Ecosistema *Python* de Computação Científica

Python, Numpy, Scipy através de exemplos

baseado livremente em `https://scipy-lectures.org`

Fabricio Ferrari
www.ferrari.pro.br

2019

IMEF – FURG

Disclaimer

Este conjunto de slides tem fim e objetivos educativos. É vedado o seu uso sem autorização. É vedado o seu uso comercial. São utilizadas idéias, equações, figuras e imagens de várias fontes; nem todos os direitos autorais são reconhecidos porque nem sempre estão explícitos. Se alguém for lesado, comunique e modificarei de acordo.

This set of slides have educational objectives and goals. Unauthorized use is forbidden. Commercial use is forbidden. Ideas, equations, figures and images from various sources are used; not all copyrights are acknowledged because they are not always explicit. If anyone is impaired, make contact and I'll modify it accordingly.

Conteúdo

1. Python[1]
2. NumPy
3. IPython e Jupyter
4. MatplotLib
5. SciPy

[1] introdução de 5 minutos

Computação Científica

- aquisição de dados (simulação, conrole de experimentos)
- manipular, processar e analisar dados
- visualizar resultados (figuras para análise e para publicação)

Compração Entre Plataformas

Linguagem	**C/C++/Fortran**	**Matlab**	**Scilab, Octave, R, IDL**	**Python**
Prós	muito rápido	biblioteca extensa, rápido, ambiente de desenvolvimento	código livre	rica biblioteca científica, orientada objeto, linguagem multipropósito, código livre, bons ambientes de desenvolvimento
Contras	ciclo de desenvolvimento penoso, sintaxe difícil, sem ambiente interativo, gerenciamento de memória manual	caro, linguagem restrita	linguagens menos desenvolvidas	lento?

Ecossistema de Computação Científica

Python

+

Numpy

+

Scilab

+

Matplotlib

Ambiente: **iPython + Jupyter**

Pacote Específicos:

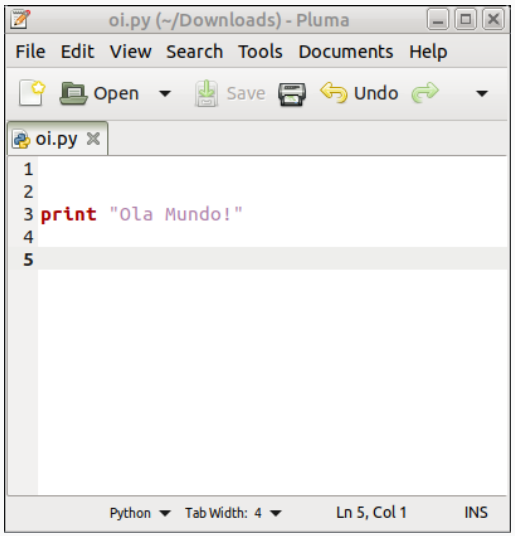
Pandas: manipulação de dados
StatsModels, Seaborn: estatística
Scikit-Image: processamento de imagens
Scikit-Learn: aprendizado de máquina

Python[a]

[a] introdução de 5 minutos

Python

```
% python3
Python 3.6.8 (default, Oct  7 2019, 12:59:55)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Olá Mundo!"
```




```
% python3 oi.py
```

Tipos Básicos

Inteiros

```
>>> 1 + 1
2
>>> a = 4
>>> type(a)
<type 'int'>
```

Ponto Flutuante

```
>>> c = 2.1
>>> type(c)
<type 'float'>
```

Complexo

```
>>> a = 1.5 + 0.5j
>>> a.real
1.5
>>> a.imag
0.5
>>> type(1. + 0j)
<type 'complex'>
```

Booleano

```
>>> 3 > 4
False
>>> test = (3 > 4)
>>> test
False
>>> type(test)
<type 'bool'>
```

Containers

Lists

```
#A list is an ordered collection of objects, that may have different types. For example:

>>> l = [3, -200, 'hello']
>>> l
[3, -200, 'hello']
>>> l[1], l[2]
(-200, 'hello')

>>> l = ['red', 'blue', 'green', 'black', 'white']
>>> type(l)
<type 'list'>
>>> l[2]
'green'
>>> l
['red', 'blue', 'green', 'black', 'white']
>>> l[2:4]
['green', 'black']


>>> L = ['red', 'blue', 'green', 'black', 'white']
>>> L.append('pink')
>>> L
['red', 'blue', 'green', 'black', 'white', 'pink']
>>> L.pop() # removes and returns the last item
'pink'
>>> L
['red', 'blue', 'green', 'black', 'white']
>>> L.extend(['pink', 'purple']) # extend L, in-place
>>> L
['red', 'blue', 'green', 'black', 'white', 'pink', 'purple']
>>> L = L[:-2]
>>> L
['red', 'blue', 'green', 'black', 'white']
```

Strings

```
s = 'Hello, how are you?'

s = "Hi, what's up"

s = '''Hello, # tripling the quotes allows the
how are you''' # the string to span more than one line

s = """Hi,
what's up?"""

# strings se comportam como listas
>>> a = "hello"
>>> a[0]
'h'
>>> a[1]
'e'
>>> a[-1]
'o'


# strings se comportam como listas
>>> a = "hello, world!"
>>> a[3:6] # 3rd to 6th (excluded) elements: elements 3, 4, 5
'lo,'
>>> a[2:10:2] # Syntax: a[start:stop:step]
'lo o'
>>> a[::3] # every three characters, from beginning to end
'hl r!'
```

Dicionários

dicionarios mapeiam palavras-chaves a valores

```
>>> tel = {'emmanuelle': 5752, 'sebastian': 5578}
>>> tel['francis'] = 5915
>>> tel
{'sebastian': 5578, 'francis': 5915, 'emmanuelle': 5752}
>>> tel['sebastian']
5578
>>> tel.keys()
['sebastian', 'francis', 'emmanuelle']
>>> tel.values()
[5578, 5915, 5752]
>>> 'francis' in tel
True
```

if/elif/else

```
>>> if 2**2 == 4:
... print('Obvious!')

>>> a = 10
>>> if a == 1:
...     print(1)
... elif a == 2:
...     print(2)
... else:
...     print('A lot')

A lot
```

For

laço com número de repetições pré definido

```
>>> for i in range(4):     # expressão se torna    for i in [0,1,2,3]:
...     print(i)           #                           print i

0
1
2
3


>>> for word in ('cool', 'powerful', 'readable'):
...     print(word)

cool
powerful
readable


>>> fruits = ["apple", "banana", "cherry"]
>>> for x in fruits:
...     print(x)

apple
banana
cherry

>>> for xx in range(2, 20, 3):
...     print(xx)
... else:
...     print("Finally finished!")

2
5
8
11
14
17
Finally finished!
```

Laços

while

repete por um número indeterminado (em princípio) de vezes

```
T = leia_temperatura()
while T<22:
    print "Frio"
    T = leia_temperatura()
```

Funçoes

```
>>> def test():
....     print('testando')


>>> teste()
testando


>>> def disk_area(radius):
...     return 3.14 * radius * radius

>>> disk_area(1.5)
7.0649999999999995
```

NumPy

NumPy Arrays

Arrays NumPy

- extensão do Python para arrays multidimensionais
- implementação mais eficiente
- projetado para computação científica

```
>>> import numpy as np


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

Criando Arrays

Especificando os valores explicitamente

1D

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

2D, 3D

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

>>> c = np.array([[[1], [2]], [[3], [4]]])
>>> c
array([[[1],
        [2]],
       [[3],
        [4]]])
>>> c.shape
(2, 2, 1)
```

Criando Arrays

Por instrução

Espaçamento:

`np.arange(inicio, fim(exclusive), passo)`

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

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

Número de pontos:

`linspace(inicio, fim, numero-de-pontos)`

```
>>> c = np.linspace(0, 1, 6) # start, end, num-points
>>> c
array([ 0. , 0.2, 0.4, 0.6, 0.8, 1. ])
>>> d = np.linspace(0, 1, 5, endpoint=False)
>>> d
array([ 0. , 0.2, 0.4, 0.6, 0.8])
```

Criando Arrays

casos particulares

Arrays frequentes

```
>>> a = np.ones((3, 3)) # reminder: (3, 3) is a tuple
>>> a
array([[ 1., 1., 1.],
       [ 1., 1., 1.],
       [ 1., 1., 1.]])

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

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

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


>>> a = np.random.rand(4) # uniform in [0, 1]
>>> a
array([ 0.95799151, 0.14222247, 0.08777354, 0.51887998])
```

