Curso básico de Python para estudantes de Física

Germán A. Racca

Universidade do Estado do Rio Grande do Norte
Faculdade de Ciências Exatas e Naturais
Departamento de Física
Mossoró - RN

22 de Junho de 2016

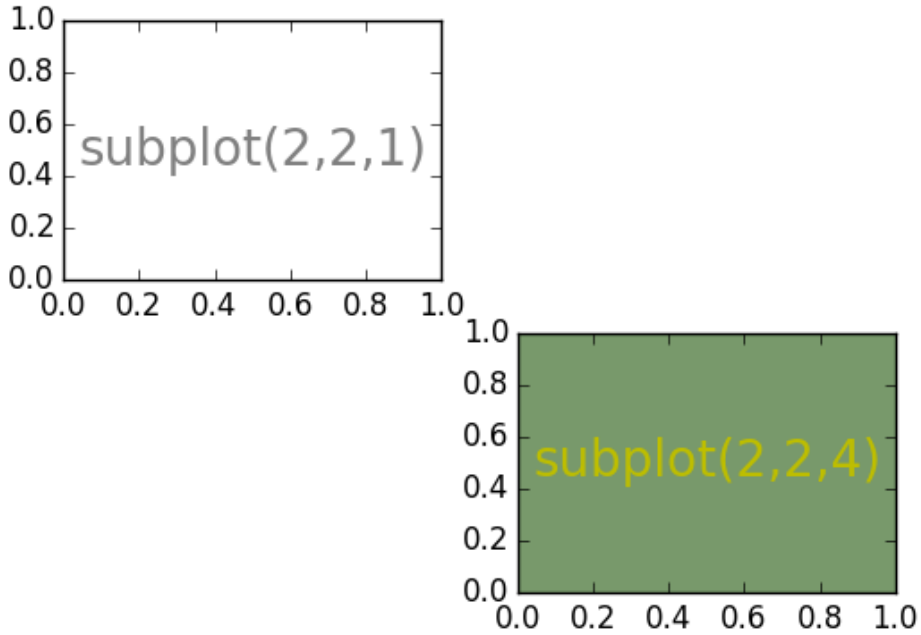
Tópicos

Tópicos

Tópicos

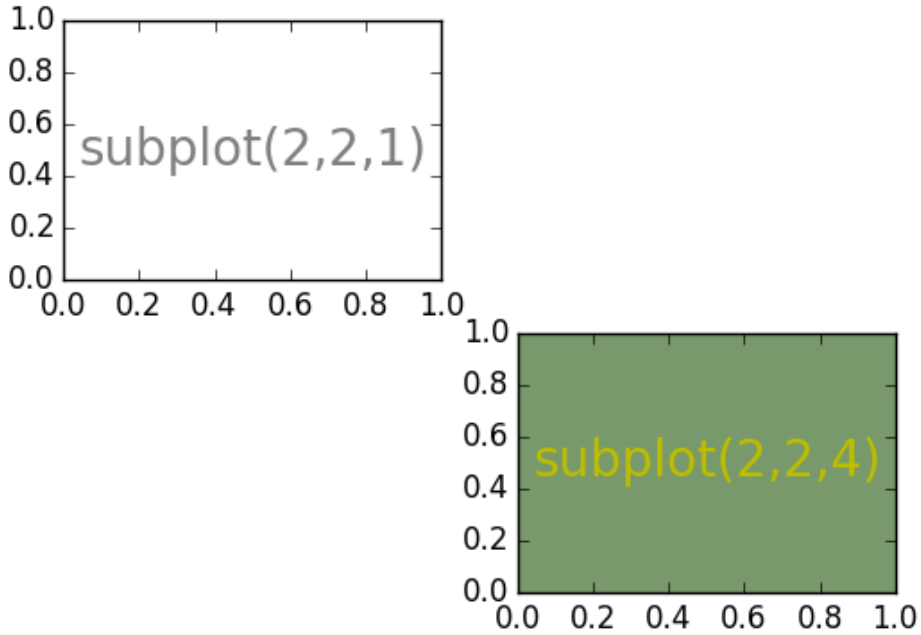
Gráficos múltiplos com subplot

```
>>> import matplotlib.pyplot as plt
>>> plt.figure(figsize=(6, 4))
>>> plt.subplot(2, 2, 1)
>>> plt.text(0.5, 0.5,
...          "subplot(2,2,1)",
...          horizontalalignment='center',
...          verticalalignment='center',
...          fontsize=20,
...          alpha=0.5)
>>> plt.subplot(2, 2, 4, axisbg=(0.47, 0.60, 0.42))
>>> plt.text(0.5, 0.5,
...          "subplot(2,2,4)",
...          ha='center',
...          va='center',
...          fontsize=20,
...          color="y")
>>> plt.show()
```



