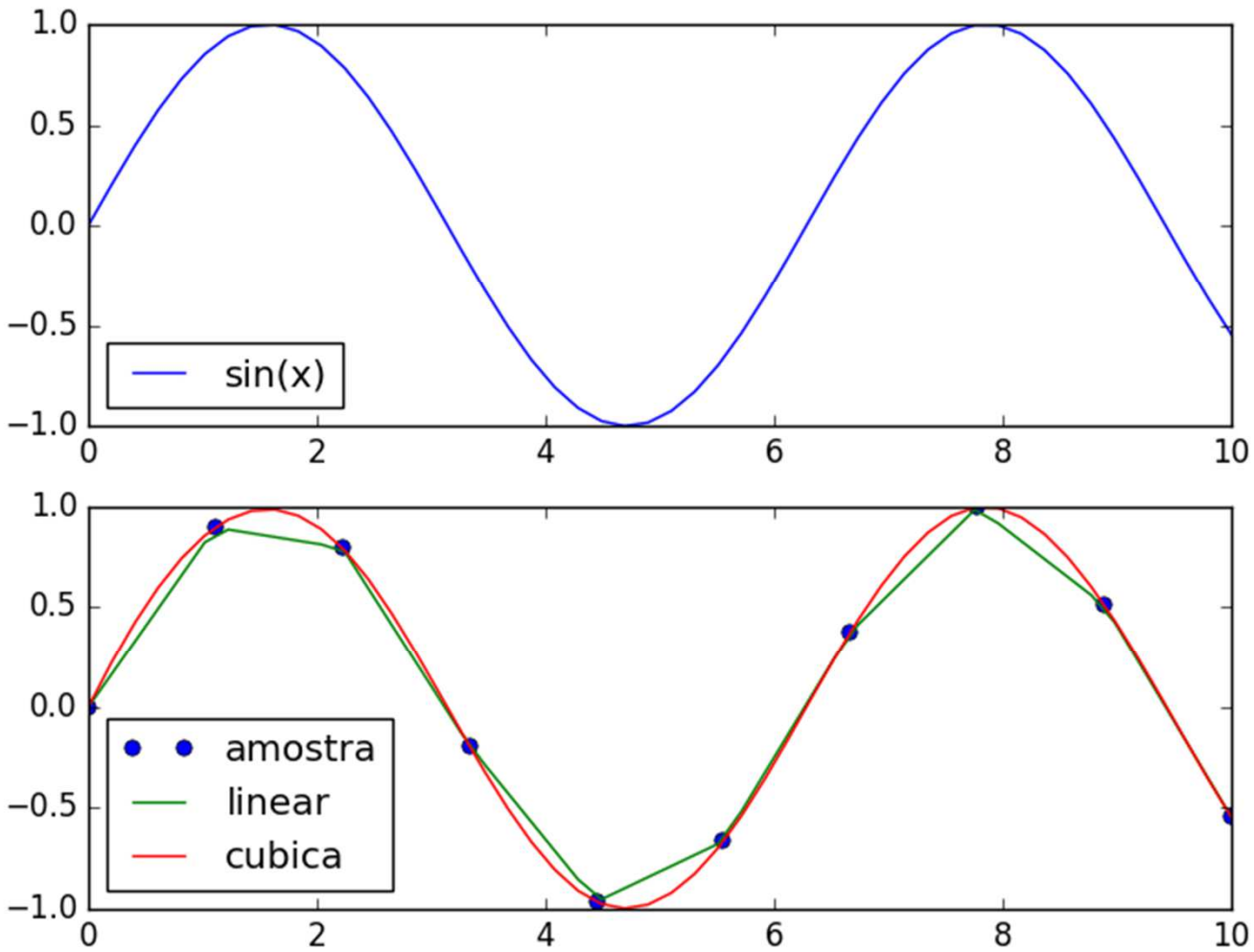
$$\int_0^\infty e^{-x} dx$$

```
>>> import numpy as np
>>> import scipy.integrate as spint

>>> def f(x):
        return np.exp(-x)

>>> spint.quad(f, 0, np.inf)
(1.000000000000002, 5.842607038578007e-11)
```

Scipy - Interpolação

```
import scipy.interpolate as sp
import numpy as np
import matplotlib.pyplot as plt


# 50 pontos de sin(x) no intervalo [0 10]
xx = np.linspace(0,10,50)
yy = np.sin(xx)


# 10 amostras de sin(x) no intervalo [0 10]
x = np.linspace(0, 10, 10)
y = np.sin(x)

# interpolacao
fl = sp.interp1d(x, y, kind='linear')
fc = sp.interp1d(x, y, kind='cubic')


# fl e fc sao os valores da interpolacao
# definidos no intervalo [0 10]
# fl utiliza interpolacao linear
# fc utiliza interpolacao cubica

xnew = np.linspace(0, 10, 50)
plt.subplot(2,1,1)

# plota funcao sin(x) (real)
plt.plot(xx,yy)
plt.legend(['sin(x)'], loc='best')

plt.subplot(2,1,2)
plt.plot(x,y,'o',xnew,fl(xnew), xnew, fc(xnew))
plt.legend(['amostra','linear','cubica'], loc='lower')

plt.show()
```