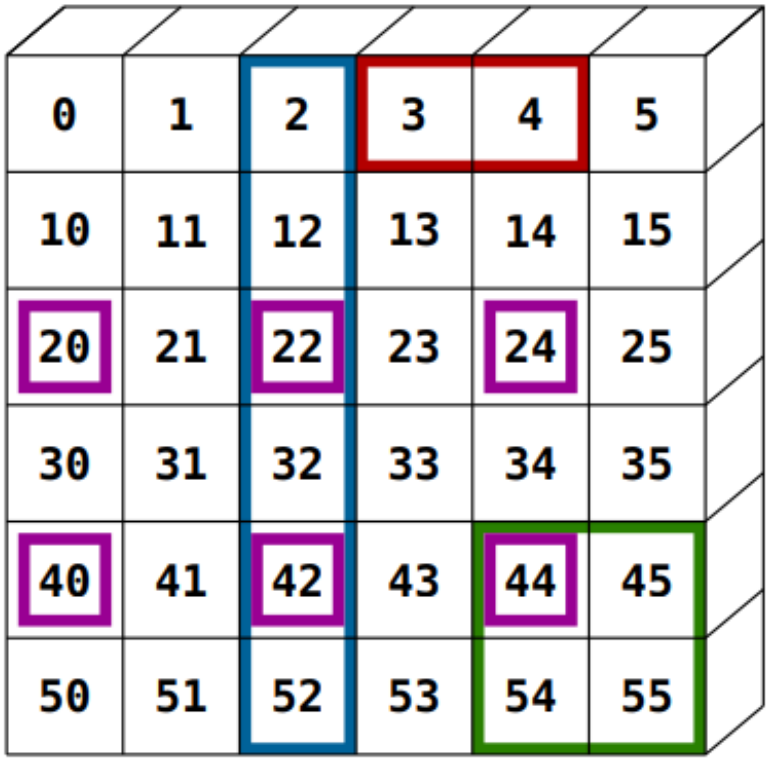
Fatias

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

>>> a[2:9:3] # [start:end:step]
array([2, 5, 8])

>>> a[:4]
array([0, 1, 2, 3])

>>> a[1:3]
array([1, 2])

>>> a[::2]
array([0, 2, 4, 6, 8])

>>> a[3:]
array([3, 4, 5, 6, 7, 8, 9])
```

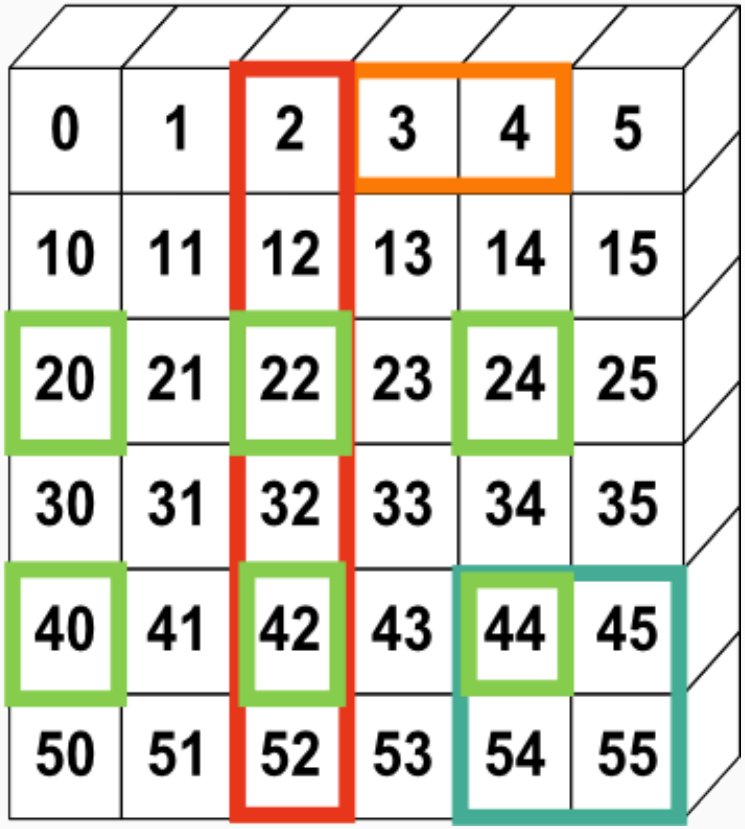
Arrays

Fatiando

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

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

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

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



```
>>> 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]])
```

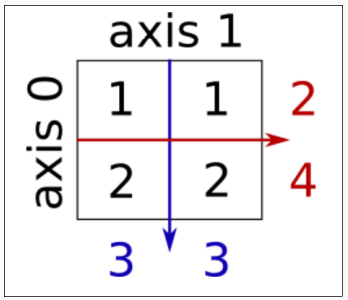


elemento a elemento

```
>>> a = np.array([1, 2, 3, 4])
>>> a + 1
array([2, 3, 4, 5])
>>> 2**a
array([ 2, 4, 8, 16])

>>> a = np.arange(5)
>>> np.sin(a) # elemento a elemento
array([ 0. , 0.84147098, 0.90929743, 0.14112001, -0.7568025 ])
>>> np.log(a) # elemento a elemento
array([ -inf, 0. , 0.69314718, 1.09861229, 1.38629436])
>>> np.exp(a) # elemento a elemento
array([ 1. , 2.71828183, 7.3890561 , 20.08553692, 54.59815003])

>>> a = np.array([1, 2, 3, 4])
>>> b = np.array([4, 2, 2, 4])
>>> a+b # elemento a elemento
array([5, 4, 5, 8])
>>> a*b # elemento a elemento
array([ 4,  4,  6, 16])
>>> a/b # elemento a elemento
array([ 0.25,  1.  ,  1.5 ,  1.  ])
>>> a-b # elemento a elemento
array([-3,  0,  1,  0])

>>> a == b
array([False, True, False, True], dtype=bool)
>>> a > b
array([False, False, True, False], dtype=bool)
```

Soma

```
>>> x = np.array([1, 2, 3, 4])
>>> x.sum()
10
```



Somando ao longo de linhas e colunas

```
>>> x = np.array([[1, 1], [2, 2]])
>>> x
array([[1, 1],
[2, 2]])
>>> x.sum(axis=0) # columas
array([3, 3])
>>> x[:, 0].sum(), x[:, 1].sum()
(3, 3)
>>> x.sum(axis=1) # linhas
array([2, 4])
```

Extrema

```
>>> x = np.array([1, 3, 2])
>>> x.min()
1
>>> x.max()
3
>>> x.argmin() # index of minimum
0
>>> x.argmax() # index of maximum
1
```

Estatísticas

```
>>> x = np.array([1, 2, 3, 1])
>>> y = np.array([[1, 2, 3], [5, 6, 1]])
>>> x.mean()
1.75
>>> np.median(x)
1.5
>>> np.median(y, axis=-1) # last axis
array([ 2., 5.])
>>> x.std() # full population standard dev.
0.82915619758884995
```

Achatar

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]])
>>> a.ravel()
array([1, 2, 3, 4, 5, 6])
>>> a.T
array([[1, 4],
[2, 5],
[3, 6]])
>>> a.T.ravel()
array([1, 4, 2, 5, 3, 6])
```

Reformatar

```
>>> a.shape
(2, 3)
>>> b = a.ravel()
>>> b = b.reshape((2, 3))
>>> b
array([[1, 2, 3],
[4, 5, 6]])
```