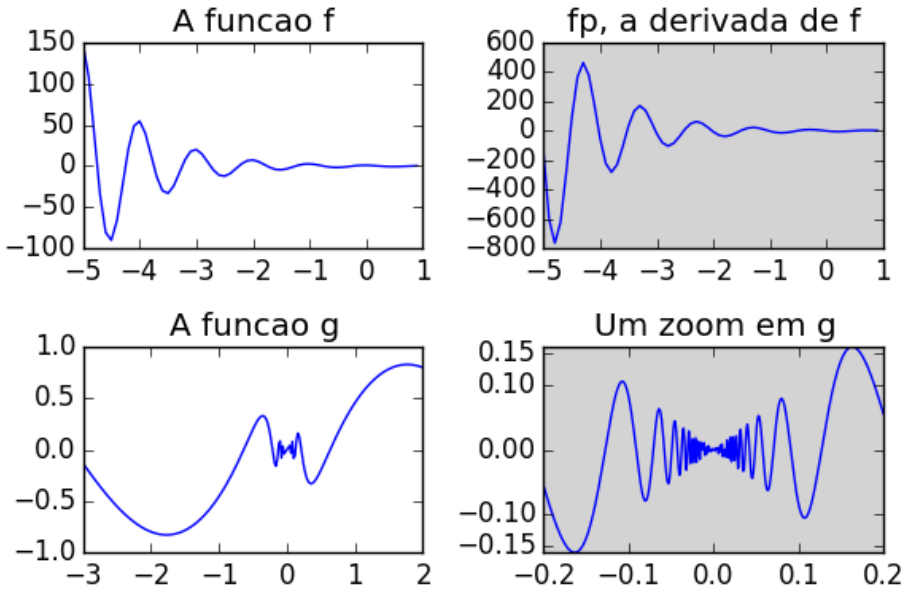
Gráficos múltiplos com subplot

```
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure(figsize=(6, 4))
>>> ax1 = fig.add_subplot(2, 2, 1)
>>> ax1.text(0.5, 0.5,
...          "subplot(2,2,1)",
...          ha='center',
...          va='center',
...          fontsize=20,
...          alpha=0.5)
>>> ax2 = fig.add_subplot(2, 2, 4, axisbg=(0.47, 0.60, 0.42))
>>> ax2.text(0.5, 0.5,
...          "subplot(2,2,4)",
...          ha='center',
...          va='center',
...          fontsize=20,
...          color="y")
>>> plt.show()
```



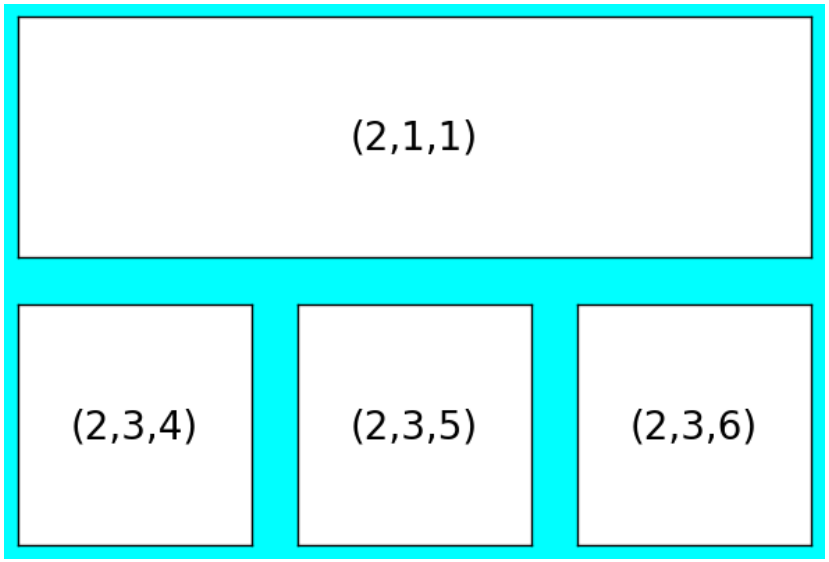
Gráficos múltiplos com subplot

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> def f(t):
...     return np.exp(-t)*np.cos(2*np.pi*t)
>>> def fp(t):
...     return -2*np.pi*np.exp(-t)*np.sin(2*np.pi*t) - np.e**(-t)*np.cos(2*np.pi*t)
>>> def g(t):
...     return np.sin(t)*np.cos(1/(t))
>>> fig = plt.figure(figsize=(6, 4))
>>> t = np.arange(-5.0, 1.0, 0.1)
>>> ax1 = fig.add_subplot(2, 2, 1)
>>> ax1.set_title("A funcao f")
>>> ax1.plot(t, f(t))
>>> ax2 = fig.add_subplot(2, 2, 2, axisbg="lightgrey")
>>> ax2.set_title("fp, a derivada de f")
>>> ax2.plot(t, fp(t))
>>> t = np.arange(-3.0, 2.0, 0.02)
>>> ax3 = fig.add_subplot(2, 2, 3)
>>> ax3.set_title("A funcao g")
>>> ax3.plot(t, g(t))
>>> t = np.arange(-0.2, 0.201, 0.001)
>>> ax4 = fig.add_subplot(2, 2, 4, axisbg="lig
>>> ax4.set_title('Um zoom em g')
>>> ax4.set_xticks([-0.2, -0.1, 0, 0.1, 0.2])
>>> ax4.set_yticks([-0.15, -0.1, 0, 0.1, 0.15]
>>> ax4.plot(t, g(t))
>>> plt.tight_layout()
>>> plt.show()
```



Gráficos múltiplos com subplot