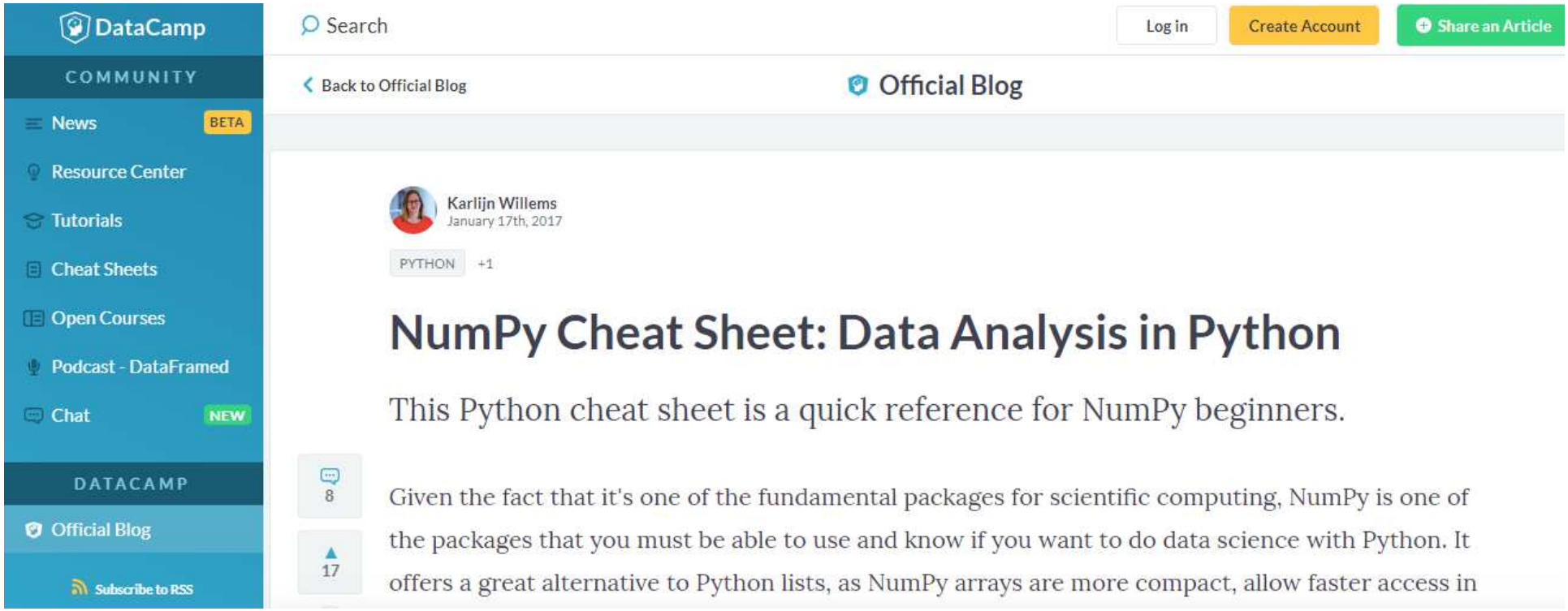
Scipy - Interpolação



Scipy – Números aleatórios

```
>>> import scipy as sp
>>> sp.rand(2,3)

>>> rnd = sp.randn(100)
>>> rnd.mean()
-0.26078817407199606

>>> sp.random.randint(1, 61, 6)
array([30, 41, 19, 14, 51, 38])

>>> sp.random.binomial(5, 0.4)
2
>>> sp.random.binomial(5, 0.4, 10)
array([2, 3, 2, 3, 0, 2, 1, 3, 1, 1])
```

Referências:



https://www.datacamp.com/community/data-science-cheatsheets

Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
 5
```

Calculations With Variables

```
>>> x+2
 7
>>> x-2
 3
>>> x*2
 10
>>> x**2
 25
>>> x%2
 1
>>> x/float(2)
 2.5
```

Sum of two variables
Subtraction of two variables
Multiplication of two variables
Exponentiation of a variable
Remainder of a variable
Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Subset

```
>>> my_list[1]
>>> my_list[-3]
```

Select item at index 1
Select 3rd last item

Slice

```
>>> my_list[1:3]
>>> my_list[1:]
>>> my_list[:3]
>>> my_list[:]
```

Select items at index 1 and 2
Select items after index 0
Select items before index 3
Copy my_list

Subset Lists of Lists

```
>>> my_list2[1][0]
>>> my_list2[1][:2]
```

my_list[list][itemOfList]

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

```
>>> my_list.index(a)
>>> my_list.count(a)
>>> my_list.append('!')
>>> my_list.remove('!')
>>> del(my_list[0:1])
>>> my_list.reverse()
>>> my_list.extend('!')
>>> my_list.pop(-1)
>>> my_list.insert(0,'!')
>>> my_list.sort()
```

Get the index of an item
Count an item
Append an item at a time
Remove an item
Remove an item
Reverse the list
Append an item
Remove an item
Insert an item
Sort the list

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
 'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
 'thisStringIsAwesomeInnit'
>>> 'm' in my_string
 True
```

String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

```
>>> my_string.upper()
>>> my_string.lower()
>>> my_string.count('w')
>>> my_string.replace('e', 'i')
>>> my_string.strip()
```

String to uppercase
String to lowercase
Count String elements
Replace String elements
Strip whitespaces

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

pandas — Data analysis
learn — Machine learning
NumPy — Scientific computing
matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python
spyder — Free IDE that is included with Anaconda
jupyter — Create and share documents with live code, visualizations, text, ...

Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

Subset

```
>>> my_array[1]
  2
```

Select item at index 1

Slice

```
>>> my_array[0:2]
  array([1, 2])
```

Select items at index 0 and 1

Subset 2D Numpy arrays

```
>>> my_2darray[:,0]
  array([1, 4])
```

my_2darray[rows, columns]

Numpy Array Operations

```
>>> my_array > 3
  array([False, False, False,  True], dtype=bool)
>>> my_array * 2
  array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
  array([6, 8, 10, 12])
```

Numpy Array Functions

```
>>> my_array.shape
>>> np.append(other_array)
>>> np.insert(my_array, 1, 5)
>>> np.delete(my_array,[1])
>>> np.mean(my_array)
>>> np.median(my_array)
>>> my_array.corrcoef()
>>> np.std(my_array)
```

Get the dimensions of the array
Append items to an array
Insert items in an array
Delete items in an array
Mean of the array
Median of the array
Correlation coefficient
Standard deviation