INTERLÚDIO

IPython e Jupyter

IPython

Improved Python – interpretador Python melhorado

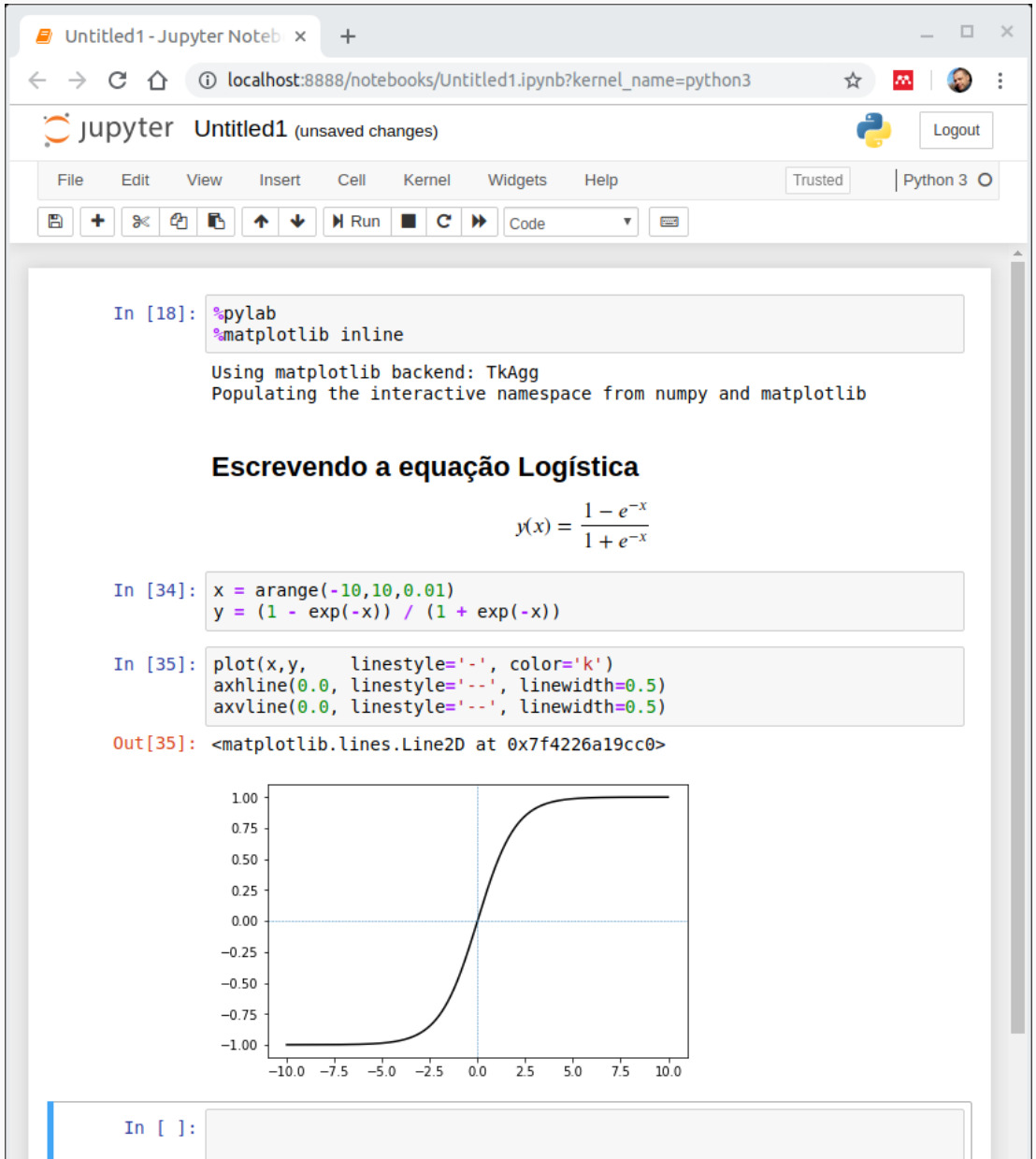
- shell melhorado
- histórico de comandos
- comandos mágicos (log de sessão,)
- depurador

```
% ipython3
Python 3.6.8 (default, Oct  7 2019, 12:59:55)
Type "copyright", "credits" or "license" for more information.

IPython 5.5.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import numpy as np

In [2]: x = np.arange(0,1,0.1)

In [3]: y = np.exp(-x)

In [4]: x
Out[4]: array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])

In [5]: y
Out[5]:
array([ 1.        ,  0.90483742,  0.81873075,  0.74081822,  0.67032005,
0.60653066,  0.54881164,  0.4965853 ,  0.44932896,  0.40656966])
```

`ipython3 --matplotlib`

Jupyter

```
% jupyter notebook
```



MatplotLib

Gráfico Simples

Rotinas Pyplot

```
import matplotlib.pyplot as plt   # carrega biblioteca de graficos
import numpy as np                # carrega numpy

x = np.arange(-10,10,0.01)         # variavel independente [-10,10] passo 0.01
y = (1-np.exp(-x))/(1+np.exp(-x))  # função logistica
plt.plot(x,y)                      # gráfico com opções padrão

plt.show()                         # mostra janela
```

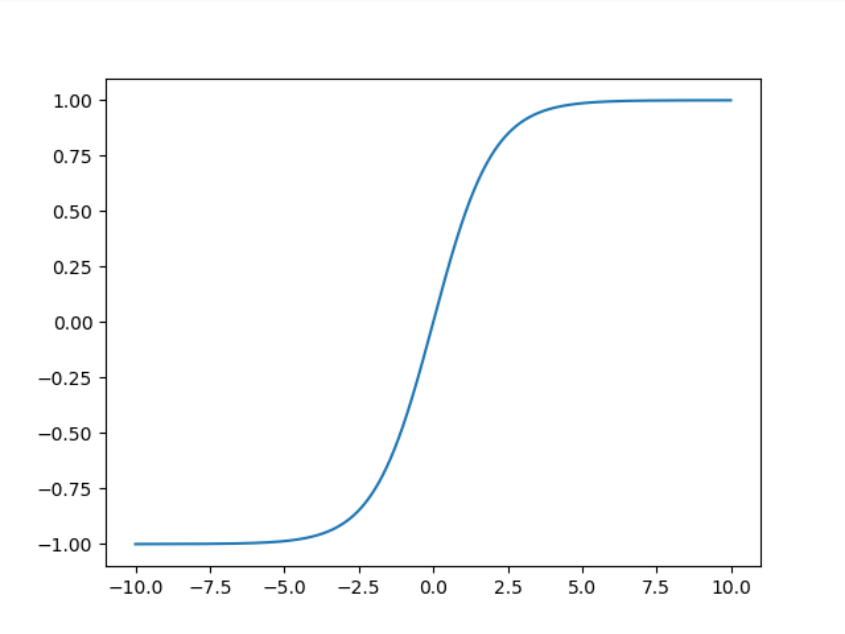
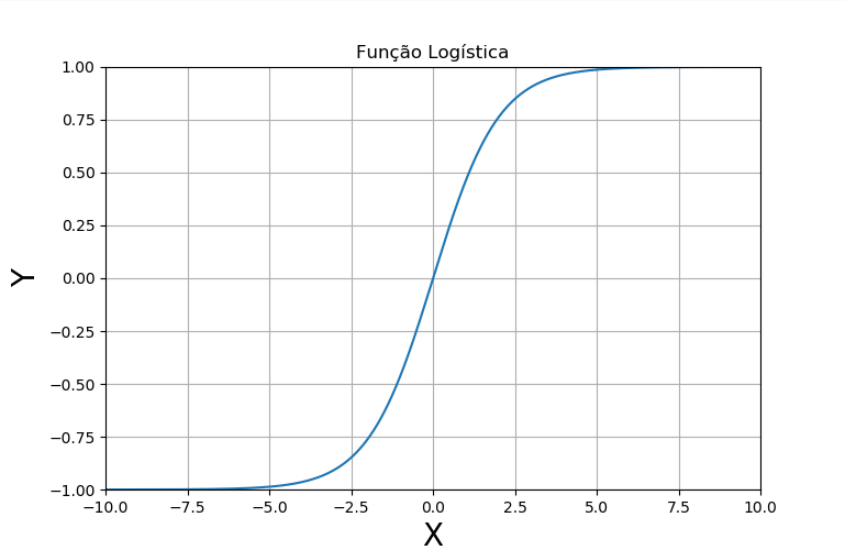


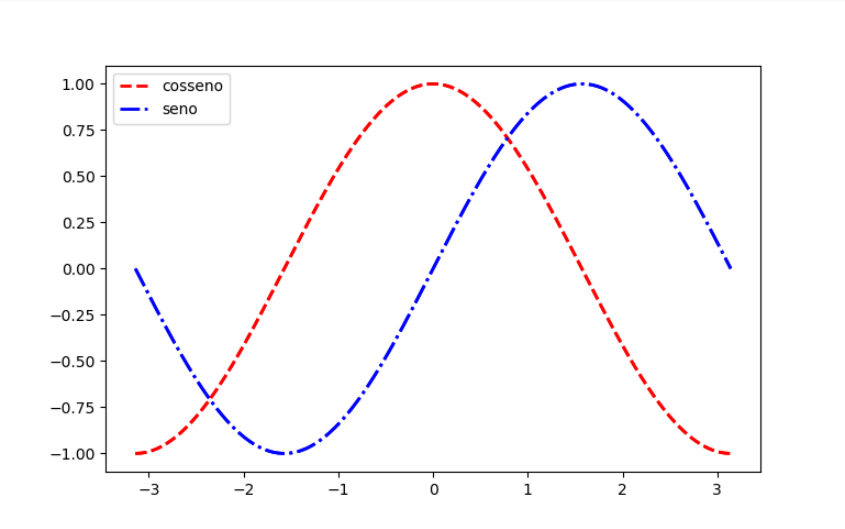
Gráfico Simples

```
plt.xlim(-10,10)
plt.ylim(-1,1)
plt.title("Função Logística")
plt.grid()
plt.xlabel("X", fontsize=20)
plt.ylabel("Y", fontsize=20)
```