```
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure(figsize=(6,4))
>>> fig.subplots_adjust(bottom=0.025, left=0.025, top=0.975, right=0.975)
>>> X = [(2, 1, 1), (2, 3, 4), (2, 3, 5), (2, 3, 6)]
>>> for nrows, ncols, plot_number in X:
...     texto = '(' + str(nrows) + ',' + str(ncols) + ',' + str(plot_number) + ')'
...     ax = fig.add_subplot(nrows, ncols, plot_number)
...     ax.text(0.5, 0.5, texto, ha='center', va='center', fontsize=20)
...     ax.set_xticks([])
...     ax.set_yticks([])
>>  fig.savefig('/tmp/fig.png', facecolor='cyan', bbox_inches='tight')
>>> plt.show()
```



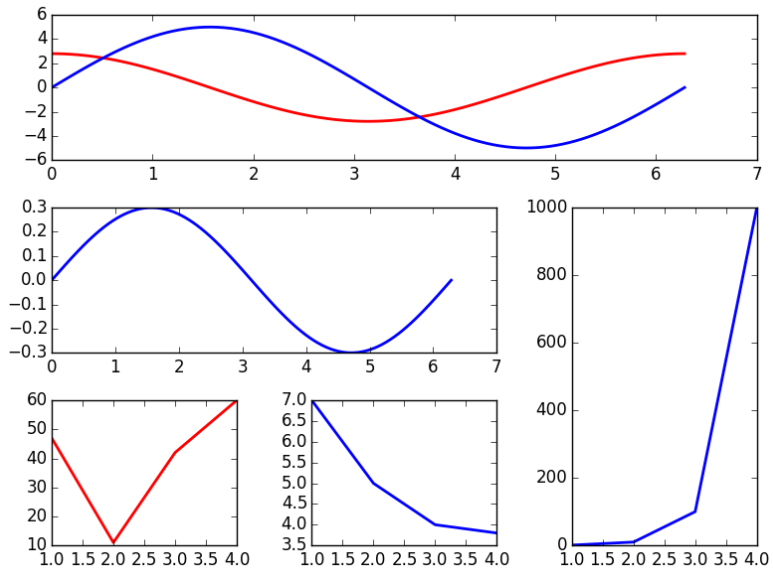
Tópicos

Gráficos múltiplos com gridspec