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

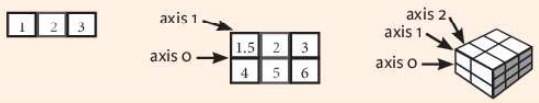
NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))                Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16) Create an array of ones
>>> d = np.arange(10,25,5)         Create an array of evenly
                                   spaced values (step value)
>>> np.linspace(0,2,9)             Create an array of evenly
                                   spaced values (number of samples)
>>> e = np.full((2,2),7)           Create a constant array
>>> f = np.eye(2)                  Create a 2X2 identity matrix
>>> np.random.random((2,2))        Create an array with random values
>>> np.empty((3,2))                Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64      Signed 64-bit integer types
>>> np.float32    Standard double-precision floating point
>>> np.complex    Complex numbers represented by 128 floats
>>> np.bool       Boolean type storing TRUE and FALSE values
>>> np.object     Python object type
>>> np.string_    Fixed-length string type
>>> np.unicode_   Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape          Array dimensions
>>> len(a)           Length of array
>>> b.ndim           Number of array dimensions
>>> e.size           Number of array elements
>>> b.dtype          Data type of array elements
>>> b.dtype.name     Name of data type
>>> b.astype(int)    Convert an array to a different type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                      Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)               Subtraction
>>> b + a                          Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                    Addition
>>> a / b                          Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                 Division
>>> a * b                          Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)               Multiplication
>>> np.exp(b)                      Exponentiation
>>> np.sqrt(b)                     Square root
>>> np.sin(a)                      Print sines of an array
>>> np.cos(b)                      Element-wise cosine
>>> np.log(a)                      Element-wise natural logarithm
>>> e.dot(f)                       Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                         Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                          Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)           Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()              Array-wise sum
>>> a.min()              Array-wise minimum value
>>> b.max(axis=0)        Maximum value of an array row
>>> b.cumsum(axis=1)     Cumulative sum of the elements
>>> a.mean()             Mean
>>> b.median()           Median
>>> a.corrcoef()         Correlation coefficient
>>> np.std(b)            Standard deviation
```

Copying Arrays

```
>>> h = a.view()         Create a view of the array with the same data
>>> np.copy(a)           Create a copy of the array
>>> h = a.copy()         Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()             Sort an array
>>> c.sort(axis=0)       Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]                  Select the element at the 2nd index
  3
>>> b[1,2]                Select the element at row 1 column 2
  6.0                     (equivalent to b[1][2])
```

Slicing

```
>>> a[0:2]                Select items at index 0 and 1
  array([1, 2])
>>> b[0:2,1]              Select items at rows 0 and 1 in column 1
  array([ 2.,  5.])
>>> b[:1]                 Select all items at row 0
  array([[1.5, 2., 3.]])  (equivalent to b[0:1, :])
>>> c[1,...]              Same as [1,:,:]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
>>> a[ : :-1]             Reversed array a
  array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2]                Select elements from a less than 2
  array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]  Select elements (1,0),(0,1),(1,2) and (0,0)
  array([ 4. ,  2. ,  6. ,  1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]  Select a subset of the matrix's rows
  array([[ 4. ,  5. ,  6. ,  4. ],  and columns
         [ 1.5,  2. ,  3. ,  1.5],
         [ 4. ,  5. ,  6. ,  4. ],
         [ 1.5,  2. ,  3. ,  1.5]])
```

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)   Permute array dimensions
>>> i.T                   Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()             Flatten the array
>>> g.reshape(3,-2)       Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))       Return a new array with shape (2,6)
>>> np.append(h,g)         Append items to an array
>>> np.insert(a, 1, 5)     Insert items in an array
>>> np.delete(a,[1])       Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)  Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))              Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                    Stack arrays vertically (row-wise)
>>> np.hstack((e,f))              Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))        Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                    Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)        Split the array horizontally at the 3rd
  [array([1]),array([2]),array([3])]  index
>>> np.vsplit(c,2)        Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

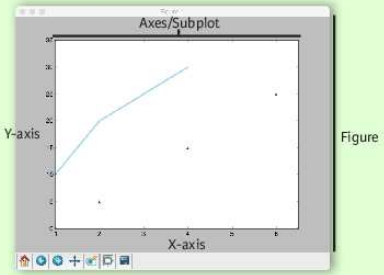
Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                  Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()             Step 2
>>> ax = fig.add_subplot(111)      Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)  Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                     Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)
>>> ax.axis('equal')
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])
>>> ax.set_xlim(0,10.5)
```

Add padding to a plot
Set the aspect ratio of the plot to 1
Set limits for x-and y-axis
Set limits for x-axis

Legends

```
>>> ax.set(title='An Example Axes',
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')
```

Set a title and x-and y-axis labels

No overlapping plot elements

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',
                   direction='inout',
                   length=10)
```

Manually set x-ticks

Make y-ticks longer and go in and out

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()
```

Adjust the spacing between subplots

Fit subplot(s) in to the figure area

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)
>>> ax1.spines['bottom'].set_position(('outward',10))
```

Make the top axis line for a plot invisible
Move the bottom axis line outward

3 Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)
>>> ax.scatter(x,y)
>>> axes[0,0].bar([1,2,3],[3,4,5])
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])
>>> axes[1,1].axhline(0.45)
>>> axes[0,1].axvline(0.65)
>>> ax.fill(x,y,color='blue')
>>> ax.fill_between(x,y,color='yellow')
```

Draw points with lines or markers connecting them
Draw unconnected points, scaled or colored
Plot vertical rectangles (constant width)
Plot horiontal rectangles (constant height)
Draw a horizontal line across axes
Draw a vertical line across axes
Draw filled polygons
Fill between y-values and 0

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)
>>> axes[1,1].quiver(y,z)
>>> axes[0,1].streamplot(X,Y,U,V)
```

Add an arrow to the axes
Plot a 2D field of arrows
Plot a 2D field of arrows

Data Distributions

```
>>> ax1.hist(y)
>>> ax3.boxplot(y)
>>> ax3.violinplot(z)
```

Plot a histogram
Make a box and whisker plot
Make a violin plot

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

Colormapped or RGB arrays