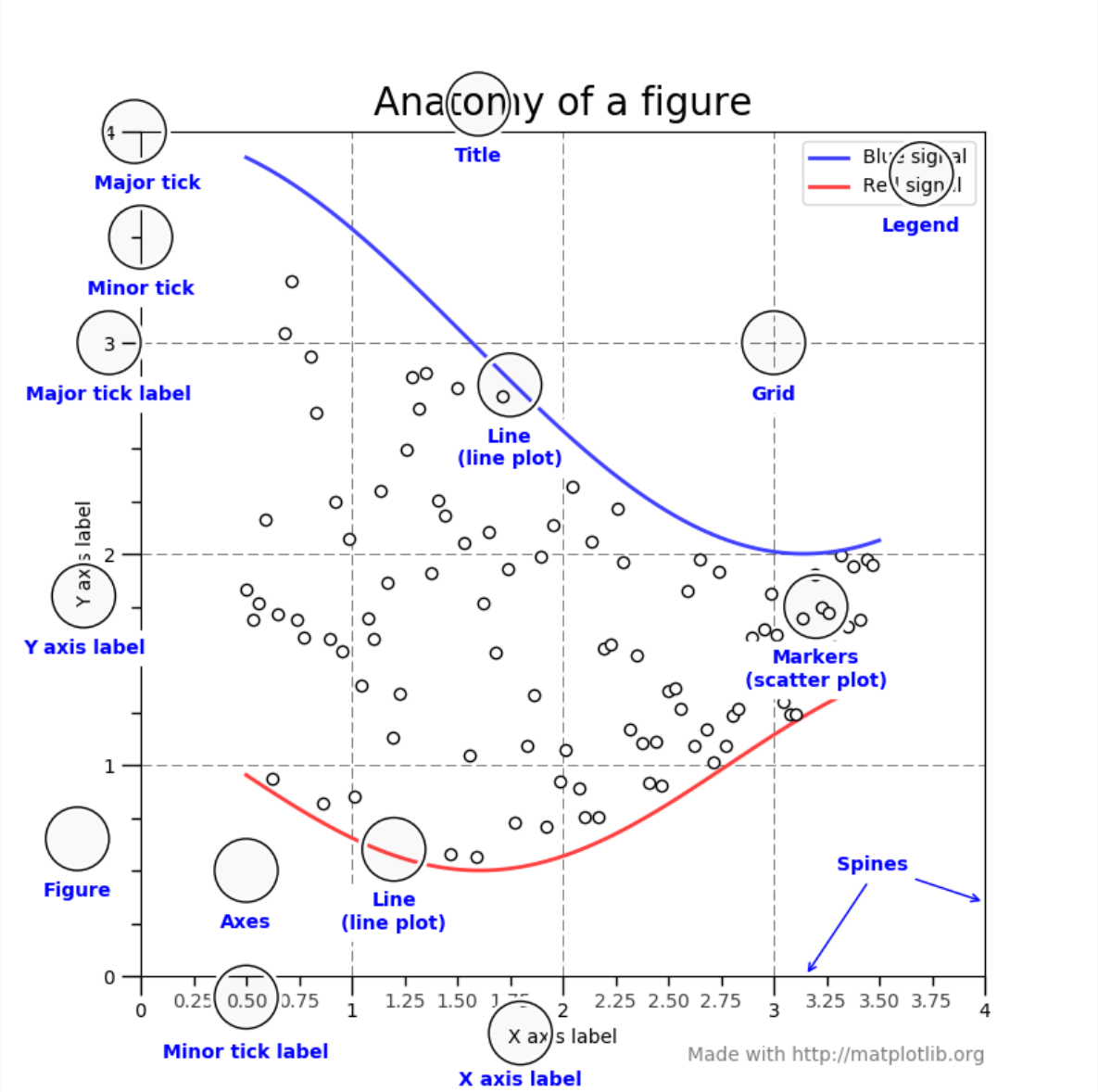


Gráficos Múltiplos

```
t = np.linspace(-np.pi, np.pi, 1000)
x = np.cos(t)
y = np.sin(t)

plt.plot(t,x, color='red', linewidth=2.0, linestyle='--', label='cosseno')
plt.plot(t,y, color='blue', linewidth=2.0, linestyle='-.', label='seno')

plt.legend()

plt.savefig('plot2.png')
```

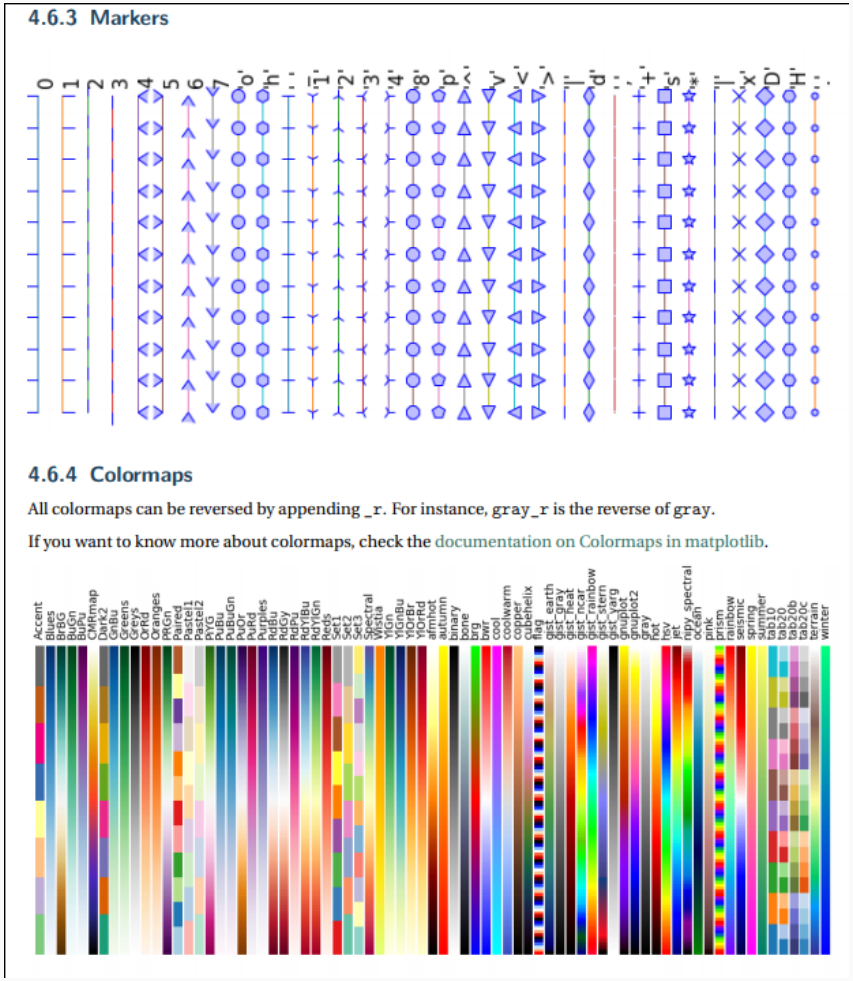


Elementos da Figura



matplotlib.org

Exemplos Matplotlib



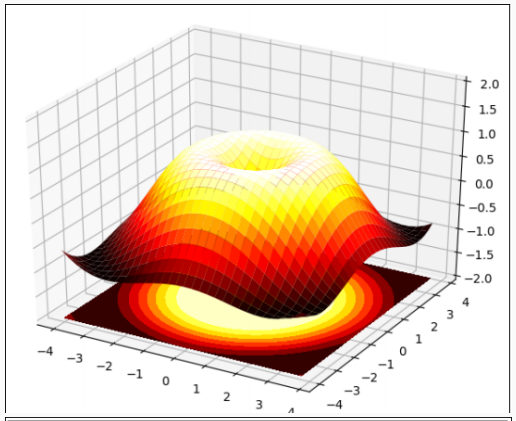
4.6.3 Markers

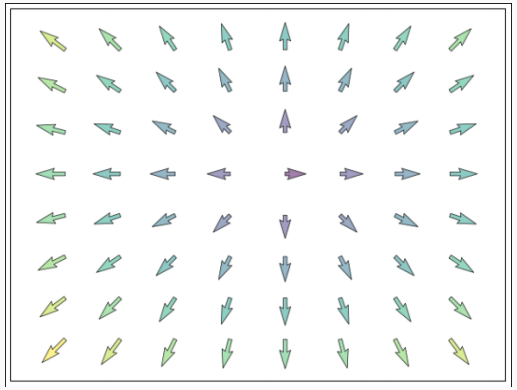
4.6.4 Colormaps

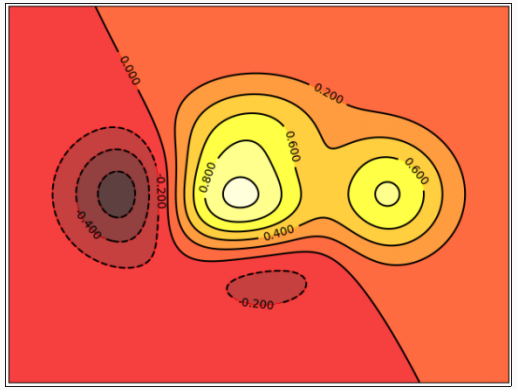
All colormaps can be reversed by appending `_r`. For instance, `gray_r` is the reverse of `gray`.

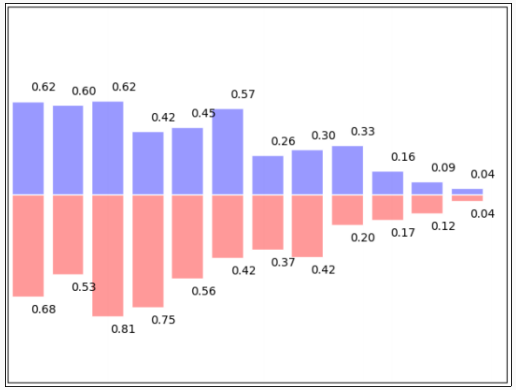
If you want to know more about colormaps, check the documentation on Colormaps in matplotlib.