```
>>> from matplotlib import gridspec
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure(figsize=(6, 4))
>>> gs = gridspec.GridSpec(3, 3)
>>> ax1 = fig.add_subplot(gs[0, :])
>>> ax1.set_xticks(())
>>> ax1.set_yticks(())
>>> ax1.text(0.5, 0.5, 'Axes 1', ha='center', va='center', size=24, alpha=.5)
>>> ax2 = fig.add_subplot(gs[1, :2])  # gs[1, :-1]
>>> ax2.set_xticks(())
>>> ax2.set_yticks(())
>>> ax2.text(0.5, 0.5, 'Axes 2', ha='center', va='center', size=24, alpha=.5)
>>> ax3 = fig.add_subplot(gs[1:, 2])  # gs[1:, -1]
>>> ax3.set_xticks(())
>>> ax3.set_yticks(())
>>> ax3.text(0.5, 0.5, 'Axes 3', ha='center', va='center', size=24, alpha=.5)
>>> ax4 = fig.add_subplot(gs[2, 0])  # gs[-1, 0]
>>> ax4.set_xticks(())
>>> ax4.set_yticks(())
>>> ax4.text(0.5, 0.5, 'Axes 4', ha='center', va='center',
>>> ax5 = fig.add_subplot(gs[2, 1])  # gs[-1, -2]
>>> ax5.set_xticks(())
>>> ax5.set_yticks(())
>>> ax5.text(0.5, 0.5, 'Axes 5', ha='center', va='center',
>>> fig.tight_layout()
>>> fig.savefig('/tmp/fig.png', facecolor='magenta')
>>> plt.show()
```

Gráficos múltiplos com gridspec

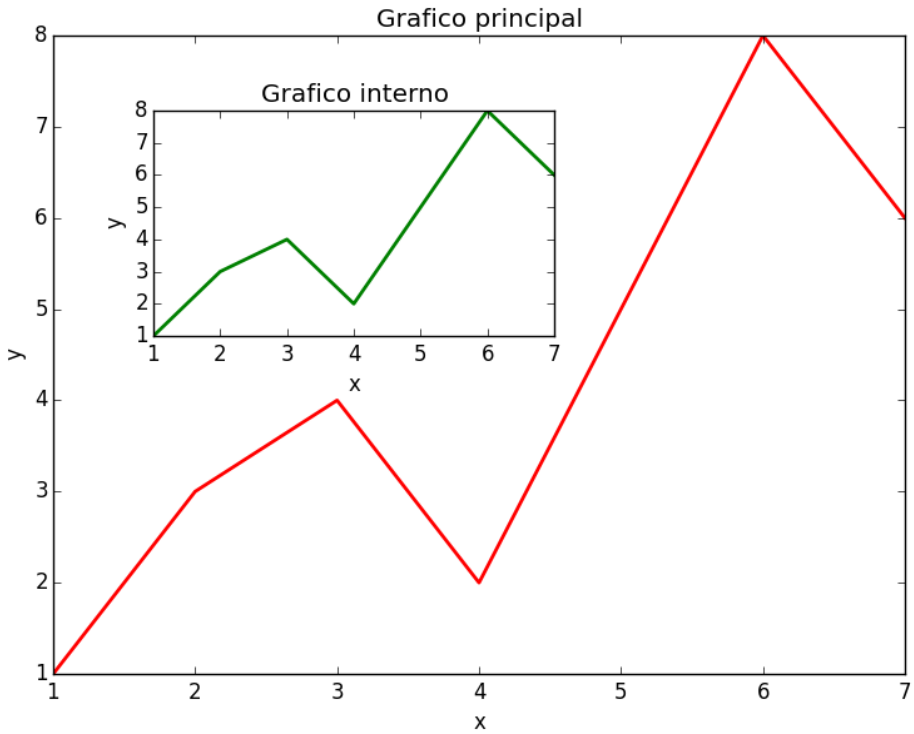
```
>>> import numpy as np
>>> from matplotlib import gridspec
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure(figsize=(8, 6))
>>> gs = gridspec.GridSpec(3, 3)
>>> X = np.linspace(0, 2*np.pi, 100, endpoint=True)
>>> F1 = 2.8*np.cos(X)
>>> F2 = 5*np.sin(X)
>>> F3 = 0.3*np.sin(X)
>>> ax1 = fig.add_subplot(gs[0, :])
>>> ax1.plot(X, F1, 'r-', X, F2, lw=2)
>>> ax2 = fig.add_subplot(gs[1, :2])
>>> ax2.plot(X, F3, lw=2)
>>> ax3 = fig.add_subplot(gs[1:, 2])
>>> ax3.plot([1, 2, 3, 4], [1, 10, 100, 1000], 'b-', lw=2)
>>> ax4 = fig.add_subplot(gs[2, 0])
>>> ax4.plot([1, 2, 3, 4], [47, 11, 42, 60], 'r-', lw=
>>> ax5 = fig.add_subplot(gs[2, 1])
>>> ax5.plot([1, 2, 3, 4], [7, 5, 4, 3.8], lw=2)
>>> fig.tight_layout()
>>> fig.savefig('/tmp/fig.png')
>>> plt.show()
```



Tópicos

Um gráfico dentro de outro

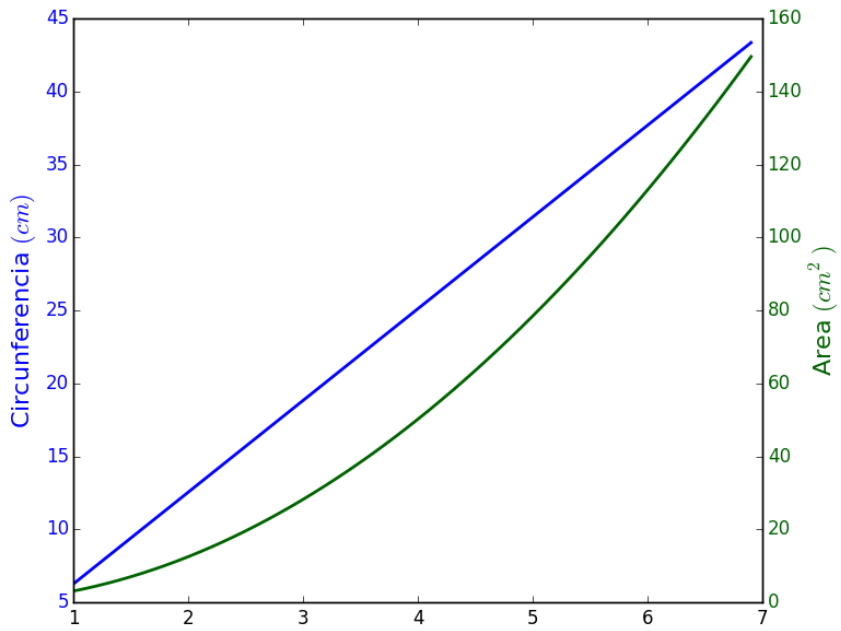
```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> X = [1, 2, 3, 4, 5, 6, 7]
>>> Y = [1, 3, 4, 2, 5, 8, 6]
>>> ax1 = fig.add_axes([0.1, 0.1, 0.85, 0.85])  # axes principal
>>> ax2 = fig.add_axes([0.2, 0.55, 0.4, 0.3])   # axes interno
>>> # figura principal
>>> ax1.plot(X, Y, 'r', lw=2)
>>> ax1.set_xlabel('x')
>>> ax1.set_ylabel('y')
>>> ax1.set_title('Grafico principal')
>>> # grafico interno
>>> ax2.plot(X, Y, 'g', lw=2)
>>> ax2.set_xlabel('x')
>>> ax2.set_ylabel('y')
>>> ax2.set_title('Grafico interno')
>>> plt.show()
```



Tópicos

Clonando um eixo

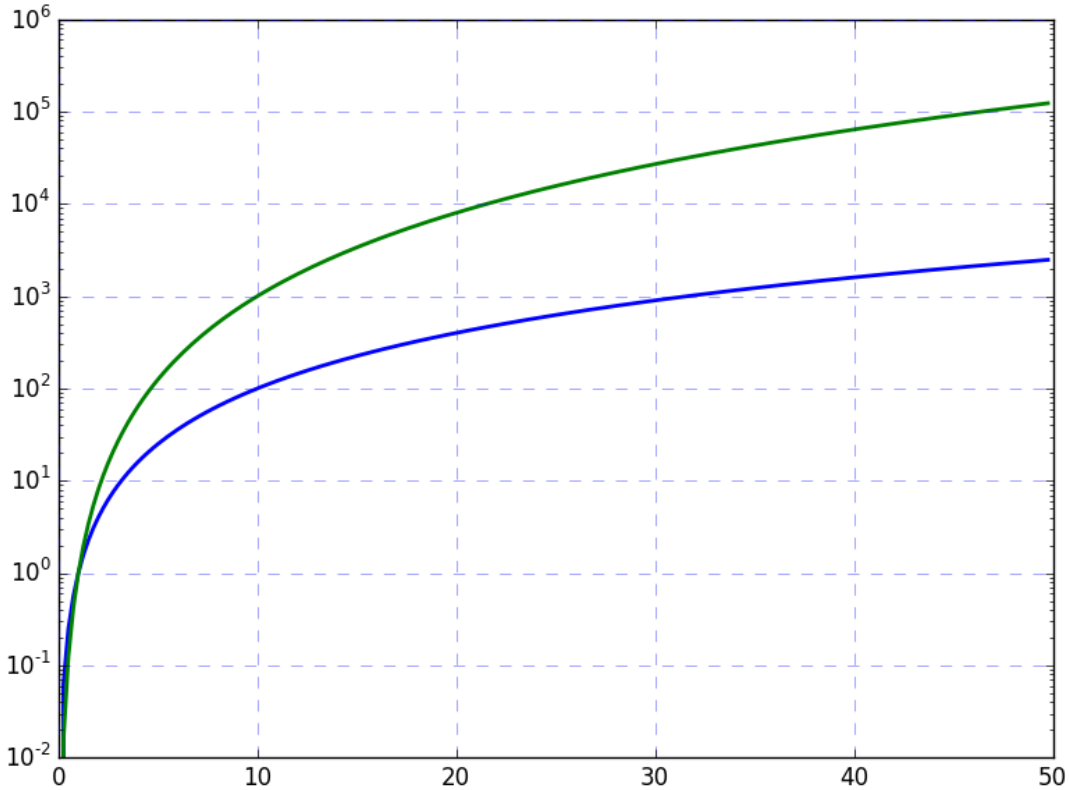
```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> x = np.arange(1, 7, 0.1)
>>> fig = plt.figure()
>>> ax1 = fig.add_subplot(1, 1, 1)
>>> ax1.plot(x, 2*np.pi*x, lw=2, color="blue")
>>> ax1.set_ylabel(r"Circunferencia $(cm)$", fontsize=16, color="blue")
>>> for label in ax1.get_yticklabels():
...     label.set_color("blue")
>>> ax2 = ax1.twinx()
>>> ax2.plot(x, np.pi*x**2, lw=2, color="darkgreen")
>>> ax2.set_ylabel(r"Area $(cm^2)$", fontsize=16, color="darkgreen")
>>> for label in ax2.get_yticklabels():
...     label.set_color("darkgreen")
>>> plt.show()
```



Tópicos

Eixo logarítmico e grade

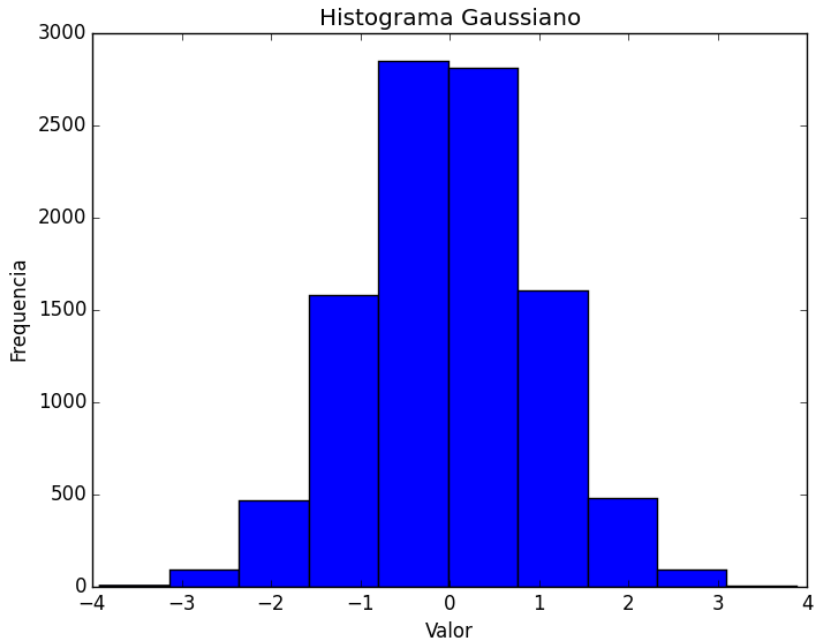
```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> x = np.arange(0, 50, 0.25)
>>> fig = plt.figure()
>>> ax = fig.add_subplot(1, 1, 1)
>>> ax.plot(x, x**2, x, x**3, lw=2)
>>> ax.set_yscale('log')
>>> ax.grid(color='b', alpha=0.5, ls='--', lw=0.5)
>>> plt.show()
```



Tópicos

Histogramas

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> g = np.random.normal(size=10000)
>>> plt.hist(g)
>>> plt.title("Histograma Gaussiano")
>>> plt.xlabel("Valor")
>>> plt.ylabel("Frequencia")
>>> plt.show()
```