```
>>> axes2[0].pcolor(data2)
>>> axes2[0].pcolormesh(data)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2]= ax.clabel(CS)
```

Pseudocolor plot of 2D array
Pseudocolor plot of 2D array
Plot contours
Plot filled contours
Label a contour plot

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()
>>> plt.clf()
>>> plt.close()
```

Clear an axis
Clear the entire figure
Close a window

Python For Data Science *Cheat Sheet*

SciPy - Linear Algebra

Learn More Python for Data Science Interactively at www.datacamp.com

SciPy

The **SciPy** library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

Index Tricks

`>>> np.mgrid[0:5,0:5]`	Create a dense meshgrid
`>>> np.ogrid[0:2,0:2]`	Create an open meshgrid
`>>> np.r_[[3,[0]*5,-1:1:10j]`	Stack arrays vertically (row-wise)
`>>> np.c_[b,c]`	Create stacked column-wise arrays

Shape Manipulation

`>>> np.transpose(b)`	Permute array dimensions
`>>> b.flatten()`	Flatten the array
`>>> np.hstack((b,c))`	Stack arrays horizontally (column-wise)
`>>> np.vstack((a,b))`	Stack arrays vertically (row-wise)
`>>> np.hsplit(c,2)`	Split the array horizontally at the 2nd index
`>>> np.vpslit(d,2)`	Split the array vertically at the 2nd index

Polynomials

`>>> from numpy import polyld`	
`>>> p = polyld([3,4,5])`	Create a polynomial object

Vectorizing Functions

`>>> def myfunc(a):`
`  if a < 0:`
`  return a*2`
`  else:`
`  return a/2`	

`>>> np.vectorize(myfunc)`	Vectorize functions

Type Handling

`>>> np.real(c)`	Return the real part of the array elements
`>>> np.imag(c)`	Return the imaginary part of the array elements
`>>> np.real_if_close(c,tol=1000)`	Return a real array if complex parts close to 0
`>>> np.cast['f'](np.pi)`	Cast object to a data type

Other Useful Functions

`>>> np.angle(b,deg=True)`	Return the angle of the complex argument
`>>> g = np.linspace(0,np.pi,num=5)`	Create an array of evenly spaced values (number of samples)
`>>> g [3:] += np.pi`	
`>>> np.unwrap(g)`	Unwrap
`>>> np.logspace(0,10,3)`	Create an array of evenly spaced values (log scale)
`>>> np.select([c<4],[c*2])`	Return values from a list of arrays depending on conditions
`>>> misc.factorial(a)`	Factorial
`>>> misc.comb(10,3,exact=True)`	Combine N things taken at k time
`>>> misc.central_diff_weights(3)`	Weights for Np-point central derivative
`>>> misc.derivative(myfunc,1.0)`	Find the n-th derivative of a function at a point

Linear Algebra

Also see NumPy

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

Basic Matrix Routines

Inverse

`>>> A.I`	Inverse
`>>> linalg.inv(A)`	Inverse
`>>> A.T`	Tranpose matrix
`>>> A.H`	Conjugate transposition
`>>> np.trace(A)`	Trace

Norm

`>>> linalg.norm(A)`	Frobenius norm
`>>> linalg.norm(A,1)`	L1 norm (max column sum)
`>>> linalg.norm(A,np.inf)`	L inf norm (max row sum)

Rank

`>>> np.linalg.matrix_rank(C)`	Matrix rank

Determinant

`>>> linalg.det(A)`	Determinant

Solving linear problems

`>>> linalg.solve(A,b)`	Solver for dense matrices
`>>> E = np.mat(a).T`	Solver for dense matrices
`>>> linalg.lstsq(D,E)`	Least-squares solution to linear matrix equation

Generalized inverse

`>>> linalg.pinv(C)`	Compute the pseudo-inverse of a matrix (least-squares solver)
`>>> linalg.pinv2(C)`	Compute the pseudo-inverse of a matrix (SVD)

Creating Sparse Matrices

`>>> F = np.eye(3, k=1)`	Create a 2 X2 identity matrix
`>>> G = np.mat(np.identity(2))`	Create a 2x2 identity matrix
`>>> C[C > 0.5] = 0`	
`>>> H = sparse.csr_matrix(C)`	Compressed Sparse Row matrix
`>>> I = sparse.csc_matrix(D)`	Compressed Sparse Column matrix
`>>> J = sparse.dok_matrix(A)`	Dictionary Of Keys matrix
`>>> E.todense()`	Sparse matrix to full matrix
`>>> sparse.issparse(C)`	Identify sparse matrix

Sparse Matrix Routines

Inverse

`>>> sparse.linalg.inv(I)`	Inverse

Norm

`>>> sparse.linalg.norm(I)`	Norm

Solving linear problems

`>>> sparse.linalg.spsolve(H,I)`	Solver for sparse matrices

Sparse Matrix Functions

`>>> sparse.linalg.expm(I)`	Sparse matrix exponential

Matrix Functions

Addition

`>>> np.add(A,D)`	Addition

Subtraction

`>>> np.subtract(A,D)`	Subtraction

Division

`>>> np.divide(A,D)`	Division

Multiplication

`>>> np.multiply(D,A)`	Multiplication
`>>> np.dot(A,D)`	Dot product
`>>> np.vdot(A,D)`	Vector dot product
`>>> np.inner(A,D)`	Inner product
`>>> np.outer(A,D)`	Outer product
`>>> np.tensordot(A,D)`	Tensor dot product
`>>> np.kron(A,D)`	Kronecker product

Exponential Functions

`>>> linalg.expm(A)`	Matrix exponential
`>>> linalg.expm2(A)`	Matrix exponential (Taylor Series)
`>>> linalg.expm3(D)`	Matrix exponential (eigenvalue decomposition)