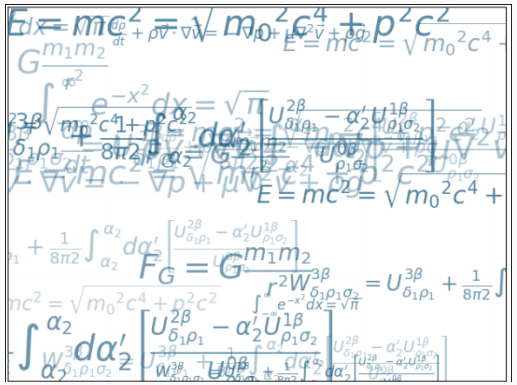
Exemplos

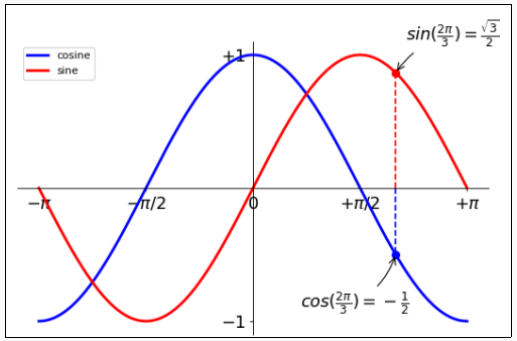





0.000
0.200
0.600
0.800
0.600
-0.200
0.400
-0.400
-0.200


0.62 0.60 0.62 0.42 0.45 0.57 0.26 0.30 0.33 0.16 0.09 0.04
0.68 0.53 0.81 0.75 0.56 0.42 0.37 0.42 0.20 0.17 0.12 0.04




cosine
sine
+1
−1
−π
−π/2
0
+π/2
+π
$\sin(\frac{2\pi}{3}) = \frac{\sqrt{3}}{2}$
$\cos(\frac{2\pi}{3}) = -\frac{1}{2}$

SciPy

Modulos SciPy

- File input/output: `scipy.io`
- Special functions: `scipy.special`
- Linear algebra operations: `scipy.linalg`
- Interpolation: `scipy.interpolate`
- Optimization and fit: `scipy.optimize`
- Statistics and random numbers: `scipy.stats`
- Numerical integration: `scipy.integrate`
- Fast Fourier transforms: `scipy.fftpack`
- Signal processing: `scipy.signal`
- Image manipulation: `scipy.ndimage`

`scipy.io` – Entrada e Saída

Ler/Escrever arquivos

- MatLab
- IDL
- Matrix Market
- Fortran
- NetCDF
- Wav
- Arff

`scipy.special` – Funções Especiais

- Função de Airy
- Integrais e Funções Elípticas
- Funções de Bessel
  1^o e 2^o tipo, argumentos reais e complexos, modificada, Haenkel 1^o e 2^o tipo, zeros, derivadas e integrais, esféricas, Riccati-Bessel
- Funções de Struve
- Funções estatísticas elementares (binomial, Poisson, student, Chi, Gamma, Gaussian)
- Funções da teoria da informação (Entropy, Kullback-Leibler divergence, ...)
- Gamma (função, logaritmo da, inversa, beta completa e incompleta, ...)
- Função erro (direta e complementar, imaginaria, Integrais de Fresnel, ...)
- Funções de Legendre
- Polinômios ortogonais (Laguerre, Legendre, Chebyshev, Jacobi, Hermite)
- Funções Hipergeométricas
- Funções de Mathieu e relacionadas
- Funções de onda esferoidais
- Funções de Kelvin
- Análise Combinatória
- Função W de Lambert e relacionadas

scipy.linalg – Álgebra Linear

Basics

inv(a[, overwrite_a, check_finite])	Compute the inverse of a matrix.
solve(a, b[, sym_pos, lower, overwrite_a, ...])	Solves the linear equation set `a * x = b` for the unknown `x` for square `a` matrix.
solve_banded(l_and_u, ab, b[, overwrite_ab, ...])	Solve the equation a x = b for x, assuming a is banded matrix.
solveh_banded(ab, b[, overwrite_ab, ...])	Solve equation a x = b.
solve_circulant(c, b[, singular, tol, ...])	Solve C x = b for x, where C is a circulant matrix.
solve_triangular(a, b[, trans, lower, ...])	Solve the equation *a x = b* for *x*, assuming a is a triangular matrix.
solve_toeplitz(c_or_cr, b[, check_finite])	Solve a Toeplitz system using Levinson Recursion
det(a[, overwrite_a, check_finite])	Compute the determinant of a matrix
norm(a[, ord, axis, keepdims])	Matrix or vector norm.
lstsq(a, b[, cond, overwrite_a, ...])	Compute least-squares solution to equation Ax = b.
pinv(a[, cond, rcond, return_rank, check_finite])	Compute the (Moore-Penrose) pseudo-inverse of a matrix.
pinv2(a[, cond, rcond, return_rank, ...])	Compute the (Moore-Penrose) pseudo-inverse of a matrix.
pinvh(a[, cond, rcond, lower, return_rank, ...])	Compute the (Moore-Penrose) pseudo-inverse of a Hermitian matrix.
kron(a, b)	Kronecker product.
tril(m[, k])	Make a copy of a matrix with elements above the k-th diagonal zeroed.
triu(m[, k])	Make a copy of a matrix with elements below the k-th diagonal zeroed.
orthogonal_procrustes(A, B[, check_finite])	Compute the matrix solution of the orthogonal Procrustes problem.
matrix_balance(A[, permute, scale, ...])	Compute a diagonal similarity transformation for row/column balancing.
subspace_angles(A, B)	Compute the subspace angles between two matrices.
LinAlgError	Generic Python-exception-derived object raised by linalg functions.
LinAlgWarning	The warning emitted when a linear algebra related operation is close to fail conditions of the algorithm or loss of accuracy is expected.

Eigenvalue Problems

eig(a[, b, left, right, overwrite_a, ...])	Solve an ordinary or generalized eigenvalue problem of a square matrix.
eigvals(a[, b, overwrite_a, check_finite, ...])	Compute eigenvalues from an ordinary or generalized eigenvalue problem.
eigh(a[, b, lower, eigvals_only, ...])	Solve an ordinary or generalized eigenvalue problem for a complex Hermitian or real symmetric matrix.
eigvalsh(a[, b, lower, overwrite_a, ...])	Solve an ordinary or generalized eigenvalue problem for a complex Hermitian or real symmetric matrix.
eig_banded(a_band[, lower, eigvals_only, ...])	Solve real symmetric or complex hermitian band matrix eigenvalue problem.
eigvals_banded(a_band[, lower, ...])	Solve real symmetric or complex hermitian band matrix eigenvalue problem.
eigh_tridiagonal(d, e[, eigvals_only, ...])	Solve eigenvalue problem for a real symmetric tridiagonal matrix.
eigvalsh_tridiagonal(d, e[, select, ...])	Solve eigenvalue problem for a real symmetric tridiagonal matrix.

scipy.linalg – Álgebra Linear

Decompositions

lu(a[, permute_l, overwrite_a, check_finite])	Compute pivoted LU decomposition of a matrix.
lu_factor(a[, overwrite_a, check_finite])	Compute pivoted LU decomposition of a matrix.
lu_solve(lu_and_piv, b[, trans, ...])	Solve an equation system, a x = b, given the LU factorization of a
svd(a[, full_matrices, compute_uv, ...])	Singular Value Decomposition.
svdvals(a[, overwrite_a, check_finite])	Compute singular values of a matrix.
diagsvd(s, M, N)	Construct the sigma matrix in SVD from singular values and size M, N.
orth(A[, rcond])	Construct an orthonormal basis for the range of A using SVD
null_space(A[, rcond])	Construct an orthonormal basis for the null space of A using SVD
ldl(A[, lower, hermitian, overwrite_a, ...])	Computes the LDLt or Bunch-Kaufman factorization of a symmetric/ hermitian matrix.
cholesky(a[, lower, overwrite_a, check_finite])	Compute the Cholesky decomposition of a matrix.
cholesky_banded(ab[, overwrite_ab, lower, ...])	Cholesky decompose a banded Hermitian positive-definite matrix
cho_factor(a[, lower, overwrite_a, check_finite])	Compute the Cholesky decomposition of a matrix, to use in cho_solve
cho_solve(c_and_lower, b[, overwrite_b, ...])	Solve the linear equations A x = b, given the Cholesky factorization of A.
cho_solve_banded(cb_and_lower, b[, ...])	Solve the linear equations `A x = b`, given the Cholesky factorization of the banded hermitian `A`.
polar(a[, side])	Compute the polar decomposition.
qr(a[, overwrite_a, lwork, mode, pivoting, ...])	Compute QR decomposition of a matrix.
qr_multiply(a, c[, mode, pivoting, ...])	Calculate the QR decomposition and multiply Q with a matrix.
qr_update(Q, R, u, v[, overwrite_qruv, ...])	Rank-k QR update
qr_delete(Q, R, k, int p=1[, which, ...])	QR downdate on row or column deletions
qr_insert(Q, R, u, k[, which, rcond, ...])	QR update on row or column insertions
rq(a[, overwrite_a, lwork, mode, check_finite])	Compute RQ decomposition of a matrix.
qz(A, B[, output, lwork, sort, overwrite_a, ...])	QZ decomposition for generalized eigenvalues of a pair of matrices.
ordqz(A, B[, sort, output, overwrite_a, ...])	QZ decomposition for a pair of matrices with reordering.
schur(a[, output, lwork, overwrite_a, sort, ...])	Compute Schur decomposition of a matrix.
rsf2csf(T, Z[, check_finite])	Convert real Schur form to complex Schur form.
hessenberg(a[, calc_q, overwrite_a, ...])	Compute Hessenberg form of a matrix.
cdf2rdf(w, v)	Converts complex eigenvalues `w` and eigenvectors `v` to real eigenvalues in a block diagonal form `wr` and the associated real eigenvectors `vr`, such that.