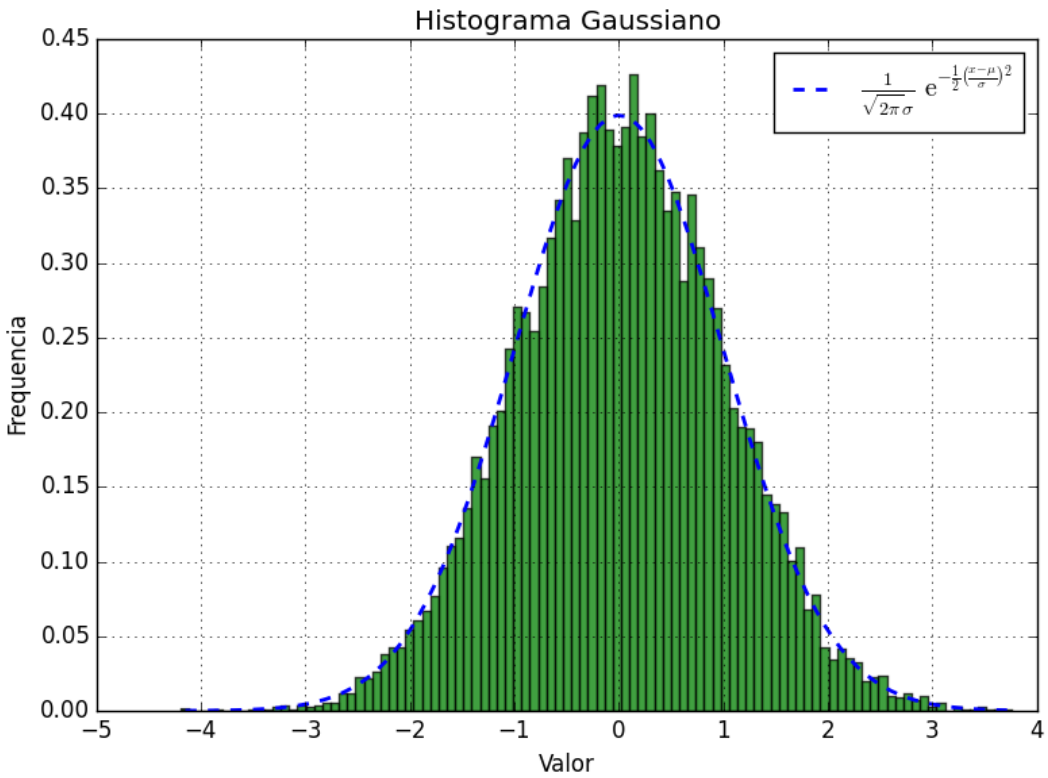



```
>>> n, bins, patches = plt.hist(g)
>>> print n
[    7.    92.   466.  1579.  2852.  2814.  1604.   484.    94.     8.]
>>> print bins
[-3.91812382 -3.13810208 -2.35808034 -1.57805859 -0.79803685 -0.01801511
  0.76200663  1.54202838  2.32205012  3.10207186  3.8820936]
>>> print bins[1] - bins[0]
0.780021742792
>>> print patches
<a list of 10 Patch objects>
>>> print patches[0]
Rectangle(-3.91812,0;0.780022x7)
```