Logarithm Function

`>>> linalg.logm(A)`	Matrix logarithm

Trigonometric Functions

`>>> linalg.sinm(D)`	Matrix sine
`>>> linalg.cosm(D)`	Matrix cosine
`>>> linalg.tanm(A)`	Matrix tangent

Hyperbolic Trigonometric Functions

`>>> linalg.sinhm(D)`	Hyperbolic matrix sine
`>>> linalg.coshm(D)`	Hyperbolic matrix cosine
`>>> linalg.tanhm(A)`	Hyperbolic matrix tangent

Matrix Sign Function

`>>> np.sigm(A)`	Matrix sign function

Matrix Square Root

`>>> linalg.sqrtm(A)`	Matrix square root

Arbitrary Functions

`>>> linalg.funm(A, lambda x: x*x)`	Evaluate matrix function

Decompositions

Eigenvalues and Eigenvectors

`>>> la, v = linalg.eig(A)`	Solve ordinary or generalized eigenvalue problem for square matrix
`>>> l1, l2 = la`	Unpack eigenvalues
`>>> v[:,0]`	First eigenvector
`>>> v[:,1]`	Second eigenvector
`>>> linalg.eigvals(A)`	Unpack eigenvalues

Singular Value Decomposition

`>>> U,s,Vh = linalg.svd(B)`	Singular Value Decomposition (SVD)
`>>> M,N = B.shape`	
`>>> Sig = linalg.diagsvd(s,M,N)`	Construct sigma matrix in SVD

LU Decomposition

`>>> P,L,U = linalg.lu(C)`	LU Decomposition

Sparse Matrix Decompositions

`>>> la, v = sparse.linalg.eigs(F,1)`	Eigenvalues and eigenvectors
`>>> sparse.linalg.svds(H, 2)`	SVD

Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

Google Custom Search

Scipy Lecture Notes

One document to learn numerics, science, and data with Python

Tutorials on the scientific Python ecosystem: a quick introduction to central tools and techniques. The different chapters each correspond to a 1 to 2 hours course with increasing level of expertise, from beginner to expert.

Download

- PDF, 2 pages per side
- PDF, 1 page per side
- HTML and example files
- Source code (github)

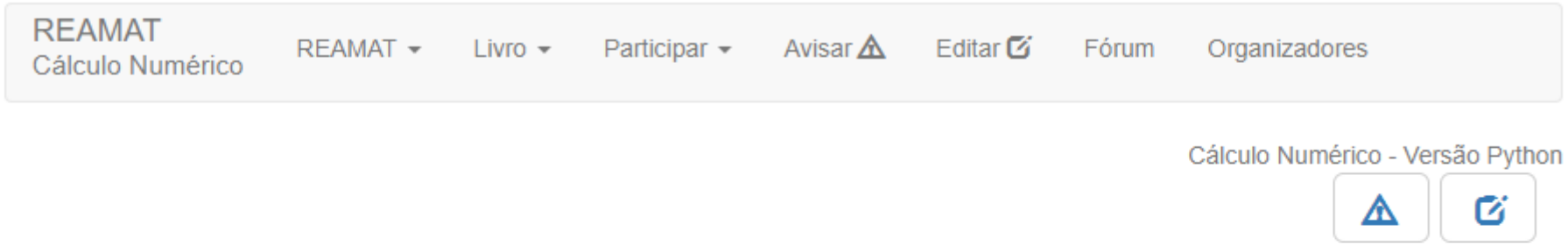
About the scipy lecture notes

Authors What's new License Contributing

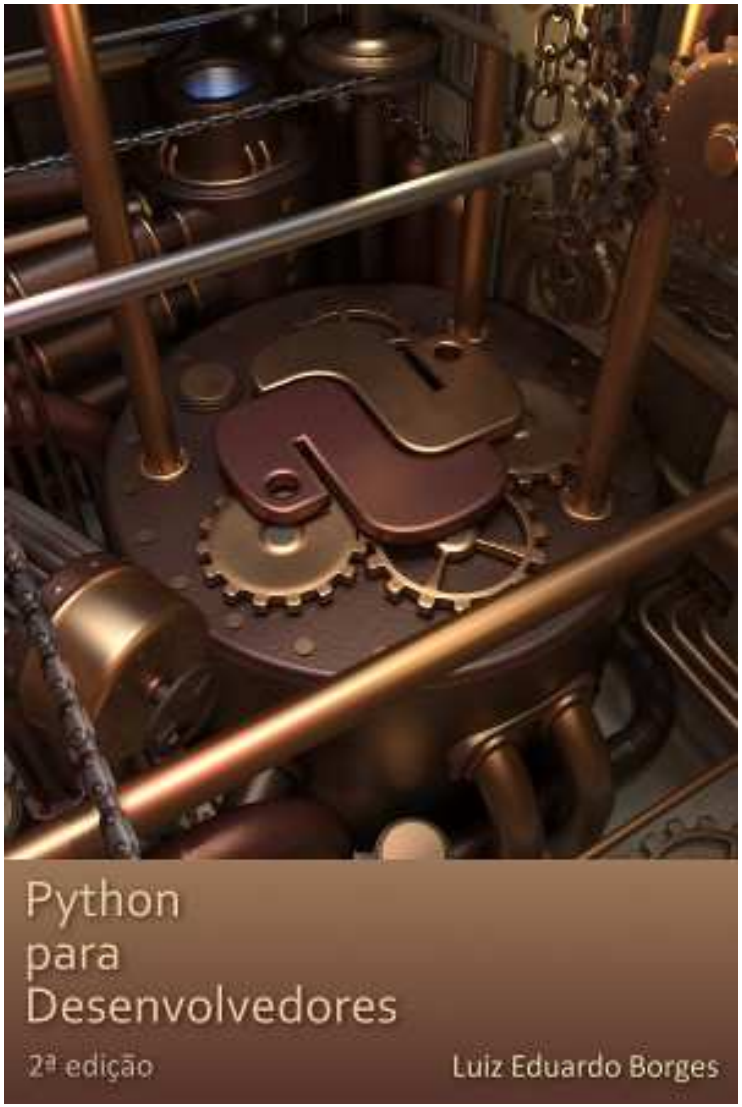
1. Getting started with Python for science