scipy.linalg – Álgebra Linear

Matrix Functions

expm(A)	Compute the matrix exponential using Pade approximation.
logm(A[, disp])	Compute matrix logarithm.
cosm(A)	Compute the matrix cosine.
sinm(A)	Compute the matrix sine.
tanm(A)	Compute the matrix tangent.
coshm(A)	Compute the hyperbolic matrix cosine.
sinhm(A)	Compute the hyperbolic matrix sine.
tanhm(A)	Compute the hyperbolic matrix tangent.
signm(A[, disp])	Matrix sign function.
sqrtm(A[, disp, blocksize])	Matrix square root.
funm(A, func[, disp])	Evaluate a matrix function specified by a callable.
expm_frechet(A, E[, method, compute_expm, ...])	Frechet derivative of the matrix exponential of A in the direction E.
expm_cond(A[, check_finite])	Relative condition number of the matrix exponential in the Frobenius norr
fractional_matrix_power(A, t)	Compute the fractional power of a matrix.

Matrix Equation Solvers

solve_sylvester(a, b, q)	Computes a solution (X) to the Sylvester equation $AX + XB = Q$.
solve_continuous_are(a, b, q, r[, e, s, ...])	Solves the continuous-time algebraic Riccati equation (CARE).
solve_discrete_are(a, b, q, r[, e, s, balanced])	Solves the discrete-time algebraic Riccati equation (DARE).
solve_continuous_lyapunov(a, q)	Solves the continuous Lyapunov equation $AX + XA^H = Q$.
solve_discrete_lyapunov(a, q[, method])	Solves the discrete Lyapunov equation $AXA^H - X + Q = 0$.

Sketches and Random Projections

clarkson_woodruff_transform(input_matrix, ...)	"

scipy.linalg – Álgebra Linear

Special Matrices

block_diag(*arrs)	Create a block diagonal matrix from provided arrays.
circulant(c)	Construct a circulant matrix.
companion(a)	Create a companion matrix.
dft(n[, scale])	Discrete Fourier transform matrix.
fiedler(a)	Returns a symmetric Fiedler matrix
fiedler_companion(a)	Returns a Fiedler companion matrix
hadamard(n[, dtype])	Construct a Hadamard matrix.
hankel(c[, r])	Construct a Hankel matrix.
helmert(n[, full])	Create a Helmert matrix of order *n*.
hilbert(n)	Create a Hilbert matrix of order *n*.
invhilbert(n[, exact])	Compute the inverse of the Hilbert matrix of order *n*.
leslie(f, s)	Create a Leslie matrix.
pascal(n[, kind, exact])	Returns the n x n Pascal matrix.
invpascal(n[, kind, exact])	Returns the inverse of the n x n Pascal matrix.
toeplitz(c[, r])	Construct a Toeplitz matrix.
tri(N[, M, k, dtype])	Construct (N, M) matrix filled with ones at and below the k-th diagonal.

Low-level routines

get_blas_funcs(names[, arrays, dtype])	Return available BLAS function objects from names.
get_lapack_funcs(names[, arrays, dtype])	Return available LAPACK function objects from names.
find_best_blas_type([arrays, dtype])	Find best-matching BLAS/LAPACK type.

See also:

scipy.linalg.blas – Low-level BLAS functions

scipy.linalg.lapack – Low-level LAPACK functions

scipy.linalg.cython_blas – Low-level BLAS functions for Cython

scipy.linalg.cython_lapack – Low-level LAPACK functions for Cython

scipy.linalg – Exemplo de Inversa

https://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html

```
>>> import numpy as np
>>> from scipy import linalg
>>> A = np.array([[1,3,5],[2,5,1],[2,3,8]])
>>> A
array([[1, 3, 5],
       [2, 5, 1],
       [2, 3, 8]])
>>> linalg.inv(A)  # inversa
array([[-1.48,  0.36,  0.88],
       [ 0.56,  0.08, -0.36],
       [ 0.16, -0.12,  0.04]])
>>> A.dot(linalg.inv(A)) # produto interno da direta e inversa é a identidade
array([[  1.00000000e+00,  -1.11022302e-16,  -5.55111512e-17],
       [  3.05311332e-16,   1.00000000e+00,   1.87350135e-16],
       [  2.22044605e-16,  -1.11022302e-16,   1.00000000e+00]])
```

scipy.linalg – Exemplo de Decomposição Espectral

https://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html

```
>>> import numpy as np
>>> from scipy import linalg
>>> A = np.array([[1, 2], [3, 4]])
>>> la, v = linalg.eig(A)   # decomposicao em autovalores e autovetores
>>> l1, l2 = la
>>> print(l1, l2)   # autovalores
    (-0.3722813232690143+0j) (5.372281323269014+0j)
>>> print(v[:, 0])   # primeiro autovetor
    [-0.82456484  0.56576746]
>>> print(v[:, 1])   # segundo autovetor
    [-0.41597356 -0.90937671]
>>> print(np.sum(abs(v**2), axis=0))  # norma dos autovetores (unitarios)
    [1. 1.]
>>> v1 = np.array(v[:, 0]).T
>>> print(linalg.norm(A.dot(v1) - l1*v1))  # verifica se Ax = lambda x
    3.23682852457e-16
```

`scipy.interpolate` – Interpolação

- Interpolação 1D
- Inerpolação multivariada
- Interpolação por splines
- Interpolação funções de base radial

scipy.interpolate – Exemplo

https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
```

scipy.interpolate – Exemplo

https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
```

```
x = np.linspace(0, 10, num=11, endpoint=True) # variaveis a serem interpoladas
y = np.cos(-x**2/9.0)                         # x e y
f = interp1d(x, y)                            # f tem coef. do  polinomio da interpolação linear
f2 = interp1d(x, y, kind='cubic')             # f2 tem coef. do  polinomio da interpolação cubica
```

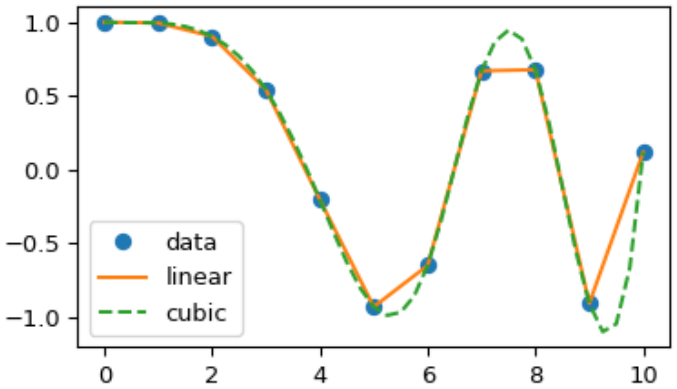
scipy.interpolate – Exemplo

https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
```

```
x = np.linspace(0, 10, num=11, endpoint=True) # variaveis a serem interpoladas
y = np.cos(-x**2/9.0)                         # x e y
f = interp1d(x, y)                            # f tem coef. do  polinomio da interpolação linear
f2 = interp1d(x, y, kind='cubic')             # f2 tem coef. do  polinomio da interpolação cubica
```

```
xnew = np.linspace(0, 10, num=41, endpoint=True)  # novas posicoes onde realizar a interpolação

plt.plot(x, y, 'o')               # pontos originais
plt.plot( xnew, f(xnew), '-')     # pontos interpolados linearmente
plt.plot( xnew, f2(xnew), '--')   # pontos interpolados cubicamente

plt.legend(['data', 'linear', 'cubic'], loc='best')
plt.show()
```

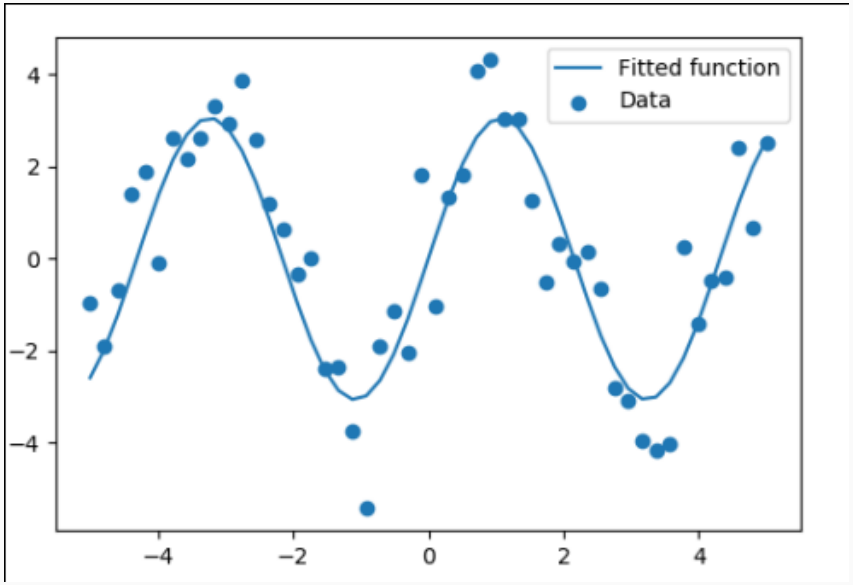


scipy.optimize

- Minimização de funções escalares multivariadas (`minimize`) com vários algoritmos (BFGS, Nelder-Mead simplex, Newton Conjugate Gradient, COBYLA or SLSQP)
- Rotinas de otimização global (basinhopping, differential evolution, shgo, dual annealing)
- Mínimos quadrados e ajuste de curvas (least_squares and curve_fit)
- Minimizadores de funçoes escalares univariadas
- Solução de raízes de equações com vários algoritmos (hybrid Powell, Levenberg-Marquardt or large-scale methods such as Newton-Krylov [KK]).

scipy.optimize – Exemplo