Histogramas

```
>>> import numpy as np
>>> import matplotlib.mlab as mlab
>>> import matplotlib.pyplot as plt
>>> g = np.random.normal(size=10000)
>>> fig = plt.figure()
>>> ax = fig.add_subplot(1, 1, 1)
>>> n, bins, patches = ax.hist(g, bins=100, normed=True, facecolor='green', alpha=0.75)
>>> centros = 0.5*(bins[1:] + bins[:-1])
>>> y = mlab.normpdf(centros, 0, 1)
>>> ax.plot(centros, y, 'b--', lw=2, label=lab)
>>> ax.set_title("Histograma Gaussiano")
>>> ax.set_xlabel("Valor")
>>> ax.set_ylabel("Frequencia")
>>> ax.legend()
>>> ax.grid()
>>> plt.show()
```

```
lab =
r"$\frac{1}{\sqrt{2\pi}\sigma} \,
  \mathrm{e}^{-\frac{1}{2} \left(
  \frac{x - \mu}{\sigma} \right)^2}$"
```



Tópicos

SciPy

SciPy

- Pacote principal de rotinas científicas em Python
- Ferramentas dedicadas a problemas comuns em computação científica
- Opera de forma eficiente em *arrays* do NumPy (trabalham lado a lado)
- Diferentes sub-módulos correspondem a diferentes aplicações (interpolação, otimização, etc)

Forma de importar os sub-módulos:

```
>>> import numpy as np
>>> import scipy as sp
>>> sp.array is np.array
True
>>> sp.cos is np.cos
True

>>> from scipy import algum_modulo
>>> algum_modulo.alguma_funcao()

>>> from scipy import stats
>>> stats.pearsonr([1, 2, 3], [4, 5, 6])
(1.0, 0.0)
```

SciPy

Sub-módulo	Descrição
`scipy.cluster`	Quantização de vetores (Kmeans)
`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`	Regressão ortogonal de distância
`scipy.optimize`	Otimização
`scipy.signal`	Processamento de sinais
`scipy.sparse`	Matrizes esparsas
`scipy.spatial`	Estrutura espacial de dados e algoritmos
`scipy.special`	Funções matemáticas especiais
`scipy.stats`	Estatística

Tópicos

Álgebra linear

- A função `linalg.det` calcula o determinante de uma matriz quadrada:

```
>>> import numpy as np
>>> from scipy import linalg
>>> arr = np.array([[1, 2],
...                 [3, 4]])
>>> linalg.det(arr)
-2.0
```

- A função `linalg.inv` calcula a inversa de uma matriz quadrada:

```
>>> linalg.inv(arr)
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

- A funçõa `linalg.svd` calcula a decomposição por valores singulares:

```
>>> arr = np.arange(9).reshape((3, 3)) + np.diag([1, 0, 1])
>>> uarr, spec, vharr = linalg.svd(arr)
>>> spec
array([ 14.88982544,   0.45294236,   0.29654967])
>>> sarr = np.diag(spec)
>>> svd_mat = uarr.dot(sarr).dot(vharr)
>>> svd_mat
array([[ 1.,  1.,  2.],
       [ 3.,  4.,  5.],
       [ 6.,  7.,  9.]])
```

Álgebra linear

- A funçõa `linalg.solve` resolve a equação $\mathrm{A} \cdot \mathrm{X} = \mathrm{B}$:

$$\left.\begin{aligned} x + 3y + 5z &= 10 \\ 2x + 5y + z &= 8 \\ 2x + 3y + 8z &= 3 \end{aligned}\right\} \Rightarrow \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix}^{-1} \begin{bmatrix} 10 \\ 8 \\ 3 \end{bmatrix} = \frac{1}{25} \begin{bmatrix} -232 \\ 129 \\ 19 \end{bmatrix} = \begin{bmatrix} -9.28 \\ 5.16 \\ 0.76 \end{bmatrix}$$