- ▶ 1.1. Python scientific computing ecosystem
- ▶ 1.2. The Python language
- ▶ 1.3. NumPy: creating and manipulating numerical data
- ▶ 1.4. Matplotlib: plotting

https://scipy-lectures.org/



https://www.ufrgs.br/reamat/CalculoNumerico/livro-py/main.html



https://ark4n.wordpress.com/python/

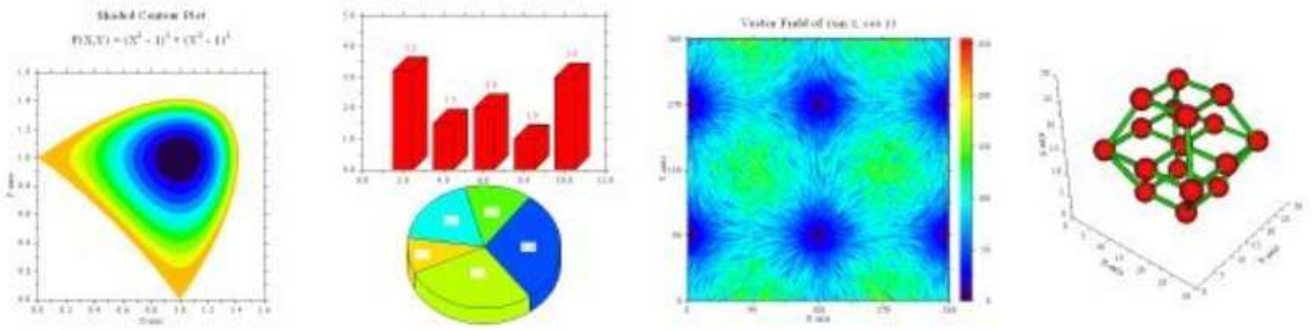
MAX PLANCK INSTITUTE
FOR SOLAR SYSTEM RESEARCH

MPS

DEUTSCH Search

ABOUT MPS | RESEARCH | NEWS AND EVENTS | CAREER | IMPRS | SERVICES

Home > Services > Graphics Software > DISLIN



DISLIN Home Page

Welcome to the home page of the scientific data plotting software DISLIN.

Helmut Michels

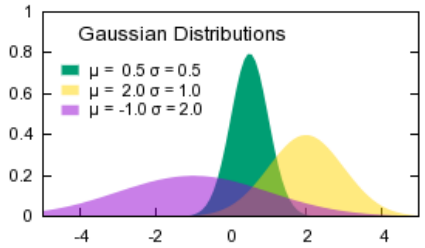
- Home
- Overview
- News
- Examples
- Gallery
- Downloads
- About

https://www.mps.mpg.de/dislin

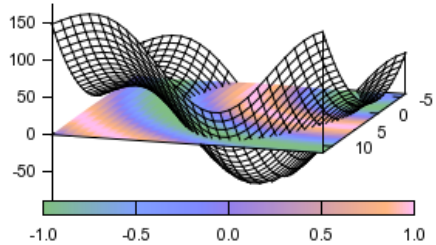
Downloading DISLIN Distributions for Windows 64-bit

- README.WIN
- MD5 sums

DISLIN Distribution	Compiler	Size	Date
dl_11_af.zip	Absoft Pro Fortran 10.x, 11.x, ..., 18.x 64-bit	11235 KB	15-Mar-2019
dl_11_fb.zip	FreeBASIC 64-bit	9534 KB	15-Mar-2019
dl_11_gc.zip	gcc, g++, gfortran Cygwin 64-bit	11281 KB	15-Mar-2019
dl_11_ic.zip	Intel compilers icl, ifort 64-bit	13280 KB	15-Mar-2019
dl_11_jl.zip	Julia 64-bit	8578 KB	15-Mar-2019
dl_11_jv.zip	Java 64-bit	10554 KB	15-Mar-2019
dl_11_mg.zip	gcc, g++, gfortran Mingw64	11281 KB	15-Mar-2019
dl_11_nf.zip	NAG Fortran 6.2 64-bit	11235 KB	15-Mar-2019
dl_11_pf.zip	Portland Visual Fortran 64-bit	11235 KB	15-Mar-2019
dl_11_pl.zip	Perl 5.16.x, 5.18.x, 5.20.x, 5.22.x 64-bit	10301 KB	15-Mar-2019
dl_11_py.zip	Python 2.7, 3.2, 3.3, 3.4, 3.6, 3.7 64-bit	10328 KB	15-Mar-2019
dl_11_r.zip	R 3.2.1 64-bit	10328 KB	15-Mar-2019
dl_11_sf.zip	Silverfrost Fortran 8.30 64-bit	11235 KB	15-Mar-2019
dl_11_vc.zip	MS Visual C++ 64-bit	12271 KB	15-Mar-2019



gnuplot homepage



Gnuplot is a portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms. The source code is copyrighted but freely distributed (i.e., you don't have to pay for it). It was originally created to allow scientists and students to visualize mathematical functions and data interactively, but has grown to support many non-interactive uses such as web scripting. It is also used as a plotting engine by third-party applications like Octave. Gnuplot has been supported and under active development since 1986.

Gnuplot supports many different types of 2D and 3D plots

Here is a Gallery of demos.

Gnuplot supports many different types of output

interactive screen display:	cross-platform (Qt, wxWidgets, x11) or system-specific (MS Windows, OS/2)
direct output to file:	postscript (including eps), pdf, png, gif, jpeg, LaTeX, metafont, emf, svg, ...
mouseable web display formats:	HTML5, svg

http://www.gnuplot.info/

GSL - GNU Scientific Library

Introduction

The GNU Scientific Library (GSL) is a numerical library for C and C++ programmers. It is free software under the GNU General Public License.