```python
# cria dados artificiais
x_data = np.linspace(-5, 5, num=50)
y_data = 2.9 * np.sin(1.5 * x_data) + np.random.normal(size=50)  # random acrescenta ruído

def test_func(x, a, b):
    "função teste para ajustar"
    return a * np.sin(b * x)

# curve_fit aplica test_func a (x_data, y_data) com parâmetros iniciais [2,2]
params, params_covariance = optimize.curve_fit(test_func, x_data, y_data, p0=[2, 2])
print(params)
# solucao encontrada é
#    [ 3.05931973 1.45754553]
# compare com os parâmetros usado para gerar y_data
```



scipy.stats

Continuous distributions

alpha	An alpha continuous random variable.
anglit	An anglit continuous random variable.
arcsine	An arcsine continuous random variable.
argus	Argus distribution
beta	A beta continuous random variable.
betaprime	A beta prime continuous random variable.
bradford	A Bradford continuous random variable.
burr	A Burr (Type III) continuous random variable.
burr12	A Burr (Type XII) continuous random variable.
cauchy	A Cauchy continuous random variable.
chi	A chi continuous random variable.
chi2	A chi-squared continuous random variable.
cosine	A cosine continuous random variable.
crystalball	Crystalball distribution
dgamma	A double gamma continuous random variable.
dweibull	A double Weibull continuous random variable.
erlang	An Erlang continuous random variable.
expon	An exponential continuous random variable.
exponnorm	An exponentially modified Normal continuous random variable.
exponweib	An exponentiated Weibull continuous random variable.
exponpow	An exponential power continuous random variable.
f	An F continuous random variable.
fatiguelife	A fatigue-life (Birnbaum-Saunders) continuous random variable.
fisk	A Fisk continuous random variable.
foldcauchy	A folded Cauchy continuous random variable.
foldnorm	A folded normal continuous random variable.
frechet_r	A Frechet right (or Weibull minimum) continuous random variable
frechet_l	A Frechet left (or Weibull maximum) continuous random variable
genlogistic	A generalized logistic continuous random variable.
gennorm	A generalized normal continuous random variable.
genpareto	A generalized Pareto continuous random variable.
genexpon	A generalized exponential continuous random variable.
genextreme	A generalized extreme value continuous random variable.
gausshyper	A Gauss hypergeometric continuous random variable.
gamma	A gamma continuous random variable.
gengamma	A generalized gamma continuous random variable.
genhalflogistic	A generalized half-logistic continuous random variable.
gilbrat	A Gilbrat continuous random variable.
gompertz	A Gompertz (or truncated Gumbel) continuous random variable.
gumbel_r	A right-skewed Gumbel continuous random variable.
gumbel_l	A left-skewed Gumbel continuous random variable.
halfcauchy	A Half-Cauchy continuous random variable.
halflogistic	A half-logistic continuous random variable.
halfnorm	A half-normal continuous random variable.
halfgennorm	The upper half of a generalized normal continuous random variable.
hypsecant	A hyperbolic secant continuous random variable.
invgamma	An inverted gamma continuous random variable.
invgauss	An inverse Gaussian continuous random variable.
invweibull	An inverted Weibull continuous random variable.
johnsonsb	A Johnson SB continuous random variable.
johnsonsu	A Johnson SU continuous random variable.
kappa4	Kappa 4 parameter distribution.
kappa3	Kappa 3 parameter distribution.
ksone	General Kolmogorov-Smirnov one-sided test.
kstwobign	Kolmogorov-Smirnov two-sided test for large N.
laplace	A Laplace continuous random variable.
levy	A Levy continuous random variable.
levy_l	A left-skewed Levy continuous random variable.
levy_stable	A Levy-stable continuous random variable.
logistic	A logistic (or Sech-squared) continuous random variable.
loggamma	A log gamma continuous random variable.
loglaplace	A log-Laplace continuous random variable.
lognorm	A lognormal continuous random variable.
lomax	A Lomax (Pareto of the second kind) continuous random variable.
maxwell	A Maxwell continuous random variable.
mielke	A Mielke Beta-Kappa continuous random variable.
moyal	A Moyal continuous random variable.
nakagami	A Nakagami continuous random variable.
ncx2	A non-central chi-squared continuous random variable.
ncf	A non-central F distribution continuous random variable.
nct	A non-central Student's t continuous random variable.
norm	A normal continuous random variable.
norminvgauss	A Normal Inverse Gaussian continuous random variable.
pareto	A Pareto continuous random variable.
pearson3	A pearson type III continuous random variable.
powerlaw	A power-function continuous random variable.
powerlognorm	A power log-normal continuous random variable.
powernorm	A power normal continuous random variable.
rdist	An R-distributed continuous random variable.
reciprocal	A reciprocal continuous random variable.
rayleigh	A Rayleigh continuous random variable.
rice	A Rice continuous random variable.
recipinvgauss	A reciprocal inverse Gaussian continuous random variable.
semicircular	A semicircular continuous random variable.
skewnorm	A skew-normal random variable.
t	A Student's t continuous random variable.
trapz	A trapezoidal continuous random variable.
triang	A triangular continuous random variable.
truncexpon	A truncated exponential continuous random variable.
truncnorm	A truncated normal continuous random variable.
tukeylambda	A Tukey-Lamdba continuous random variable.
uniform	A uniform continuous random variable.
vonmises	A Von Mises continuous random variable.
vonmises_line	A Von Mises continuous random variable.
wald	A Wald continuous random variable.
weibull_min	Weibull minimum continuous random variable.
weibull_max	Weibull maximum continuous random variable.
wrapcauchy	A wrapped Cauchy continuous random variable.

Multivariate distributions

multivariate_normal	A multivariate normal random variable.
matrix_normal	A matrix normal random variable.
dirichlet	A Dirichlet random variable.
wishart	A Wishart random variable.
invwishart	An inverse Wishart random variable.
multinomial	A multinomial random variable.
special_ortho_group	A matrix-valued SO(N) random variable.
ortho_group	A matrix-valued O(N) random variable.
unitary_group	A matrix-valued U(N) random variable.
random_correlation	A random correlation matrix.

Discrete distributions

bernoulli	A Bernoulli discrete random variable.
binom	A binomial discrete random variable.
boltzmann	A Boltzmann (Truncated Discrete Exponential) random variable.
dlaplace	A Laplacian discrete random variable.
geom	A geometric discrete random variable.
hypergeom	A hypergeometric discrete random variable.
logser	A Logarithmic (Log-Series, Series) discrete random variable.
nbinom	A negative binomial discrete random variable.
planck	A Planck discrete exponential random variable.
poisson	A Poisson discrete random variable.
randint	A uniform discrete random variable.
skellam	A Skellam discrete random variable.
zipf	A Zipf discrete random variable.
yulesimon	A Yule-Simon discrete random variable.

scipy.stats

Summary statistics

describe(a[, axis, ddof, bias, nan_policy])	Compute several descriptive statistics of the passed array.
gmean(a[, axis, dtype])	Compute the geometric mean along the specified axis.
hmean(a[, axis, dtype])	Calculate the harmonic mean along the specified axis.
kurtosis(a[, axis, fisher, bias, nan_policy])	Compute the kurtosis (Fisher or Pearson) of a dataset.
mode(a[, axis, nan_policy])	Return an array of the modal (most common) value in the passed array.
moment(a[, moment, axis, nan_policy])	Calculate the nth moment about the mean for a sample.
skew(a[, axis, bias, nan_policy])	Compute the sample skewness of a data set.
kstat(data[, n])	Return the nth k-statistic (1<=n<=4 so far).
kstatvar(data[, n])	Returns an unbiased estimator of the variance of the k-statistic.
tmean(a[, limits, inclusive, axis])	Compute the trimmed mean.
tvar(a[, limits, inclusive, axis, ddof])	Compute the trimmed variance.
tmin(a[, lowerlimit, axis, inclusive, ...])	Compute the trimmed minimum.
tmax(a[, upperlimit, axis, inclusive, ...])	Compute the trimmed maximum.
tstd(a[, limits, inclusive, axis, ddof])	Compute the trimmed sample standard deviation.
tsem(a[, limits, inclusive, axis, ddof])	Compute the trimmed standard error of the mean.
variation(a[, axis, nan_policy])	Compute the coefficient of variation, the ratio of the biased standard deviation to the mean.
find_repeats(arr)	Find repeats and repeat counts.
trim_mean(a, proportiontocut[, axis])	Return mean of array after trimming distribution from both tails.
gstd(a[, axis, ddof])	Calculate the geometric standard deviation of an array
iqr(x[, axis, rng, scale, nan_policy, ...])	Compute the interquartile range of the data along the specified axis.
sem(a[, axis, ddof, nan_policy])	Calculate the standard error of the mean (or standard error of measurement) of the values in the input array.
bayes_mvs(data[, alpha])	Bayesian confidence intervals for the mean, var, and std.
mvsdist(data)	'Frozen' distributions for mean, variance, and standard deviation of data.
entropy(pk[, qk, base])	Calculate the entropy of a distribution for given probability values.
median_absolute_deviation(x[, axis, center, ...])	Compute the median absolute deviation of the data along the given axis.

Frequency statistics

cumfreq(a[, numbins, defaultreallimits, weights])	Return a cumulative frequency histogram, using the histogram function.
itemfreq(*args, **kwds)	itemfreq is deprecated! itemfreq is deprecated and will be removed in a future version.
percentileofscore(a, score[, kind])	The percentile rank of a score relative to a list of scores.
scoreatpercentile(a, per[, limit, ...])	Calculate the score at a given percentile of the input sequence.
relfreq(a[, numbins, defaultreallimits, weights])	Return a relative frequency histogram, using the histogram function.

binned_statistic(x, values[, statistic, ...])	Compute a binned statistic for one or more sets of data.
binned_statistic_2d(x, y, values[, ...])	Compute a bidimensional binned statistic for one or more sets of data.
binned_statistic_dd(sample, values[, ...])	Compute a multidimensional binned statistic for a set of data.

Correlation functions

f_oneway(*args)	Performs a 1-way ANOVA.
pearsonr(x, y)	Pearson correlation coefficient and p-value for testing non-correlation.
spearmanr(a[, b, axis, nan_policy])	Calculate a Spearman rank-order correlation coefficient and the p-value to test for non-correlation.
pointbiserialr(x, y)	Calculate a point biserial correlation coefficient and its p-value.
kendalltau(x, y[, initial_lexsort, ...])	Calculate Kendall's tau, a correlation measure for ordinal data.
weightedtau(x, y[, rank, weigher, additive])	Compute a weighted version of Kendall's τ.
linregress(x[, y])	Calculate a linear least-squares regression for two sets of measurements.
siegelslopes(y[, x, method])	Computes the Siegel estimator for a set of points (x, y).
theilslopes(y[, x, alpha])	Computes the Theil-Sen estimator for a set of points (x, y).

Statistical tests

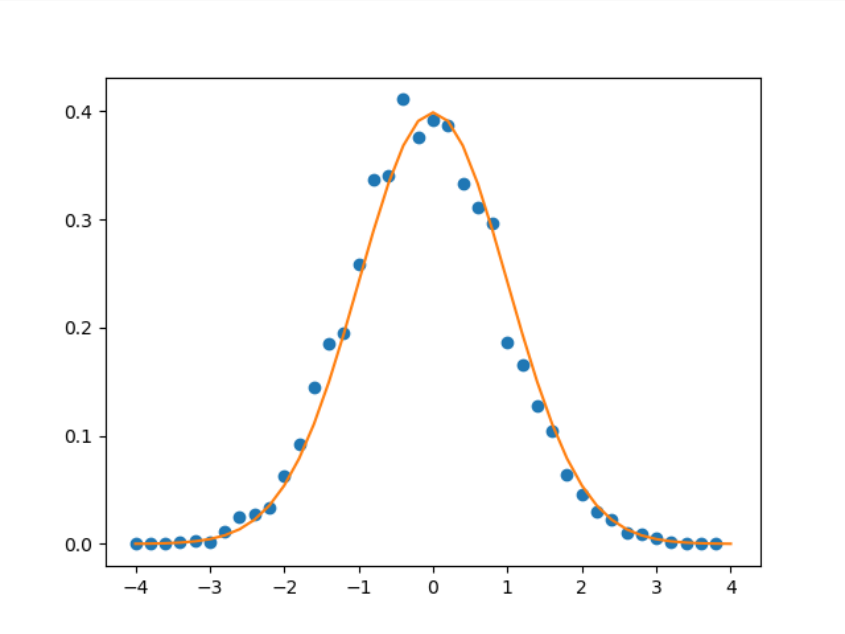
ttest_1samp(a, popmean[, axis, nan_policy])	Calculate the T-test for the mean of ONE group of scores.
ttest_ind(a, b[, axis, equal_var, nan_policy])	Calculate the T-test for the means of *two independent* samples of scores.
ttest_ind_from_stats(mean1, std1, nobs1, ...)	T-test for means of two independent samples from descriptive statistics.
ttest_rel(a, b[, axis, nan_policy])	Calculate the T-test on TWO RELATED samples of scores, a and b.
kstest(rvs, cdf[, args, N, alternative, mode])	Perform the Kolmogorov-Smirnov test for goodness of fit.
chisquare(f_obs[, f_exp, ddof, axis])	Calculate a one-way chi square test.
power_divergence(f_obs[, f_exp, ddof, axis, ...])	Cressie-Read power divergence statistic and goodness of fit test.
ks_2samp(data1, data2[, alternative, mode])	Compute the Kolmogorov-Smirnov statistic on 2 samples.
epps_singleton_2samp(x, y[, t])	Compute the Epps-Singleton (ES) test statistic.
mannwhitneyu(x, y[, use_continuity, alternative])	Compute the Mann-Whitney rank test on samples x and y.
tiecorrect(rankvals)	Tie correction factor for ties in the Mann-Whitney U and Kruskal-Wallis H tests.
rankdata(a[, method])	Assign ranks to data, dealing with ties appropriately.
ranksums(x, y)	Compute the Wilcoxon rank-sum statistic for two samples.
wilcoxon(x[, y, zero_method, correction, ...])	Calculate the Wilcoxon signed-rank test.
kruskal(*args, **kwargs)	Compute the Kruskal-Wallis H-test for independent samples
friedmanchisquare(*args)	Compute the Friedman test for repeated measurements
brunnermunzel(x, y[, alternative, ...])	Computes the Brunner-Munzel test on samples x and y
combine_pvalues(pvalues[, method, weights])	Methods for combining the p-values of independent tests bearing upon the same hypothesis.
jarque_bera(x)	Perform the Jarque-Bera goodness of fit test on sample data.

ansari(x, y)	Perform the Ansari-Bradley test for equal scale parameters
bartlett(*args)	Perform Bartlett's test for equal variances
levene(*args, **kwds)	Perform Levene test for equal variances.
shapiro(x)	Perform the Shapiro-Wilk test for normality.
anderson(x[, dist])	Anderson-Darling test for data coming from a particular distribution
anderson_ksamp(samples[, midrank])	The Anderson-Darling test for k-samples.
binom_test(x[, n, p, alternative])	Perform a test that the probability of success is p.
fligner(*args, **kwds)	Perform Fligner-Killeen test for equality of variance.

scipy.stats – Exemplo

Estimando uma PDF Gaussiana

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

samples = np.random.normal(size=5000)
bins = np.arange(-4.0, 4.2, 0.2)
histogram = np.histogram(samples, bins=bins, normed=True)[0]
pdf = stats.norm.pdf(bins) # norm is a distribution object
plt.plot(bins[:-1], histogram, 'o')
plt.plot(bins, pdf, '-')
plt.show()
```