```
>>> import numpy as np
>>> from scipy import linalg
>>> A = np.array([[1, 3, 5], [2, 5, 1], [2, 3, 8]])
>>> B = np.array([[10], [8], [3]])
>>> linalg.inv(A).dot(B)  # metodo lento
array([[-9.28],
       [ 5.16],
       [ 0.76]])
>>> linalg.solve(A, B)  # metodo rapido
array([[-9.28],
       [ 5.16],
       [ 0.76]])
>>> A.dot(linalg.solve(A, B)) - B
array([[  0.00000000e+00],
       [ -1.77635684e-15],
       [ -1.77635684e-15]])
```

Tópicos

Otimização e ajustes

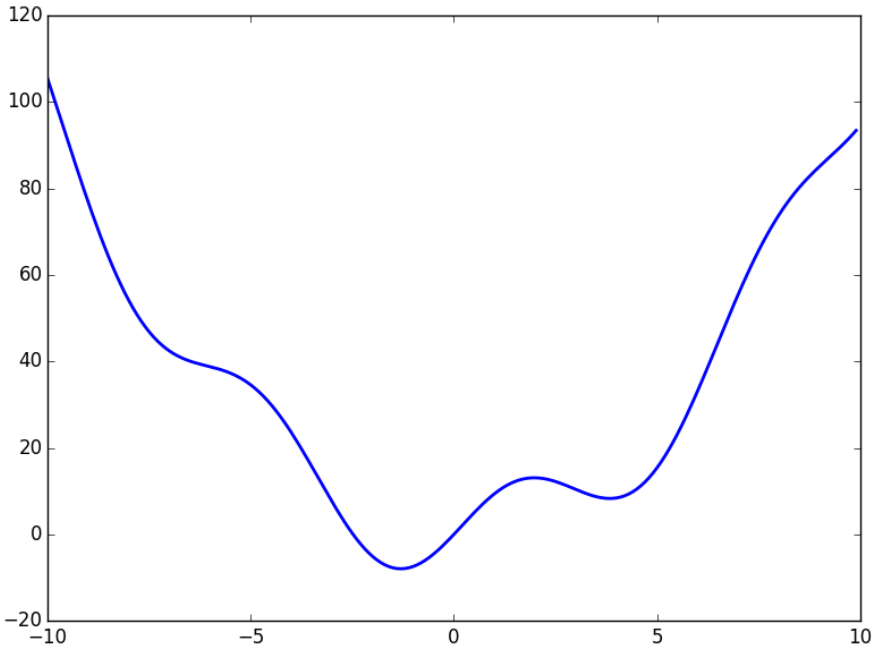
- Encontrando o mínimo de uma função escalar:

```
>>> import numpy as np
>>> from scipy import optimize
>>> import matplotlib.pyplot as plt

>>> def f(x):
...     return x**2 + 10*np.sin(x)
>>> x = np.arange(-10, 10, 0.1)
>>> plt.plot(x, f(x), lw=2)
>>> plt.show()

>>> # metodo de Broyden, Fletcher,
>>> # Goldfarb e Shanno (BFGS)
>>> optimize.fmin_bfgs(f, 0)
Optimization terminated successfully.
         Current function value: -7.945823
         Iterations: 5
         Function evaluations: 24
         Gradient evaluations: 8
array([-1.30644003])

>>> optimize.fmin_bfgs(f, 3, disp=0)
array([ 3.83746663])
```



Otimização e ajustes

- Mínimo global:

```
>>> grid = (-10, 10, 0.1)
>>> optimize.brute(f, (grid,))  # metodo da forca bruta
array([-1.30641113])

>>> optimize.brent(f)  # metodo de Brent
-1.3064400120612139

>>> optimize.basinhopping(f, 0)  # algoritmo de basin-hopping
                    nfev: 1659
minimization_failures: 0
                     fun: -7.945823375615284 5
                       x: array([-1.30644001])
                 message: ['requested number of basinhopping iterations completed succe
                    njev: 553
                     nit: 100
```

- Mínimo local:

```
>>> optimize.fminbound(f, 0, 10)  # metodo de Brent no intervalo
3.8374671194983834
```

Otimização e ajustes

- Encontrando as raízes de uma função escalar:

```
>>> optimize.fsolve(f, 1)
array([ 0.])
>>> optimize.fsolve(f, -2.5)
array([-2.47948183])
```

- Ajustando uma curva:

```
>>> def f2(x, a, b):
...     return a*x**2 + b*np.sin(x)
>>> xdata = np.linspace(-10, 10, 20)
>>> ydata = f(xdata) + np.random.randn(xdata.size)

>>> chute = [2, 2]
>>> param, param_covar = optimize.curve_fit(f2, xdata, ydata, chute)
>>> param
array([ 0.99916269,  9.70101786])
>>> param_covar
array([[  1.66934910e-05,  -1.56347239e-11],
       [ -1.56347239e-11,   8.52573326e-02]])
```

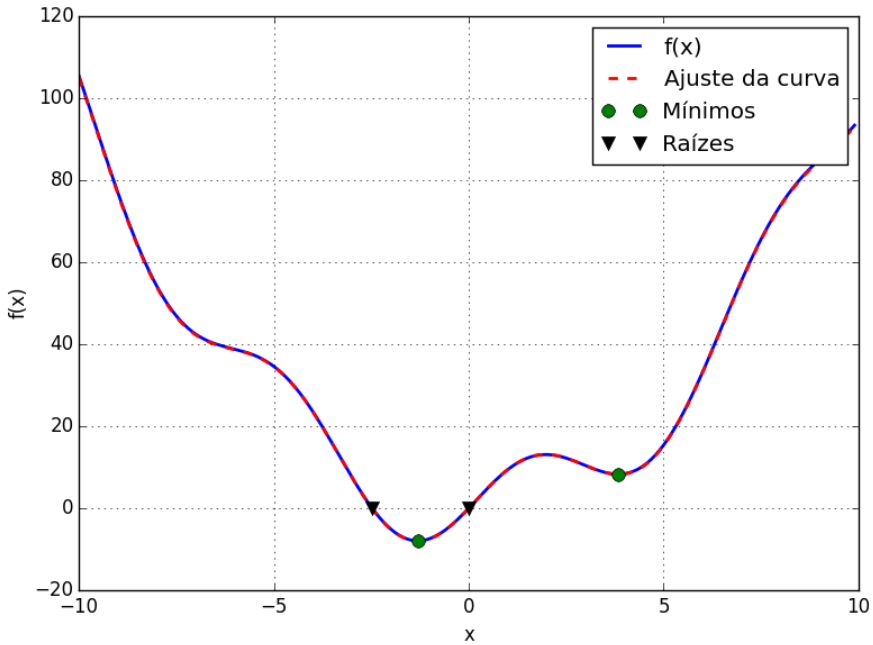
Otimização e ajustes