The library provides a wide range of mathematical routines such as random number generators, special functions and least-squares fitting. There are over 1000 functions in total with an extensive test suite.

The complete range of subject areas covered by the library includes,

Complex Numbers	Roots of Polynomials
Special Functions	Vectors and Matrices
Permutations	Sorting
BLAS Support	Linear Algebra
Eigensystems	Fast Fourier Transforms
Quadrature	Random Numbers
Quasi-Random Sequences	Random Distributions
Statistics	Histograms
N-Tuples	Monte Carlo Integration
Simulated Annealing	Differential Equations
Interpolation	Numerical Differentiation
Chebyshev Approximation	Series Acceleration
Discrete Hankel Transforms	Root-Finding
Minimization	Least-Squares Fitting
Physical Constants	IEEE Floating-Point
Discrete Wavelet Transforms	Basis splines
Running Statistics	Sparse Matrices and Linear Algebra

https://www.gnu.org/software/gsl/

NUMERICAL RECIPES™
The Art of Scientific Computing
Third Edition

Important note: We have changed our web address. We are no longer "nr.com". We are now "numerical.recipes" (without any .com). Please change your bookmarks. We are the same enterprise and look forward to continuing to serve your Numerical Recipes needs into the future.

E-book readers: Our new, easy-to-remember e-book URLs are numerical.recipes/book for individual subscribers, and numerical.recipes/corporate for corporate and institutional users.

Click on any image below to display in the right column more information about the product or service.

The book.

The on-line electronic book.

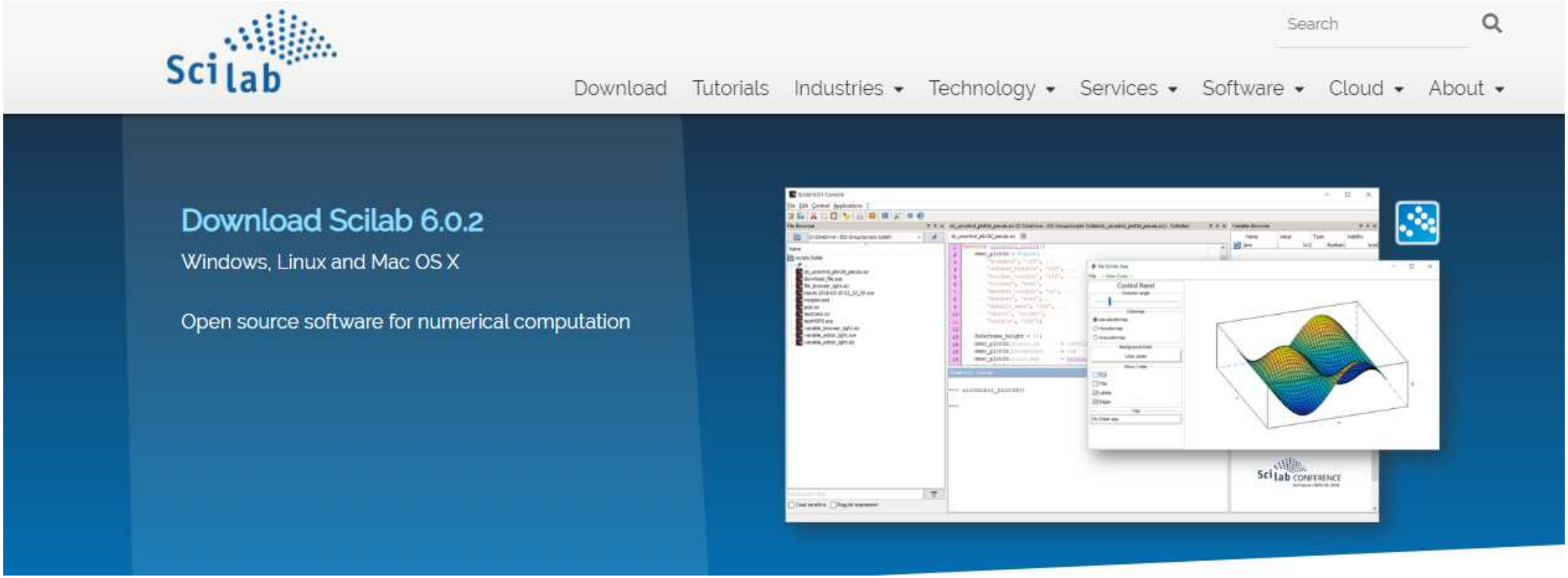
Numerical Recipes Home Page

We are **numerical.recipes**, Numerical Recipes Software. We are one of the oldest continuously operating sites on the Web, with the historic former domain **nr.com** dating back to 1993, one of the first 25,000 domains in the Internet. (Today, that number is about 200,000,000.) In partnership with Cambridge University Press, we develop the *Numerical Recipes* series of books on scientific computing and related software products.

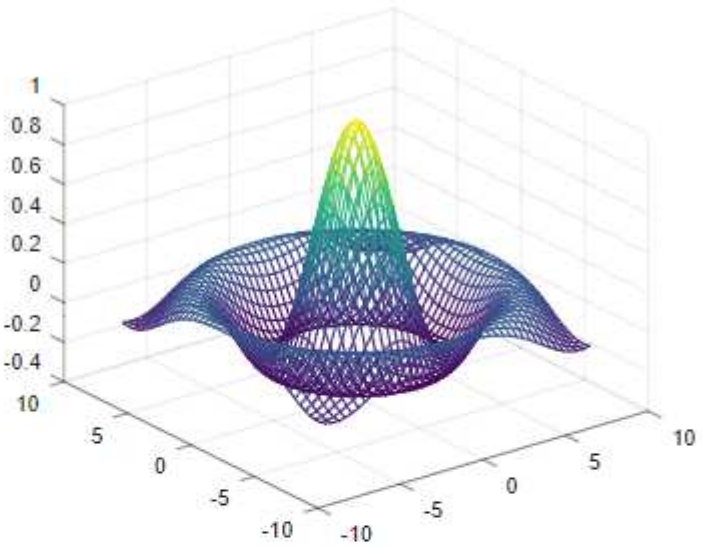

Latest News!