scipy.integrate – Rotinas de Integração Numérica

Rotinas para integração de funções

- quad – rotinas de propósito geral.
- dblquad – rotinas de integração dupla.
- tplquad – rotinas de integração tripla.
- fixed_quad – Integra f(x) com quadratura Gaussiana de ordem n.
- quadrature – Integra f(x) com quadratura Gaussiana de ordem n com dada tolerância
- romberg – Integra f(x) usando integração de Romberg

Metódos de integração a partir de pontos

- trapz – método trapezoidal pra calcular integral de amostras
- cumtrapz – método trapezoidal cumulativo pra calcular integral de amostras
- simps – método Simpson pra calcular integral de amostras
- romb – método Romberg pra calcular integral de amostras (2**k + 1) igualmente espaçadas.

Integração de EDOs

- odeint – integração geral
- ode – integração com otinas VODE e ZVODE

Calcula

$$\int_0^4 x^2 \, dx = \frac{4^3}{3}$$

e xompara com o valor numérico

```
>>> from scipy import integrate
>>> x2 = lambda x: x**2
>>> integrate.quad(x2, 0, 4)
    (21.333333333333332, 2.3684757858670003e-13)
>>> print(4**3 / 3.)  # analytical result
    21.3333333333
```

$$\int_0^4 x^2 \, dx = \frac{4^3}{3}$$

FIM

FIM

mais em `www.ferrari.pro.br`