```
>>> import numpy as np
>>> from scipy import optimize
>>> import matplotlib.pyplot as plt

>>> def f(x):
...     return x**2 + 10*np.sin(x)
>>> x = np.arange(-10, 10, 0.1)
>>> grid = (-10, 10, 0.1)
>>> xmin_global = optimize.brute(f, (grid,))
>>> xmin_local = optimize.fminbound(f, 0, 10)
>>> root = optimize.fsolve(f, 1)
>>> root2 = optimize.fsolve(f, -2.5)

>>> def f2(x, a, b):
...     return a*x**2 + b*np.sin(x)
>>> xdata = np.linspace(-10, 10, 20)
>>> ydata = f(xdata) + np.random.randn(xdata.size)
>>> chute = [2, 2]
>>> param, param_covar = optimize.curve_fit(f2, xdata, ydata, chute)

>>> fig = plt.figure()
>>> ax = fig.add_subplot(1, 1, 1)
>>> ax.plot(x, f(x), 'b-', lw=2, label="f(x)")
>>> ax.plot(x, f2(x, *param), 'r--', lw=2, label="Ajuste da curva")
>>> xmins = np.array([xmin_global[0], xmin_local])
>>> ax.plot(xmins, f(xmins), 'go', ms=8, label=u"Minimos")
>>> roots = np.array([root, root2])
>>> ax.plot(roots, f(roots), 'kv', ms=8, label=u"Raizes")
>>> ax.legend()
>>> ax.grid()
>>> ax.set_xlabel("x")
>>> ax.set_ylabel("f(x)")
>>> plt.show()
```



Tópicos

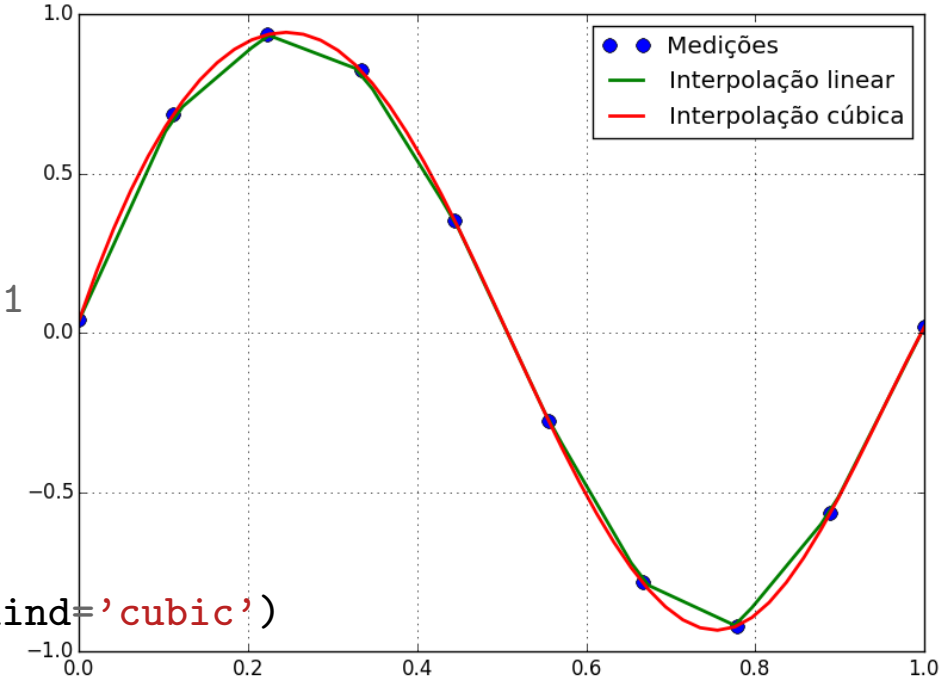
Interpolação

- A função `interpolate.interp1d` retorna uma função de interpolação:

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

t_med = np.linspace(0, 1, 10)
noise = (np.random.random(10)*2 - 1)*1e-1
y_med = np.sin(2*np.pi*t_med) + noise

interp_linear = interp1d(t_med, y_med)
t_calc = np.linspace(0, 1, 50)
y_linear = interp_linear(t_calc)
interp_cubica = interp1d(t_med, y_med, kind='cubic')
y_cubica = interp_cubica(t_calc)
```




```
plt.plot(t_med, y_med, 'o', ms=8, label=u'Medicoes')
plt.plot(t_calc, y_linear, lw=2, label=u'Interpolacao linear')
plt.plot(t_calc, y_cubica, lw=2, label=u'Interpolacao cubica')
plt.legend()
plt.grid()
plt.show()
```

Tópicos

Material do curso

Aulas do Lázaro Camargo (INPE):

https://x4p8nx.s.cld.pt

material-python-03.tar.bz2

```
matplotlib_aula_03_histogramas.pdf
matplotlib_aula_04_graficos_barras.pdf
matplotlib_graficos_interativos.pdf
scipy_aula_01_introducao.pdf
scipy_aula_02_algebra_linear.pdf
```