Like 2.3K We're now on Facebook as Numerical Recipes Software. Give us a "like"

http://numerical.recipes/



https://www.scilab.org/



GNU Octave

Scientific Programming Language

- Powerful mathematics-oriented syntax with built-in plotting and visualization tools
- Free software, runs on GNU/Linux, macOS, BSD, and Windows
- Drop-in compatible with many Matlab scripts

Download | Docs

Syntax Examples

The Octave syntax is largely compatible with Matlab. The Octave interpreter can be run in GUI mode, as a console, or invoked as part of a shell script. More Octave examples can be found in the wiki.

Solve systems of equations with linear algebra operations on **vectors** and **matrices**.

```
b = [4; 9; 2] # Column vector
A = [ 3 4 5;
      1 3 1;
      3 5 9 ]
x = A \ b     # Solve the system Ax = b
```

https://www.gnu.org/software/octave/

http://freemat.sourceforge.net/

[Home]

Download

CRAN

R Project

About R
Logo
Contributors
What's New?
Reporting Bugs
Conferences
Search
Get Involved: Mailing Lists
Developer Pages
R Blog

R Foundation

Foundation

The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred CRAN mirror.

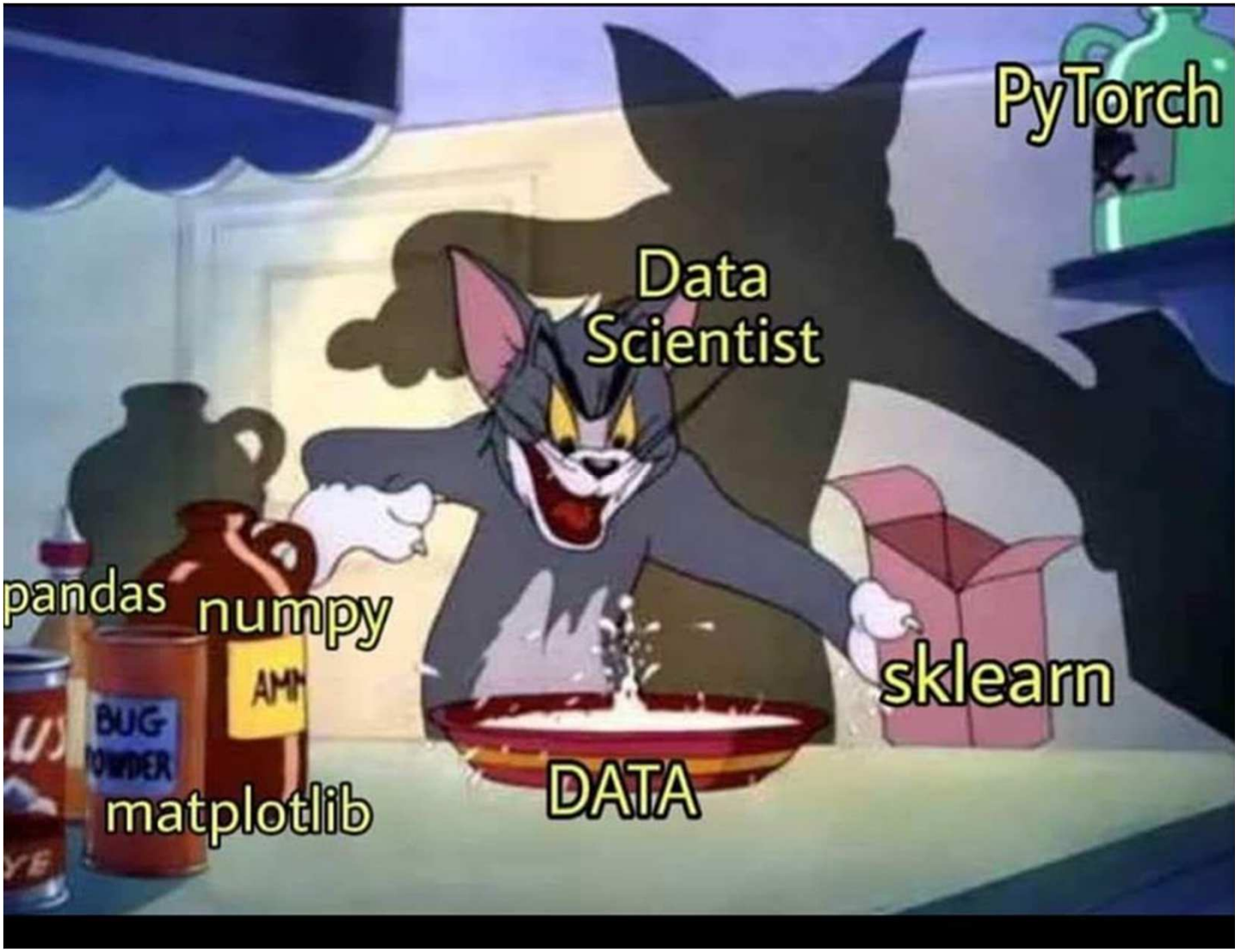
If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

News

- **R version 3.6.1 (Action of the Toes)** has been released on 2019-07-05.
- useR! 2020 will take place in St. Louis, Missouri, USA.
- **R version 3.5.3 (Great Truth)** has been released on 2019-03-11.
- The R Foundation Conference Committee has released a call for proposals to host useR! 2020 in North America.
- You can now support the R Foundation with a renewable subscription as a supporting member
- The R Foundation has been awarded the Personality/Organization of the year 2018 award by the professional association of German market and social researchers.

https://www.r-project.org/

Obrigado!!!!



lazaro.camargo@inpe.br